

Publishing and Interacting with Linked Data

Roberto García, Josep Maria Brunetti, Antonio López-Muzás, Juan Manuel Gimeno, Rosa Gil

Universitat de Lleida

Jaume II, 69

25001 Lleida, Spain

+34 973702742

{rgarcia, jmbrunetti, jmgimeno, rgil}@diei.udl.cat, lopezmuzas@gmail.com

ABSTRACT

In order to make a Semantic Web dataset more usable to a wider range of users, specially Linked Data ones, Rhizomer constitutes a tool for data publishing in the web that in addition to common data browsing mechanisms based on HTML rendering, provides a set of components that facilitate awareness of the dataset at hand borrowed from Information Architecture. Rhizomer automatically generates navigation menus taking into account the ontologies used by the dataset and facets based on how properties are instantiated for each of the classes in the dataset. This makes it possible for users to easily be aware of the main kinds of things in the dataset but also their main properties and the values the take while they perform faceted navigation. These generic IA components are complemented with specialised interaction services that can be dynamically deployed and associated to resource using semantic web services. Among these services, Rhizomer features one that provides simple edition of the data using autocomplete forms guided by the ontologies used in the dataset and the available resources.

Categories and Subject Descriptors

H.3 [Information Storage and Retrieval]

H.5 [Information Interfaces and Presentation]

H.4 [Information Systems Applications]

General Terms

Management, Documentation, Design, Experimentation, Human Factors.

Keywords

Ontology, Semantic Web, Linked Data, metadata, human-computer interaction, usability, visualisation.

1. INTRODUCTION

The amount of semantic data available in the Web is increasing at a really good pace in the last years, as it is shown by initiatives like Linked Open Data (LOD). The cloud of interrelated and open datasets included in the LOD cloud has

rapidly evolved, from the 2 billion triples and 30 datasets one year after its creation in February 2007, to more than 25 billion triples and 200 datasets in September 2010 [1].

The potential of this huge amount of data is enormous but it is not being fully realised as end-users, non-linked data experts, find a great barrier when facing it. The barrier is that most of this data is available as raw data dumps or SPARQL [2] endpoints.

For data dumps, it is really complicated to realise what data does one have at hand, what it refers to and what kind of terms are used. And it requires some experience in Semantic Web tools in order to do those.

For SPARQL endpoints, the amount of work required for grasping the internalities of the data set might be reduced. Besides, a good knowledge of SPARQL is required in order to generate and understand a set of queries that allow realising how big the dataset is, which are the main kinds of things, how are they interrelated, etc. And in any case, the results from the queries are not very usable, list of URIs and appearances counts.

The best approach to make a dataset more usable to a wider range of users is to use some sort of data publishing tool. At least, this kind of tools usually provides an HTML rendering for each resource in the dataset. Each HTML page lists all the properties for the corresponding resource. Pages are interlinked based on the connections among resources in the underlying RDF graph and the user can follow HTML links to browse through the graph.

However, this feature is only useful if the user has some a priori knowledge about the dataset, especially the URI of a given resource. There is no way to get at least an overview of the kind of resources in the dataset. Some data publishing platforms like OpenLink's Virtuoso do provide a faceted view on a specific subset of the data, but in order to get it, it is necessary to provide an URI or some keywords for textual search.

Consequently, existing tools make very difficult, especially for an user that deals for the first time with a dataset, to realize what kind of resources there are, what properties they have and how they are related.

Our proposal is to draw from the great experience accumulated in the Information Architecture (IA) domain [3] and reuse and adapt existing IA components to provide this kind of information to users. This kind of components is well known to Web users, as they are present in almost any web page. They are navigation bars, navigation facets, sitemaps, breadcrumbs, etc.

However, as they just provide a generic but fixed way of visualising and navigating data, and because we are dealing with highly heterogeneous data, we complement these IA

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WIMS'11, May 25-27, 2011 Sogndal, Norway

Copyright © 2011 ACM 978-1-4503-0148-0/11/05...\$10.00.

components with a framework that allows dynamically deployment and linkage of specific interaction plugins. These plugins are Semantic Web services that receive RDF data and provide specific ways of visualisation and interaction with this data. They are dynamically associated to the resources they can process based on semantic expressions.

Despite there are already systems that provide this kind of specialised interaction services depending on resource characteristics, such as maps, timelines or plots, our approach is to provide a framework that facilitates their integration and association to resources in a completely dynamic way.

Overall, the objective is to build a tool, called Rhizomer, that can be deployed on top of any dataset based on Semantic Web technologies and publish it, while facilitating user awareness of what is in there. This awareness is accomplished by IA components like navigation bars, which show the main kinds of resources in the dataset, or facets, that show the more significant properties for different kinds of resources and their values. Moreover, it is possible to deploy Semantic Web services that provide specialised ways to interact with the data and analyse it. We even provide an interaction service that allows performing simple edition of the data using autocomplete forms guided by the ontologies used in the dataset and the available resources.

In conjunction, the platform facilitates publishing and browsing a dataset, like many other similar tools, but also allows that users realise what is the value of the dataset in the context of their particular needs. Consequently, it contributes to the adoption of the Semantic Web by raising the awareness of the usefulness of the many datasets currently available and thus motivating the development of specific tools that profit from them.

The rest of this paper is organised as follows. First, the related work is presented in Subsection 1.1. Then, Rhizomer is introduced in Section 2 and its Information Architecture components are detailed in Section 3. Finally, the future plans and the conclusions are presented in Section 4.

1.1 Related Work

The first tool that comes to mind when trying to realise what a dataset is about are Semantic Web browsers. They are especially useful when dealing with a dataset published as Linked Data because they provide a smooth browsing experience through the graph. However, they just provide a view of a resource, or set of resources, and the properties coming directly out, and eventually also coming in, the resource, e.g. Disco [4].

However, Semantic Web browsers do not provide additional support for getting a broader view of the dataset being browsed, just a view on the current resource or at most of the steps followed so far using something similar to breadcrumbs. In some cases it is also possible to get more informative components like facets but not as part of a generic browser, just for a given dataset as in the case of the DBPedia Faceted Browser [5] or \facet [6].

Another interesting feature of many Semantic Web Browsers is that, in addition to the rendering of properties and values for a resource, they provide specialised visualisations like maps for geo-located resources or timelines for time-framed ones, e.g. Tabulator [7].

However, all browsers we are aware of provide a closed set of such visualisations and adding more of them requires changes in the source code. Finally, it is important to note that this kind of tools makes it possible to browse datasets previously published as Linked Data. If the dataset is just available as a dump file or SPARQL endpoint, then some extra tools or a different approach is needed.

Explorator [8] is a tool that makes it possible to browse a dataset available as a SPARQL endpoint. Though Explorator makes it possible to browse the dataset by combining search, facets or operations on sets of resources, it makes it also difficult to get a broader view on the dataset other than a list of all the classes or properties used. The same applies to the Information Workbench [9]. This tool provides faceted navigation of query results using Microsoft PivotViewer¹ but does not provide any mechanism that provide a broader view on the dataset.

Other alternatives are Content Management Systems (CMS) or Wikis with semantic capabilities. Some mainstream CMSs and wiki systems have started to incorporate semantic technologies. The most significant case are the Drupal RDF Modules². These modules (RDF API, RDF DB, RDF Export, RDF Import and RDF Schema) are extensions to the basic Drupal functionality that provide features such as RDF semantic metadata storage, querying, importation or simple rendering as a table. However, semantic CMSs, like Drupal or ODESeW [10], are intended more for content creation than for the importation and publication of existing. Consequently, they do not provide features for facilitating access to the imported data.

