UNIVERSIDAD DE CASTILLA-LA MANCHA

University of Castilla - La Mancha

Analyzing Linked Data tools for SHARK

Technical Report

Cristina Roda, Elena Navarro, Carlos E. Cuesta

September 2013

Architectural Knowledge (AK) has been an integral part of Software Architecture specification since its original inception, but it has been explicitly managed only recently. It can be described as a computational structure of design decisions and rationales; recent research emphasizes that availability must be complemented by an effective use of this information. Therefore, when systems are complex and large, an effective searching method becomes critical. We propose the use of Linked Data techniques to define and query AK, thus achieving a flexible storage and scalable search. Our approach suggests storing the network of decisions in RDF format to be retrieved efficiently by means of SPARQL queries. This format is, by definition, general and extensible. As a side effect, many different AK structures can be described this way, which then becomes a general format to describe AK. In that way, this work analyzes some significant features regarding to AK of several Linked Data tools in order to determine which ones are the best/worst for handling AK as Linked Data.

1 What is Open Data?

According to [1], *Open Data* can be defined as "data that can be freely used, reused and redistributed by anyone". So, this type of information is available to everyone in a computational format.

This idea of *openness* has some requirements in relation to data which are [1]:

1. *Availability and Access*, data must be available in a convenient form, preferably through the Internet as nowadays everybody have open and freely access to it everywhere.
2. *Reuse and Redistribution*, data must be provided allowing its easy reuse and redistribution.
3. *Universal Participation*, everyone must be able to use, reuse and redistribute this kind of data, without discrimination against any person or group of people.

We may find lots of types of Open Data, such as:

1. *Cultural Data*, information about cultural works, like titles or authors, normally stored by libraries, archives and museums.
2. *Science Data*, data produced by means of scientific research in any scientific field.
3. *Financial Data*, data from government or financial markets, like accounts, stocks, shares, etc.
4. *Statistic Data*, data produced by statistical offices.
5. *Geodata*, information used to make any type of maps.
6. *Weather Data*, data used to predict and understand the weather.
7. *Transport Data*, such as timetables, routes or on-time statistics.
8. *Environment Data*, data related to the natural environment (rivers, seas, etc.).

And *why should data be open?* These are the main reasons:

1. *Transparency*. In a well-functioning, democratic society, citizens need to know what their government is doing. To do that, they must be able freely to access government data and information and to share that information with other citizens. Transparency isn't just about access, it is also about sharing and reuse — often, to understand material it needs to be analysed and visualized and this requires that the material be open so that it can be freely used and reused.
2. *Releasing social and commercial value*. In a digital age, data is a key resource for social and commercial activities. Everything from finding your local post office to building a search engine requires access to data, much of which is created or held by government. By opening up data, government can help drive the creation of innovative business and services that deliver social and commercial value.
3. *Participation and engagement* – participatory governance or for business and organizations engaging with your users and audience. Much of the time citizens are only able to engage with their own governance sporadically — maybe just at an election every 4 or 5 years. By opening up data, citizens are enabled to be much more directly informed and involved in decision-making. This is more than transparency: it's about making a full "read/write" society, not just about knowing what is happening in the process of governance but being able to contribute to it.

2 What is Linked Data?

As its name indicates, *Linked Data* implies using the Web in order to create typed links between data from different sources, i.e. it refers to a set of best practices for publishing and connecting structured data on the Web [2]. All this information produces a global data space, commonly called the *Web of Data* or *Semantic Web*.

Until now, we have been used the term Web of Documents due to the fact that the information connected on the Web was made up of simply documents, but today this paradigm is changing. Lots types of data can be interconnected through the Web, not only documents, so the architecture of the Web of Data tends to be very similar to the WWW one.

In this sense, the *Semantic Web* isn't just about putting data on the web [3]. It is about making links, so that a person or machine can explore the Web of Data. With Linked Data, when you have some of it, you can find other, related, data.

Tim Berners-Lee outlined the Linked Data principles in [3]:

1. Use *URIs* as names for things.
2. Use *HTTP URIs* so that people can look up those names.
3. If a URI is looked up, provide useful information, using the standards *RDF* and *SPARQL*.
4. Include *links* to other URIs so that they can discover more things.