The same applies to semantic wikis, such as the semantic extension for MediaWiki. This extension, called Semantic MediaWiki [11], makes it possible to mix wiki mark-up with semantic annotations. As in the case of CMSs, it is also possible to import existing data but the wiki does not provide mechanisms that facilitate user awareness about the structure of the data that has been imported.

Finally, there are some tools that are specialised, to some extent, in publishing Linked Data. The aspect that we consider here is the kind of support they provide to users when accessing the dataset and try to get an idea about what it is about.

One kind of these tools is the one that publishes relational databases as linked data, like D2R Server [12] or ODEMapster [13]. However, they provide the same kind of view on the published data that Semantic Web browsers provide, i.e. resources and their associated properties but no general view on the dataset.

There are also specialised tools that publish existing datasets or SPARQL endpoints as Linked Data. Paget is a framework for building linked data applications. At the moment it is focussed on publishing data but the intention is that it is capable of managing updates too. It is resource-centric and data driven. From the RDF data describing resources identified by their URI it generates different representations (RDF, HTML, JSON and Turtle) using content negotiation.

Pubby is similar to Paget. It builds a Linked Data frontend for SPARQL endpoints with dereferenceable URIs for the resources in the endpoint and content negotiation. It also features a

¹ <http://www.microsoft.com/silverlight/pivotviewer/>

² Drupal RDF modules, <http://drupal.org/node/222788>

metadata extension that provides provenance information. However, in both cases, the frontends they provide are like those provided by Semantic Web browser.

To conclude, it is also possible to consider platforms for semantic data storage and publishing like Talis Platform³ or OpenLink Virtuoso⁴. In both cases, in addition to the data stores, there is an HTML frontend for the datasets beyond SPARQL. However, like with previous tools, the support for broader awareness of the dataset structure is very limited.

The most significant contribution is in OpenLink Virtuoso, which provides a faceted view on a specific subset of the data, but in order to get it, it is necessary to provide an URI or some keywords for textual search. Consequently, the facets view is limited to the resources retrieved from a previous search and there is no way to previously get an overview of the kinds of resources in the dataset.

2. RHIZOMER

First of all, Rhizomer⁵ is based on a simple architecture that makes it flexible, scalable and capable of adapting to different deployment and use scenarios. Its core is rooted on simple HTTP mechanisms and follows a REST approach [14]. Rhizomer also implements content negotiation taking into account the requested content type thus providing the requested data in the desired format.

Each resource is managed through the URI referencing where it is published, thus basing the whole system on a Resource Oriented Approach. The basic HTTP commands allow managing each resource: GET retrieves the semantic data associated with the resource in the requested format, PUT updates the data for the resource with the submitted one, POST creates a new resource with the submitted semantic description and DELETE removes the specified resource and the corresponding data.

An alternative to this REST approach is to use SPARQL Update⁶ as the way to insert, update and delete. However, due to the lack of maturity of the update language, which is currently being standardised, and the benefits of a REST approach [15], especially in relation with facilitating the integration with external services, we have adopted this approach as the basic way to manage the stored data.

In any case, the GET command is also used to pose semantic queries based on the SPARQL query language [16] like in any SPARQL endpoint. Consequently, it is also possible to use this approach to benefit from SPARQL Update, which might be more convenient when various resources are updated simultaneously.

All the previous HTTP commands, and the SPARQL queries, are then forwarded to the underlying data store, see Figure 1. Currently, Rhizomer integrates connectors for Jena and Virtuoso. These connectors make it possible to implement all the data management operations, especially updates, using the REST mechanism independently of the SPARQL Update availability.

³ <http://www.talis.com/platform>

⁴ <http://virtuoso.openlinksw.com>

⁵ <http://rhizomik.net/rhizomer>

⁶ <http://www.w3.org/TR/sparql11-update>

Rhizomer also has a generic connector for any repository providing a SPARQL interface. In this case, if the repository offers a SPARQL Update implementation, it is also possible to perform insertions, updates and deletions through the Rhizomer REST interface. All this functionality is encapsulated in the server part of the Rhizomer tool.

On the other hand, the client-side functionalities have been developed with the aim of improving the usability of the user interface. They are deployed in the user's browser and implemented using JavaScript and asynchronous HTTP calls (AJAX [17]), though most of the functionality is also available without JavaScript in order to improve accessibility [18].

All the interactions with the user are built on top of the REST operations. However, the RDF syntax of semantic data is completely hidden in order to increase usability. Like many Semantic Web browsers or data publishing tools, Rhizomer provides an HTML view on the data that also facilitates the navigation across the data graph, as detailed in Section 3.1.

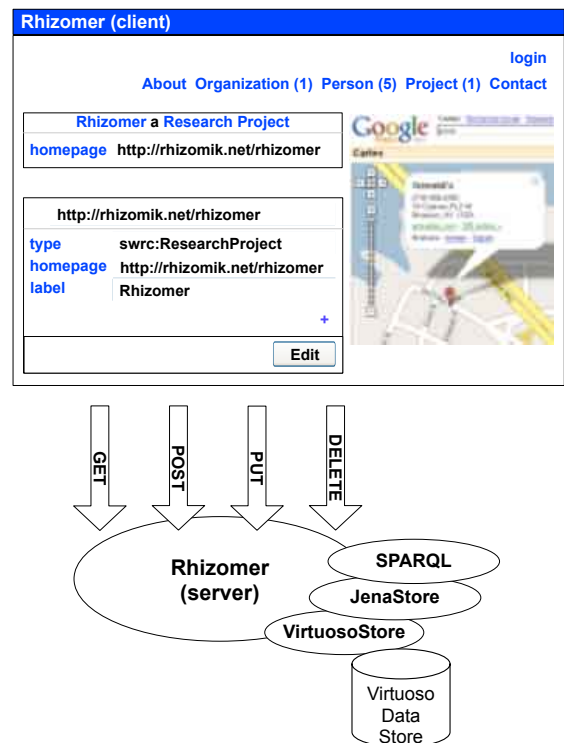


Figure 1. Rhizomer architecture overview

However, as it has been shown in the related work section, this approach does not contribute towards an awareness of the overall structure of a dataset. In order to provide such functionality, Rhizomer features a set of components inspired by those common in Information Architecture. The added value is that in the case of Rhizomer, these components are automatically generated and updated from the data and ontologies in the published dataset, as described in Section 3.

Besides, Rhizomer incorporates Semantic Web services providing specialised interaction means beyond generic browsing. Each user action corresponds to a Semantic Web service whose description incorporates the constraints a resource must satisfy to be a valid input for the service. Consequently, the semantic description of a resource determines which actions can

be applied to it. In Section 3.5 this mechanism for dynamically associate resources to interaction services will be completely described.

The previous mechanisms for generic data browsing, awareness of the data structure and specialised interaction for data analysis constitute the mechanisms that Rhizomer offers to publish data in a way that allows users realizing their potential value. However, in many cases, data is generated through not completely reliable mechanisms that introduce different kinds of errors or lacks.

The awareness mechanisms facilitate the detection of these errors and, in order to mitigate the reduction in the value perceived by the user when detecting them in a dataset, Rhizomer also features a mechanism for data edition. Edition is implemented through HTML forms with autocomplete that assist the user during the edition process. Properties and values are recommended taking into account what the user types and the data and ontologies in the dataset, as it is detailed in Section 3.6.

3. INFORMATION ARCHITECTURE COMPONENTS

Although the semantic query forms make it easier for users to query a given dataset taking profit from its semantic structure, user tests show that this is not the more convenient way of making users interact with a dataset, specially when it is the first time they face it.

When users interact with a dataset with an unknown structure, they require mechanisms that show the underlying structure more clearly than simple keyword-based forms. Even the semantic form presented in the previous section, though they show the more relevant properties for a given kind of resources, require a way to make the user aware of the kinds of resources available in the dataset.

When evaluating different mechanisms to make users aware of the structure of the information they are facing through web pages, we came to the Information Architecture (IA) discipline [3] and the huge amount of experience it has accumulated about how to structure a web site to make it easier for users to have access to the information it contains.

Information Architecture identifies four kinds of systems:

- Organisation systems: they allow presenting information in different ways, following different schemas that make it possible to group or differentiate information using different criteria, like chronological or alphabetic order.
- Navigation systems: they help users move across the available information. For instance, there are navigation bars or site maps.
- Labelling systems: they describe categories, options and links using terms that are meaningful for users. They are all around the information architecture of a site, even as part of other systems, e.g. navigation bars labels.
- Search systems: they allow users to search specific information chunks based on some sort of keywords. They also offer mechanisms to restrict the search space.

In the context of an information architecture rooted on semantic data, it is quite natural to develop organisation system components that profit from the underlying ontologies and

schemas. These components tend to be quite specific to the criterion used for information organisation and, as it is shown in Section 3.5, we have faced the integration of such specific components into Rhizomer using a framework for the deployment of these components based on web services.

Another way to make users benefit from the many ways semantic data can be organised is through facets. These information architecture components are part of the navigation system and our approach to deploy them on top of semantic data is presented in Section 3.4.

However, facets are useful when the user does already have focused on some particular kind of resources and the many ways to organise them using the properties they have in common are shown as facets. Before this faceted navigation is performed, a more general view on the dataset is required. The best candidates for this are global navigation systems.

Global navigation systems typically take the form of navigation bars that, in the case of web sites, are present in all pages. They provide a view of the main kinds of things covered in an information system. The rest of the IA systems have been also considered in the context of Rhizomer. Labels, as in the case of generic data browsing presented in Section 3.1, and search, as it is shown in Section 3.2.

The drawback of all these IA systems is that they are quite expensive to develop and maintain. Fortunately, when they are built on top of the highly structured data typical in the Semantic Web and Linked Data, it is possible to automate most of the development and maintenance work.

Section 3.3 details how global navigation menus are automatically created and maintained in Rhizomer starting from the underlying ontologies and schemas and how they are structured and instantiated. A similar approach is taken for generating facets for each kind of resource and also for prioritising them taking into account their “utility”, as detailed in Section 3.4.

3.1 Generic Data Browsing

The client part of Rhizomer provides a generic HTML view of the data retrieved when performing a GET operation on a resource. This is a typical feature of Semantic Web browsers and many Linked Data publishing tools that facilitates the interaction with users. However, in the Rhizomer case, this view introduces some particularities.

First of all, the HTML rendering for the data associated to a resource in Rhizomer includes both HTML and RDFa. This makes it possible to publish data using diverse methods in order to improve the visibility of the data. Rhizomer provides the data for a resource as RDF if this is the requested content type but even if the request is for HTML, it is possible for machines to retrieve the original data from the HTML rendering thanks to the embedded RDFa.

The rendering also tries to facilitate things for human users. First of all, the HTML features links for all resources and properties. This is common for resources but not so much for properties. The intention is to make it possible for users, when they are not sure about the intended meaning of a property, to click on the property and get all the data for the property that usually include comments that might be helpful.

Another feature geared towards improving the usability of the HTML rendering is that all URIs are replaced with their labels, when they are available. Moreover, the language tag associated to the labels is taken into account so it is easy to implement a multilanguage interface just by providing language tagged label for resources.

The objective is to avoid the clutter that showing the whole URI in the interface introduces. Consequently, if there is no label, the fragment or the last part of the URI is used instead. In order to avoid the ambiguity that this might introduce, if different URI share the same fragment, when a user passes the mouse over the link the whole URI is shown.

In addition to all the labels for the resources and properties that appear in the data for the resource being browsed, we also consider appropriate to include all the data for the anonymous resources that are mentioned. We do so to avoid the lack of context resulting from showing an anonymous resource isolated from the identified resources that might refer to it. For instance, an address resource is usually modelled as an anonymous resource, cf. the vCard specification⁷, and it is more informative for the user to display it together with the resource that is located at that address.

These requirements, the presence of labels for are resources and properties and the inclusion of the data for the anonymous resources, have motivated that we employ a slightly different approach when building the data fragments that are retrieved when the data for a resource is requested.

The more common existing approaches are to include all the triples that have the requested resource as subject or to include all the triples that have it as either subject or object. None of these approaches includes the labels for other resources that the requested one neither the data for the anonymous resources.

For the later, there is the alternative to use Concise Bounded Descriptions (CBD) [19]. This approach does include all the data for the anonymous resources referred from the requested resource but lacks the labels for the other resources and properties. Consequently, we have specified and implemented a custom way of building the data fragments for data browsing. It is based on CBD and adds all the “rdfs:label” properties for all the resources and properties in a CBD fragment.

For instance, Figure 2 shows how an example graph would be fragmented following this approach. As it can be seen, there are two fragments, each one corresponding to an identified resource described by at least one triple, for which it is the subject. The first fragment describes <http://rhizomik.net/~rosa> and includes an anonymous resource for the address. The second one, for <http://www.udl.cat>, can be reached from the first one through a browsing step. Unlike the address, it is shown independently because it is not anonymous.

Finally, fragments are rendered using HTML, which is viewable using a web browser, a tool users feel comfortable with. In order to generate HTML from RDF, fragments are serialised as RDF/XML and transformed using an XSLT. The XSL transformation, which is part of the Rhizomer platform, guarantees consistent results whenever the input RDF/XML has been generated from fragments based on the Rhizomer approach.

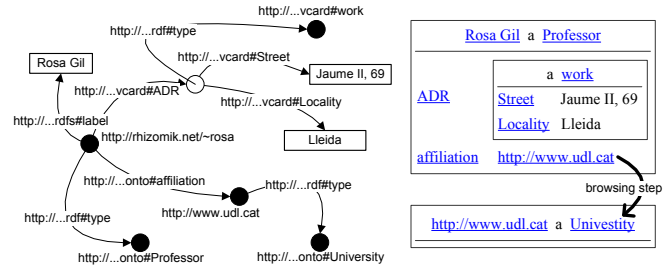


Figure 2. Fragmentation of an example RDF graph and the resulting HTML rendering

This mechanism has been implemented as successive DESCRIBE queries for the identified resource URIs to the SPARQL endpoint. The DESCRIBE operation of the SPARQL endpoint has been reimplemented in order to build the proposed fragments, i.e. CBDs plus all the involved labels. Labels are used, when available, in the place of resources URLs in order to make the HTML rendering more readable.