Namely, a *URI* identify real world objects, abstract concepts, web documents and digital content. This URI must be a *HTTP URI* which can identify the same as URIs and enable these URIs to be dereferenced (or look up) over the HTTP protocol into a description of the identified object or concept. This description should follow the standard *RDF* which is a single data model for publishing structured data on the Web, but not a data format, so RDF data can be serialized in different formats, like RDF/XML, RDFa, Turtle, etc. Finally, this information represented as RDF has to be connected using *RDF links* that are hyperlinks in the Linked Data context which connect disparate data into a single global data space [4].

In summary, like the Web of Hypertext, the Web of Data is constructed with documents on the web. However, unlike the Web of Hypertext, where links are relationships anchors in hypertext documents written in HTML, the Web of Data has links between arbitrary things described by RDF [3].

3 What is Linked Open Data?

At this point, we should know what is Open Data and Linked Data as two separate things so that now it is easier to define what is *Linked Open Data* (LOD). It is [3] Linked Data released under an open license, which does not impede its reuse for free.

In this sense, Tim Berners-Lee [3] proposes a star scheme in order to know how good and available is our Linked Open Data. Under this scheme, you get one star if the information has been made public at all, i.e. if it has an open licence. The more stars you get, as you make it progressively more powerful, the easier for people to use.

★ Available on the web (whatever format) but with an open licence, to be Open Data.

★★ Available as machine-readable structured data (e.g. excel instead of image scan of a table).

★★★ As (2) plus non-proprietary format (e.g. CSV instead of excel).

★★★★ All the above plus use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff.

★★★★★ All the above plus link your data to other people's data to provide context.

4 Linked Data Tools

This section is intended to show how to use some Linked Data tools in order to manage Architectural Knowledge (AK). More specifically, Fig 1 displays the AK decision network that we are going to exploit within the different Linked Data tools.

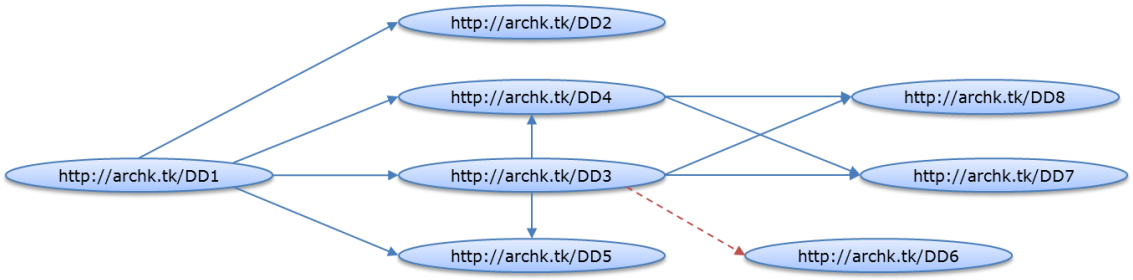


Fig 1 - AK decision network

Each *http://archk.tk/DDX* node represents a *Design Decision* (DD) in our network. All relationships between these nodes are *constrains* type, except for the connection between *http://archk.tk/DD3* and *http://archk.tk/DD6*, which is a relation of exclusion.

The following RDF/XML code lines represent our AK decision network, written in an *.rdf* file.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE rdf:RDF[
        <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
        <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
        <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
        <!ENTITY prop 'http://archk.tk/property#'>
        <!ENTITY kind 'http://archk.tk/dd/kinds#'>
        <!ENTITY conn 'http://archk.tk/dd/relation#'>
        <!ENTITY att 'http://archk.tk/dd/attributes#'>
        <!ENTITY rat 'http://archk.tk/dd/attributes/rationale#'>
]>
<rdf:RDF xml:base="http://archk.tk/DD3" xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:conn="http://archk.tk/dd/relation#" xmlns:prop="http://archk.tk/property#"
xmlns:kind="http://archk.tk/dd/kinds#" xmlns:att="http://archk.tk/dd/attributes#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rat="http://archk.tk/dd/attributes/rationale#">
        <rdf:Description rdf:about="http://archk.tk/DD3">
                <att:author>Rafael</att:author>
                <att:category>Structural</att:category>
                <att:decision>Common Business Logic is Packaged as a Layer</att:decision>
                <att:risk rdf:datatype="&xsd;integer">0</att:risk>
                <att:scope>Universal</att:scope>
                <att:state>Approved</att:state>
                <att:rationale> The business logic of the application must be
encapsulated in a set of components for both the mobile application and the web server;
the common set of components is defined as a separate layer.
                </att:rationale>
                <rat:Layer>Business</rat:Layer>
```