Then, the XSL transformation from RDF/XML to HTML is invoked from the client using AJAX, which is also responsible for sending the SPARQL queries and making the whole process go smoothly behind the scenes, making the user experience even more comfortable. Finally, the AJAX part of Rhizomer at the client also keeps track of the browsing steps so it is possible to use the "back" and "forward" browser buttons. Moreover, the browsing steps are cached at the browser in order to improve responsiveness.

3.2 Assisted Data Search

Rhizomer features the typical keyword based search but it also makes possible for users to pose queries that profit from the data structure. The biggest problem of semantic queries is that users should be aware of the query language and also of how data is structured. In order to avoid these problems, Rhizomer features semantic query forms that are generated taking into account the underlying ontologies and schemas: moreover, these forms, when submitted, are automatically translated into the query language, i.e. SPARQL.

Consequently, Rhizomer allows users to perform semantic queries without any knowledge of semantic query languages. All they need is to know how to fill query forms. These forms are generated dynamically from the kind of resource they are interested in, more concretely from the properties specific for that kind of resource.

The kind of resources the users is interested in will usually be selected from the global navigation bar that usually appears at the top of the page. There are more details about the navigation bar in Section 3. The resource kind corresponds to one of classes used in the dataset. It is used in order to query for all the properties whose domain is that class or any superclass of it, plus all the properties for which there is a restriction when the property applies to that class or its subclasses, e.g. the ontology statement *SubClassOf(Thesis AllValuesFrom(:author :Person))* is used in order to retrieve the “author” property as one applying to the class “Thesis”.

Following these criteria, the properties to be considered for semantic form generation are retrieved using the SPARQL query template shown in Table 1. This query retrieves all the

⁷ <http://www.w3.org/Submission/vcard-rdf/>

properties specific to a kind of resource, which are then used in order to generate a form with one input field for each property. The user can then fill the form in order to establish the search criteria and when the form is submitted the corresponding SPARQL query is generated. That query will retrieve all resources of type the initial class with the properties that have been filled in the form and whose values contain the input field filler.

Table 1. SPARQL query that retrieves the properties specific for a class

```
SELECT ?p
WHERE {
  { ?p rdfs:domain ?d.
    ?t rdfs:subClassOf ?d.
    FILTER (?t = [CLASS] && ?d != rdfs:Resource) }
  UNION
  { ?r rdf:type owl:Restriction.
    ?r owl:onProperty ?p.
    ?t rdfs:subClassOf ?r.
    FILTER (?t = [CLASS]) }}
```

Moreover, users can add other properties that, without being specific, might also apply to resources of that kind. These properties are retrieved using the SPARQL query shown in Table 2. This query retrieves all properties (RDF properties or OWL data-type or object-type properties) that are generically defined as having any resource as domain or that do not have any domain defined.

Table 2. SPARQL query that retrieves the properties specific for a class

```
SELECT ?p
WHERE {
  ?p rdf:type ?t.
  FILTER(?t = rdf:Property ||
    ?t = owl:DatatypeProperty ||
    ?t=owl:ObjectProperty)
  OPTIONAL {?p rdfs:domain ?d}
  FILTER(?d=rdfs:Resource || !bound(?d))
}
```

The resources selected by the query are presented together with their description, i.e. all the metadata describing them. To do that, the generated SPARQL query is a DESCRIBE one with the type and property restrictions for the filled form fields in the WHERE clause. The procedure is the one described in Section 3.1, the DESCRIBE query retrieves the CBD for the selected resources plus all the involved labels. Then, the retrieved RDF data is rendered as HTML. It is also possible, if the amount of resources or the amount of data for each resource is too big, to generate a CONSTRUCT query that just generates the basic data for each resource, e.g. types, labels and descriptions. If the user is interested in a particular resource in the query results listing, it is possible to click it and get the whole description using a DESCRIBE query for the resource URI.

As it has been shown in this section, the semantic forms profit from the underlying ontologies. However, in many cases the actual data does not follow the underlying ontologies, or there are not restrictive enough to retrieve the information required for semantic forms. In order to deal with these cases, we are planning to also use statistics about the data in order to guide semantic forms generation. This is currently future work but we are already performing this data analysis when generating the information architecture components detailed in the next section.

3.3 Navigation Menus

Navigation menus, in the case of website, let users navigate through different sections and pages of the site. They tend to be the only consistent navigation element, being present on every page of the site.

Traditionally, user-centred design techniques are used to develop the navigation menus of a site. The typical one is Card Sorting, where users are given a set of cards labelled with the main topics of the site and they group these cards following their own criteria. In order to generate menus as meaningful as possible for the broader range of users, the card sorting is repeated with different users.

This process requires a lot of time and effort from developers. Moreover, most of this effort is wasted as soon as the structure of the menu is established and fixed in a menu that becomes something static. If new kinds of items are introduced or a part of the content becomes more relevant, the Card Sorting should be repeated, at least in part.

The opportunity in the case of web sites build on top of semantic data is to automate part of the process of generation and maintenance of the navigation menus.

The objective is to generate a global navigation menu that takes into account all the classes considered in a dataset but also how they are instantiated. Consequently, if there are few instances of some classes or they are not instantiated at all, they should be less relevant in the menu bar. On the contrary, classes that do have a lot of instances should be shown prominently in the menu bar. This way the menu facilitates the access to the more significant classes but also makes it possible for new users to realise what are the main kinds of things in a dataset.

To do this, we use the Jena Ontology API to obtain a hierarchical list of domain classes and apply inference rules to get new relations between them. This list of classes is stored in a data structure and then used to generate the navigation menu. For each class we save information about the number of instances of the class, its URI, its labels and a list of its subclasses. Using this information it is possible to generate a hierarchical menu that represents all the classes of the domain.

This component can generate both global and local menus, i.e. a menu for the whole dataset or for a subset of it. The site administrator can also configure some parameters:

- The number of levels in the hierarchical menu.
- The number of items in each level of the menu.
- The order of items: alphabetical or by number of instances.
- A list of classes or namespaces to omit.

According to these parameters, this component generates the menu applying a recursive algorithm, shown in Table 3, that mainly performs two operations:

- Split classes with a large amount of instances in subclasses.
- Group classes with few instances in a superclass.

The algorithm starts with a Menu tree structure that initially contains the whole hierarchy of classes and the number of instances for each class. The first step of the algorithm is to remove all the empty classes that have zero instances. Then, depending on the number of intended items in the final menu,

i.e. parameter “numItems”, the algorithm performs mainly two operations:

- If the number of menu items is higher than the input parameter, the classes with fewer instances are grouped in a new class called "Other".
- If the number of menu items is smaller than the input parameter, the class with more instances is divided into subclasses.

Table 3. Overview of the navigation menu generation algorithm

```

generateMenu(Menu menu, int numItems)
{
    menu.removeEmpty();
    while(menu.size()>numItems)
    {
        Node other = menu.createOther();
        Node min = menu.getMinNode();
        other.mergeWith(min);
    }
    while(menu.size()<numItems)
    {
        Node max = menu.getMaxNode();
        menu.splitNode(max);
    }
}

```

These operations are recursively performed until the menu is completed. Figure 3 illustrates the process of generating the navigation menu for a subset of DBPedia, with 7 elements in the first level. In the original hierarchy there are only 3 classes in the first level. Therefore, there are 4 free spots in the menu. To cover these free spots, the algorithm identifies which classes are appropriate to divide, taking into account their number of instances and their number of subclasses.

At first, the Eukariote class is removed and its subclasses, Plant and Animal, move up to a higher level in the hierarchy. After this step, the navigation menu contains 4 elements: Plant, Animal, Bacteria and Archaea. From here, the algorithm is applied recursively until the menu is completely generated. In the next step, the Animal class is chosen and divided. However, in this case, there is not space for all its subclasses in the first level of the menu. For this reason, the subclasses with a higher number of instances move up to the main level of the menu while the rest of subclasses are grouped inside Other Animal.

It is important to note that the procedure depicted so far that takes into account the whole dataset classes and instances at a given moment and generates the corresponding menu as an static snapshot. In order to avoid repeating the whole process each time changes are performed on the dataset, it is possible for Rhizomer to monitor all changes to the dataset if they are performed through its interface and not directly on the store.

Whenever a change is detected, the records for all the involved classes are updated accordingly. If a new instance is inserted, all the classes it belongs to are updated and the new instances count is increased. Conversely, if an instance is removed, the instantiation counters for the involved classes are decreased. Finally, if the changes involve the classes themselves, the hierarchical structures among the class records are updated accordingly in order to model the new shape of hierarchy of classes.

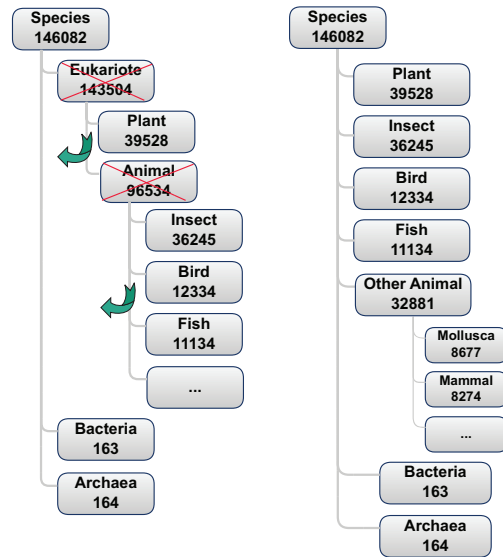


Figure 3. Generating a navigation submenu for DBPedia Species with 7 options (left original, right result)

This approach makes it possible to show the user the navigation bar that best fits the data in the dataset at that particular moment. For instance, if the dataset changes from containing mainly data about projects to mainly about publications, the menu would change accordingly to show more prominently the part of the underlying ontologies about publications. More concretely, initially the different kinds of projects were shown in the menu, as top or second level menu options. On the other hand, at most the option “Publication” was shown. With the data update, the menu bar would change and show just the “Project” option but include the main kinds of publications, especially those for which there are more instances.

On the other hand, one possible drawback of this approach, as it has been pointed by some usability expert evaluations, is that users find it very disturbing that the navigation menus change from visit to visit due to changes in the underlying data. This is an inconvenient effect of navigation menus dynamism, as users see them as a static part of the site and, as they get used to them, they rely on them as a handful guide to the site.

In any case, our experiments show that these changes are only systematic if there is very few data. Under those circumstances, the navigation menu undergoes changes quite often when adding new resources. However, as more resources are introduced, changes in the navigation menu tend to be minimal and as soon as the amount of data is statistically significant to keep the natural tendency in the dataset evolution, the changes in the menu bar are practically inexistent or not significant from the point of view of the user as they only affect to particular options in the submenus that are added or removed in the context of more general options in the menu, that keep users in the track to the information they need.

3.4 Facets

Users don’t always know exactly what they are looking for and, sometimes, they don’t even know what its name is. Other times, they are unfamiliar with the domain or they want to learn about a topic. This is particularly true when facing Semantic Web datasets. In these cases, exploratory search is a strategy that allows users to refine their search by successive iterations. An

exploratory interface such as faceted browsing allows users to find information without a priori knowledge of its schema.

With navigation menus we can make the user aware of the hierarchical structure of a dataset but, once they choose the class of things they are interested in, the face the barrier of not knowing how they are described. In other words, what are the main properties that describe them, which ones are the more relevant for that particular kind of things, the range of values they have in that particular case, etc.

Faceted navigation is an exploratory technique for navigating a collection of elements in multiple ways, rather than a single and pre-determined order. Facet browser interfaces provide a user-friendly way to navigate through a wide range of data collections. A faceted classification system allows contents to be classified in multiple dimensions. These dimensions are called facets and represent characteristics of the information elements. For example, a collection of books can be classified using an author facet, a subject facet, a date facet, etc. In the Semantic Web, expressed in RDF, resources constitute the collection of browsed elements and facets are the properties that describe them.

Traditional facet browsers relied on manual identification of the facets and on a previous knowledge of the target domain. In Semantic Web it is possible to automate this process and a semantic faceted browser should be able to handle any RDF dataset without any configuration. Since Semantic Web integrates data from lot of sources, we can't assume a single fixed schema for all data. A faceted browsing system in Semantic Web should be scalable and generic, not depending on a particular dataset.

In traditional Web, facet browsers are developed to navigate through homogeneous data and facets are fixed. This conflicts with Semantic Web, where data is too diverse to use a single set of facets: facets that make sense for one type of resource could be inappropriate for other types. Moreover, when new data is added the system should be able to add new facets at run time.

One of the most important aspects of a facet browser is that, when constraining the dataset, all properties and values that would lead to an empty set of results need to be automatically removed from the interface, protecting the user against dead ends.

To build the facets, and to keep them updated, what Rhizomer does is to perform SPARQL queries for each class in the dataset that retrieve all the properties their instances have, the different values each property has and the cardinality for each value, i.e. how many times that property for that class takes that value.

Facets are pre-calculated and stored in a data structure. They are then updated whenever the dataset is edited through Rhizomer. They are also updated, but just a local copy associated to a user session, when the user starts browsing and selecting values for different facets. In this case, the set of instances used for facets generation is constrained by the choices made so far and the facets are recalculated for that constrained set of instances. Those facets that are no longer relevant, i.e. no instance uses them, are removed from the facets set. For instance, if the value "2010" has been selected for facet "date", and for that date there is no instance in the dataset with the property "completedOn", then this facet will not be included in the set shown to the user.

When a dataset is very large and heterogeneous, the number of facets will also be very large. Therefore, it is needed an automated method to choose which facets are more useful and important for the user. We need to find those facets that best represent the dataset and those that are best to navigate the dataset. Choosing the right facets is very important, a suitable facet should allow efficient navigation through the dataset and be representative for those objects.

Faceted browsing can be seen as a decision tree. A path in the tree represents a set of constraints that select the resources of interest. As the tree is constructed dynamically and the information space changes, facets and their ranking need to be recalculated at each step of the decision tree.