```
		<att:timeStamp rdf:datatype="&xsd;dateTime">2012-05-
01T13:55:33.280368+01:00</att:timeStamp>
		<kind:classification>Existence</kind:classification>
		<conn:constrains rdf:resource="http://archk.tk/DD4" />
		<conn:constrains rdf:resource="http://archk.tk/DD5" />
		<conn:constrains rdf:resource="http://archk.tk/DD7" />
		<conn:constrains rdf:resource="http://archk.tk/DD8" />
		<conn:excludes rdf:resource="http://archk.tk/DD6" />
		<prop:type>DesignDecision</prop:type>
	</rdf:Description>
	<rdf:Description rdf:about="http://archk.tk/DD1">
		<att:author>Cris</att:author>
		<att:category>Structural</att:category>
		<att:decision>Using a three-layered architecture</att:decision>
		<att:risk rdf:datatype="&xsd;integer">0</att:risk>
		<att:scope>Universal</att:scope>
		<att:state>Approved</att:state>
		<att:rationale>Rationale1</att:rationale>
		<rat:Layer>Business</rat:Layer>
		<att:timeStamp rdf:datatype="&xsd;dateTime">2012-05-
01T13:55:33.280368+01:00</att:timeStamp>
		<kind:classification>Existence</kind:classification>
		<conn:constrains rdf:resource="http://archk.tk/DD2" />
		<conn:constrains rdf:resource="http://archk.tk/DD3" />
		<conn:constrains rdf:resource="http://archk.tk/DD4" />
		<conn:constrains rdf:resource="http://archk.tk/DD5" />
		<prop:type>DesignDecision</prop:type>
	</rdf:Description>
	<rdf:Description rdf:about="http://archk.tk/DD2">
		<att:author>Cris</att:author>
		<att:category>Structural</att:category>
		<att:decision>Using a User-Interface layer</att:decision>
		<att:risk rdf:datatype="&xsd;integer">0</att:risk>
		<att:scope>Universal</att:scope>
		<att:state>Approved</att:state>
		<att:rationale>Rationale2</att:rationale>
		<rat:Layer>Business</rat:Layer>
		<att:timeStamp rdf:datatype="&xsd;dateTime">2012-05-
01T13:55:33.280368+01:00</att:timeStamp>
		<kind:classification>Existence</kind:classification>
		<prop:type>DesignDecision</prop:type>
	</rdf:Description>
	<rdf:Description rdf:about="http://archk.tk/DD4">
		<att:author>Cris</att:author>
		<att:category>Structural</att:category>
		<att:decision>Defining classes of the mobile application for managing the
on-site ticketing process</att:decision>
		<att:risk rdf:datatype="&xsd;integer">0</att:risk>
		<att:scope>Universal</att:scope>
		<att:state>Approved</att:state>
		<att:rationale>Rationale4</att:rationale>
		<rat:Layer>Business</rat:Layer>
		<att:timeStamp rdf:datatype="&xsd;dateTime">2012-05-
01T13:55:33.280368+01:00</att:timeStamp>
		<kind:classification>Existence</kind:classification>
		<conn:constrains rdf:resource="http://archk.tk/DD7" />
		<conn:constrains rdf:resource="http://archk.tk/DD8" />
		<prop:type>DesignDecision</prop:type>
	</rdf:Description>
	<rdf:Description rdf:about="http://archk.tk/DD5">
		<att:author>Cris</att:author>
		<att:category>Structural</att:category>
		<att:decision>Defining classes for managing the main services offered by
the web application</att:decision>
		<att:risk rdf:datatype="&xsd;integer">0</att:risk>
		<att:scope>Universal</att:scope>
		<att:state>Approved</att:state>
```