To measure the quality of a facet, and therefore showing it more prominently to the user, we use three metrics:

- **Predicate frequency:** we are interested in those predicates that occur frequently inside the instances being browsed. The more resources covered by the predicate, the more useful it is in dividing the information space. If the predicate is not frequent it will only affect a small subset of the collection. We compute the predicate frequency as the number of resources for which the predicate p is defined. We normalise this value dividing it by the total number of resources:

$$freq(p) = \frac{n_r(p)}{n_r}$$

- **Predicate balance:** the facet helps the user better discriminate the set of instances being browsed when it takes a well-balanced range of values for the facet property. On the contrary, a facet whose property takes always or mainly a particular value is less useful. The same happens if each instance has a different value for the facet property. Consequently, we will favour facets that show behaviours in between these worst cases. To compute the predicate balance we use the Shannon's entropy formula:

$$H(S) = - \sum_{i=1}^n p(v_i) \log_n p(v_i)$$

- **Value cardinality:** a suitable predicate should have a small amount of values to choose from. If there are too many choices it is difficult to display all the options and it might confuse the user. We compute the value cardinality as the number of different values for a predicate. This metric is normalized using a function based on the Gaussian density that can be regulated through the μ and σ parameters to the top and bottom values of the range of different values we are interested in. This range is still to be fixed experimentally but existing work recommends ranges similar to from 2 to 20 [20].

$$card(p) = \begin{cases} 0 & \text{if } n_o(p) \leq 1 \\ e^{-\frac{(n_o(p)-\mu)^2}{2\sigma^2}} & \text{otherwise} \end{cases}$$

The three metrics are combined using a weighted function that produces a unique usefulness value for each facet. We are currently combining them with equal weights but we plan to explore different arrangement and test them with end-users.

Once the facets have been generated and prioritised given their usefulness, we are currently generating a simple HTML rendering of the facets that allow users to select individual values or range of them in the case of numeric values. Details about our future plans about facet rendering are available in Section 4.

3.5 Interaction Services

Navigation menus and facets are well suited for given users an overview of the kind of resources in a dataset and their properties. These components are quite generic and can be adapted to any kind of semantic data. However, they are so generic that they might miss the particularities of a given set of resources and how these particularities can be used to provide a better way to build a presentation for users or let them interact.

To provide these specialised interaction services when they are available, Rhizomer features a framework where they can be deployed and dynamically linked to resources. The linking is facilitated by the semantic description of the resources and also because the interaction services are wrapped as Semantic Web services that also feature a semantic description. Rhizomer uses these semantic descriptions to provide a completely dynamic integration of the interaction services because they are not preconfigured for a given type of resources, i.e. they can be seen as independent entities.

In order to reduce coupling, the semantic description of the service just needs to define the minimal restrictions on the resources it is capable of processing. For instance, a service that shows resources in a map is described semantically as requiring resources that have latitude and longitude attributes, or a geographic point property. This reduces coupling because the service does not get tied to a particular kind of resources, as it would happen if it was tied to resources of a particular type, e.g. Place.

Other parts of the semantic description of these visualisation services we have considered are the URI where it is available as a web service, the label to be used for end-user presentation and a characterisation of the output of the service that facilitates integrating the results back into the user interface.

Instead of developing a custom vocabulary for these descriptions, we have evaluated existing ones and chosen an ontology for web services description as the source of terms for our needs, concretely OWL-S 1.1 [21]. This was the most appropriate vocabulary for our needs. With OWL-S it was easier to detect the classes and properties more appropriate to the kind of descriptions we required and to use them in isolation without any concern about the rest of the framework. Only the *Service Profile* provided by OWL-S is used for a high-level description of the service.

In fact, only the class *Process* and the properties *hasInput* and *hasOutput* (defined in OWL-S) are used. *Process* allows identifying the resources that correspond to interaction services and the resource URI corresponds to where the web service is available.

The *hasInput* property is associated to *Process* resources and is used to characterise the resources that can serve as input for the interaction service. Here, we have explored two options. The first one is to make *hasInput* point to an OWL class that states the necessary and sufficient conditions to be fulfilled by the resources that constitute service input. Consequently, for an

interaction service to be available for a resource, this resource must belong to the class defined as the input of the service.

For instance, as it is shown in Table 4, it is possible to define *GeolocatedEntity* as the class of all the resources with properties *lat* and *long* and use it as the *hasInput* class for a service named “map”. There is no need to explicitly classify all the geolocated entities into this class. An OWL DL reasoner is responsible for classifying into it all the resources that satisfy these restrictions.

Table 4. Description of a geographical information visualization service using an OWL class

```
<rdf:RDF... xmlns:process=".../services/owl-s/1.1/Process.owl#"
  xmlns:pos="...w3.org/2003/01/geo/wgs84_pos#">
<process:Process
  rdf:about="http://rhizomik.net/rhizomer/services/map">
  <rdfs:label>map</rdfs:label>
  <process:hasInput>
    <owl:Class rdf:ID="GeolocatedEntity">
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="&pos;lat"/>
          <owl:minCardinality>1</owl:minCardinality>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&pos;long"/>
          <owl:minCardinality>1</owl:minCardinality>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </process:hasInput>
  <process:hasOutput>text/html</process:hasOutput>
</process:Process></rdf:RDF>
```

The drawback of using OWL classes to characterise the input of the interaction services is that, in order to decouple services from resources, it is necessary to use OWL classes with the necessary and sufficient conditions and OWL reasoner is required to infer the association. This is not feasible for many datasets due to scalability issues.

An alternative is to use SPARQL queries as the way to characterise the resources that might serve as input for a given interaction service. In this case, *hasInput* instead of pointing to an OWL class definition points to an ASK SPARQL query like it is shown in Table 5. This kind of queries return true if there is a data being queried satisfies all the restrictions posed by the query. The result is false otherwise.

Table 5. Description of a geographical information visualization service using a SPARQL query

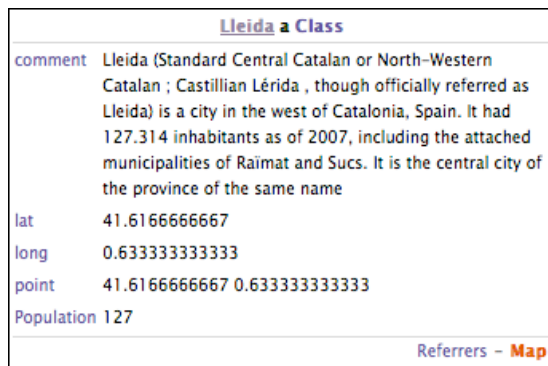
```
<rdf:RDF... xmlns:pos="...w3.org/2003/01/geo/wgs84_pos#">
<process:Process
  rdf:about="http://rhizomik.net/rhizomer/services/map">
  <rdfs:label>map</rdfs:label>
  <process:hasInput>
    ASK WHERE { ?r pos:lat ?lat; pos:long ?long }
  </process:hasInput>
  <process:hasOutput>text/html</process:hasOutput>
</process:Process></rdf:RDF>
```

The approach is to perform the ASK queries for the interaction services under consideration over each one of the resources the user is interested in. If the ASK query returns true, then it is possible to fetch the resource and its description to the interaction service. Once the mechanism to dynamically associate the interaction services and the resources described by a dataset is in place, it is time to define how the interaction

services are offered to the user. We currently consider two situations.

First of all, there might be interaction services that are the best choice for users when interacting with resources of a given kind or that feature some particular properties, e.g. resource of type *Picture* or with properties *lat* and *long*. In this cases, the services are marked as default and whenever a user navigates to a resource for which such a default service exists, the service is used to generate the visualisation of the resource instead of the visualisation for generic data browsing presented in Section 3.1.

The second way to offer an interaction service to users is when the service is not considered the best choice but just an alternative way. In this case, while the is viewing a given resource using the generic browsing mechanism or another interaction service, the alternative interaction services are shown to the user as links the user can activate in order to use the corresponding service to interact with the resource. For instance, in the resource labelled “Lleida” features latitude and longitude so the “Map” interaction service is offered as an alternative way to interact with the resource.



Lleida a Class	
comment	Lleida (Standard Central Catalan or North-Western Catalan ; Castillian Lérida , though officially referred as Lleida) is a city in the west of Catalonia, Spain. It had 127.314 inhabitants as of 2007, including the attached municipalities of Raimat and Sucs. It is the central city of the province of the same name
lat	41.6166666667
long	0.633333333333
point	41.6166666667 0.633333333333
Population	127
Referrers - Map	

Figure 4. Generic view of a resource with latitude and longitude with a “Map” alternative interaction service

It is important to note that the same interaction service might be the default for some resources and alternative for others. Consequently, we have introduced a new property called *hasInputDefault* that specifies the resources for which the service is the default while the original *hasInput* is kept for the resources for which it is alternative. In both cases, the cardinality might be greater than one if more than one OWL class or SPARQL query is necessary to specify the resources the interaction service applies to. Alternatively, just one OWL class or the SPARQL query might be defined combining different patterns for the different kinds of resources, e.g. using the UNION clause to combine the patterns in the case of SPARQL queries.

It is important to note that we have not considered any particular property for the service description that specifies the input parameters for the service, e.g. a “lat” and “long” parameters in the case of the “Map” service. Instead, we consider that all services receive the input via a POST message and that the payload of the message is the RDF/XML serialisation of the description of the resource (or resources) that the service receives as input.

This approach reduces the coupling between services and resources and makes it possible for the interaction services to benefit from the additional properties in the resource

description. For instance, the service might use the available labels in order to build a more appealing visualisation for users that avoids showing them URIs and complement it with descriptions, abstracts, etc.

The drawback of this approach is that the direct invocation of web services passing them RDF metadata as input is not common. Therefore, in many cases, the URI associated with a service is actually pointing to a wrapper that receives the RDF, extracts the data needed by the service, and makes the “real” invocation of the service. This additional layer between Rhizomer and the services, though it complicates the implementation, allows using visualisation services such as GoogleMaps or SIMILE Timeline⁸ that are only available as JavaScript libraries. In this case the wrapper is implemented as a servlet that generates the web page that uses the JavaScript library and provides the final result.

Finally, the *hasOutput* property specifies the output type of the service, i.e. the MIME type of the output. The output is shown in a new HTML layer within the Rhizomer interface and the MIME type is used to correctly interpreting the result.

3.6 Data Edition

One additional feature that has been added to Rhizomer is the possibility of directly editing the data through the Rhizomer interface. This has been identified as a valuable feature because all the information architecture components described so far, in addition to improve the awareness of the structure of the dataset, also make errors in the data more evident.

This likely produces that the user perceives the dataset as less valuable and the objective is to make these small editions to correct errors observed in the data while browsing easy to fix.

Currently, this feature is limited to authorised users but the objective is to make it open to all users and incorporate the trust management mechanism that facilitate integrating the proposed changes in a controlled way.

Consequently, the target user is a non-expert end-user and the kind of editions supported are small changes or the creation of new data based on the existing one. It is not intended as an ontology editor like Protégé [22]. We profit from this setting and take into account usability evaluation outcomes, like those reported in [23] for Linked Data authoring tools.

Edition is implemented through HTML forms with autocomplete that assist the user during the edition process. Properties and values are recommended taking into account what the user types and the data and ontologies in the dataset.

The approach for fragmenting the graph while browsing based on CBD plus labels, presented in Section 3.1, besides being the foundation for browsing, allows constraining the metadata editing and deletion actions to a limited set of triples. This way, it is possible to implement editing actions as the replacement of a given fragment with the one resulting from the editing process. The same strategy applies for the deletion action.

All these operations are also carried out through an HTML interface. In addition to the RDF to HTML transformation, the Rhizomer platform includes an XSL transformation from RDF to HTML forms. These forms are generated automatically from the RDF/XML corresponding to a fragment.

⁸ Simile Timeline, <http://simile.mit.edu/timeline>

This transformation, instead of generating text values and links for literals and resource, generates input fields for each triple. The field is named using the corresponding property URI and its value corresponds to the triple value. The fields can be used in order to edit the property value, either a resource URIs or a literal. Moreover, properties and values can be removed or added.

The user is assisted during the editing process using an autocomplete feature. Basically, when the user chooses to add a new property, a SPARQL query is used in order to retrieve all the “recommended” properties for the resource being edited whose label or part of the URI start with the text typed so far by the user.

These come out the set of properties not constrained to a particular resource type, i.e. no domain restriction, plus those constrained to the types of the resource being edited, i.e. those whose domain is one of the type of the resource being edited or those the resource is instance of an OWL Restriction on the property.

The SPARQL queries to do this are the same than when building a search form for a given resource type like described in Section 3.1. The only addition is the additional filters on the label of the property or its URI containing the chars typed so far by the user in the input field, and from which the autocomplete is performed.

The properties specific of a given class are retrieved using the query at Table 1 and the generic properties, applicable to any kind of resource, with the query at Table 2. The later are just retrieved once per user session and cached because it is the same set for all resources. It is important to note that in this case, we combine the properties for all the classes the edited resource is an instance of.

Once the user has added a new property, or if the value of an existing property is being edited, Rhizomer also provides assistance while defining property values that are not literals. In this case, the extra guidance is provided by the range of the property whose value is being edited. If the range is a literal, the user can type freely the property value. If it is a range, then the value is an instance of the range class.

Here the term “range” generalises both the *rdfs:range* of the property but also the constraint that an OWL Restriction puts on the values of a property in the context of a class the edited resource instantiates. This constraint is defined by an *owl:allValueFrom* or *owl:someValuesFrom* OWL primitive which points to a class that values of the property respectively should or might instantiate.

Consequently, the autocomplete feature for property values is implemented using SPARQL queries against all the resources in the repository whose label or URI contains the text typed by the user so far. Moreover, those resources should be instances of the ranges of the property whose value is being edited.

However, the user might type something and there might not be anything in the dataset with that label or URI. In this case, the outcome is that a new resource is created. The resource is type with the “range” of the property and labelled with the string typed by the user. From this point, the user can start adding new properties to the new resource.

Finally, an algorithm has been developed in order to reverse the mapping from RDF to HTML forms. In other words, this

algorithm is responsible for generating the RDF that results from the editing process by mapping the form input fields to the corresponding RDF triples.

There is an input field, generated during the RDF to HTML form step, which stores the URI of the resource being edited. This one becomes the subject for all the RDF triples generated from that form. The input field identifiers and their fillers become the subjects and objects for that triples describing the subject resource. Fig. 1 shows a graphical representation of the whole edition process.

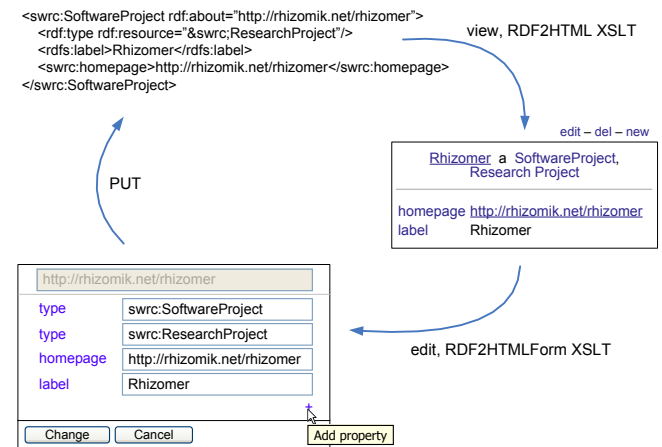


Fig. 1. Transformation from RDF to HTML form and back to edited RDF

4. CONCLUSIONS AND FUTURE WORK

As it has been shown, Rhizomer implements a set of features that make it possible to deploy it on top of any dataset based on Semantic Web technologies and publish it, while facilitating user awareness of what is in there. This awareness is accomplished by components borrowed from the Information Architecture discipline. Concretely, navigation bars, which show the main kinds of resources in the dataset, and facets, that show the more significant properties for different kinds of resources and their values. Moreover, it is possible to deploy Semantic Web services that provide specialised ways to interact with the data and analyse it. We even provide an interaction service that allows performing simple edition of the data using autocomplete forms guided by the ontologies used in the dataset and the available resources.

Our preliminary tests with users show that Rhizomer facilitates publishing and browsing a dataset, like many other similar tools, but also allows that users realise what is the value of the dataset in the context of their particular needs. It has also shown its scalability from small datasets like the one for the Rhizomik initiative⁹ to really big ones like the whole DBpedia¹⁰, both datasets can be accessed online through Rhizomer at the provided URIs.

The user tests are currently just preliminary qualitative evaluations of the resulting information architecture. We will start a quantitative evaluation when we finish developing a

⁹ <http://rhizomik.net>

¹⁰ <http://rhizomik.net/dbpedia>

reference IA we can compare to. We are currently developing an IA following “traditional” techniques for the DBpedia dataset.

The main technique is Cardsorting [3], which is based on providing to some test users a set of cards that they then organise, independently, into groups of cards they find strongly connected. These groups are then used when building the navigation bar menus. In order to develop a comparable information architecture, the cardsorting is being conducted on the set of cards corresponding to the top levels of the DBpedia Ontology.

Other areas of future work are related with facets and interaction services. For facets, the objective is to generate facets customised to the kind of values being managed, i.e. numerical values, alphabetical values, dates, geographical points, etc. We are also experimenting with different ways of combining the three metrics used to rank the facets. And, it would be also interesting to consider not only the statistical value of each facet but also their descriptive value.

In relation with interaction services, the idea is to profit from the easy integration of external services to explore more appropriate interaction and design patterns together with the more useful information visualisations that facilitate analysing heterogeneous semantic data.

To conclude, the edition service open the door to the integration of authentication and trust management mechanisms that allow building reliable data curation communities around datasets published using Rhizomer. In this case, the objective is to make it open to all users to contribute and incorporate the trust management mechanism that facilitate integrating the proposed changes in a controlled way.

5. ACKNOWLEDGMENTS

The work described in this paper has been partially supported by Spanish Ministry of Science and Innovation through the Open Platform for Multichannel Content Distribution Management (OMediaDis) research project (TIN2008-06228).

6. REFERENCES

- [1] <http://linkeddata.org>
- [2] <http://www.w3.org/TR/rdf-sparql-query>
- [3] Morville, P. and Rosenfeld, L. 2006. Information Architecture for the World Wide Web. O'Reilly Media.
- [4] Bojars, U., Passant, A., Giasson, F., Breslin, J. G.: “An Architecture to Discover and Query Decentralized RDF Data”, in : Proceedings of Workshop on Scripting for the Semantic Web, SFSW 2007. CEUR Workshop Proceedings, vol. 248 (2007)
- [5] Rasmus Hahn, Christian Bizer, Christopher Sahnwaldt, Christian Herta, Scott Robinson, Michaela Bürgle, Holger Düwiger, Ulrich Scheel: Faceted Wikipedia Search. 13th International Conference on Business Information Systems (BIS 2010), Berlin, Germany.
- [6] Hildebrand, M., Ossenbruggen, J., and Hardman, L. 2006. /facet: A Browser for Heterogeneous Semantic Web Repositories. The Semantic Web - ISWC 2006. Springer. 272-285.
- [7] Berners-Lee, T. et al. 2006. Tabulator: Exploring and Analyzing linked data on the Semantic Web. Proceedings of the Semantic Web and User Interaction Workshop (SWUI'06), Athens, USA.
- [8] Araujo, S., Schwabe D., Barbosa S. 2009. Experimenting with Explorator: a Direct Manipulation Generic RDF Browser and Querying Tool. Visual Interfaces to the Social and the Semantic Web (VISSW 2009), Sanibel Island, Florida.
- [9] Haase, P., Mathäß, T., Schmidt, M., Eberhart, A., Walther, U. 2010. Semantic Technologies for Enterprise Cloud Management. The Semantic Web – ISWC 2010. pp. 98-113 Springer, Berlin, DE.
- [10] O. Corcho, A. López-Cima, A. Gómez-Pérez. 2006. The ODESeW 2.0 semantic web application framework. Proceedings of the 15th International Conference on World Wide Web, WWW '06, ACM Press, pp. 1049-1050.
- [11] M. Krötzsch, D. Vrandečić, M. Völkel: “Semantic MediaWiki”, in : Proceedings of the Int. Semantic Web Conference, ISWC'06. LNCS Vol. 4273, 2006, pp. 935-942.
- [12] Bizer, C., Cyganiak, R. 2006. D2R Server - Publishing Releational Databases on the Semantic Web. Poster at the 5th International Semantic Web Conference (Athens, USA, 2006).
- [13] Rodriguez, J.B. and Gómez-Pérez, A. 2006. Upgrading relational legacy data to the semantic web. Proceedings of the 15th international conference on World Wide Web (New York, NY, USA, 2006), 1069-1070.
- [14] L. Richardson, S. Ruby: “Restful Web Services”, O'Reilly, Cambridge, MA, 2007.
- [15] Richardson, L. and Ruby, S. 2007. Restful Web Services. O'Reilly.
- [16] E. Prud'hommeaux, A. Seaborne: “SPARQL Query Language for RDF”. Recommendation, World Wide Web Consortium (2008). <http://www.w3.org/TR/rdf-sparql-query>
- [17] D. Crane, E. Pascarello, D. James: “Ajax in Action”, Manning, Greenwich, CO, 2005.
- [18] R. García, J.M. Gimeno, F. Perdrix, R. Gil, M. Oliva, J.M. López, A. Pascual, M. Sendín: “Building a Usable and Accessible Semantic Web Interaction Platform”, World Wide Web, in press, 2010.
- [19] Sticler, P. 2005. CBD - Concise Bounded Description. World Wide Web Consortium.
- [20] Oren, E. et al. 2006. Extending Faceted Navigation for RDF Data. In International Semantic Web Conference (2006), 559–572.
- [21] D. Martin (Ed.). 2004. OWL-S: Semantic Markup for Web Services. W3C Member Submission. <http://www.w3.org/Submission/OWL-S>
- [22] Noy, N. et al. 2001. Creating Semantic Web contents with Protege-2000. Intelligent Systems, IEEE. 16, 2 (2001), 60-71.
- [23] Davies, S., Hatfield, J., Donaher, C., Zeitz, J. 2010. User Interface Design Considerations for Linked Data Authoring Environments. LDOW2010, April 27, 2010, Raleigh, USA.