```
                <att:rationale>Rationale5</att:rationale>
                <rat:Layer>Business</rat:Layer>
                <att:timeStamp rdf:datatype="&xsd;dateTime">2012-05-
01T13:55:33.280368+01:00</att:timeStamp>
                <kind:classification>Existence</kind:classification>
                <prop:type>DesignDecision</prop:type>
        </rdf:Description>
        <rdf:Description rdf:about="http://archk.tk/DD6">
                <att:author>Cris</att:author>
                <att:category>Structural</att:category>
                <att:decision>Using a Data Management layer</att:decision>
                <att:risk rdf:datatype="&xsd;integer">0</att:risk>
                <att:scope>Universal</att:scope>
                <att:state>Approved</att:state>
                <att:rationale>Rationale6</att:rationale>
                <rat:Layer>Business</rat:Layer>
                <att:timeStamp rdf:datatype="&xsd;dateTime">2012-05-
01T13:55:33.280368+01:00</att:timeStamp>
                <kind:classification>Existence</kind:classification>
                <prop:type>DesignDecision</prop:type>
        </rdf:Description>
        <rdf:Description rdf:about="http://archk.tk/DD7">
                <att:author>Cris</att:author>
                <att:category>Structural</att:category>
                <att:decision>Connecting the mobile application and the web
application</att:decision>
                <att:risk rdf:datatype="&xsd;integer">0</att:risk>
                <att:scope>Universal</att:scope>
                <att:state>Approved</att:state>
                <att:rationale>Rationale7</att:rationale>
                <rat:Layer>Business</rat:Layer>
                <att:timeStamp rdf:datatype="&xsd;dateTime">2012-05-
01T13:55:33.280368+01:00</att:timeStamp>
                <kind:classification>Existence</kind:classification>
                <prop:type>DesignDecision</prop:type>
        </rdf:Description>
        <rdf:Description rdf:about="http://archk.tk/DD8">
                <att:author>Cris</att:author>
                <att:category>Structural</att:category>
                <att:decision>Packaging separately the API that connects the Business
Logic to the Data Layer</att:decision>
                <att:risk rdf:datatype="&xsd;integer">0</att:risk>
                <att:scope>Universal</att:scope>
                <att:state>Approved</att:state>
                <att:rationale>Rationale8</att:rationale>
                <rat:Layer>Business</rat:Layer>
                <att:timeStamp rdf:datatype="&xsd;dateTime">2012-05-
01T13:55:33.280368+01:00</att:timeStamp>
                <kind:classification>Existence</kind:classification>
                <prop:type>DesignDecision</prop:type>
        </rdf:Description>
</rdf:RDF>
```

The Linked Data tools that we are going to talk about are *Virtuoso* (section 4.1), *Linked Media Framework* (section 4.2), *Apache Jena and Fuseki* (section 4.3), *TopBraid Suite* (section 4.4), *Sesame* (section 4.5), *Mulgara* (section 4.6), *RedStore* (section 4.7), *Callimachus* (section 4.8). In section 4.9, we are going to present other Linked Data tools that are not going to take part of the feature analysis in section 5, but we consider that they have to be mentioned.

4.1 Virtuoso

Virtuoso is a single data server, developed by *OpenLink Software*, which offers functionality from between traditional Relational data management to Linked Data server. In

particular, this hybrid product allows you to deal with the following areas [5], of which we are particularly interested in the italics marked ones:

- Relational data management
- *RDF data management*
- XML data management
- Free text content management and full text indexing
- Document Web server
- *Linked Data server*
- Web application server
- Web services deployment

In order to exploit this server with Linked Data, we may use *OpenLink Data Spaces* (ODS) [6] that is an expanding line of Virtuoso-based applications for establishing and managing data on the web and the emerging Semantic web. This application suite includes a web based file sharing platform, called *ODS Briefcase* [7], which allows users to control file access rights, search based on content and metadata. In addition, all resources are exposed as RDF data sets so file server functionality can be exploited by means of SPARQL query language for Semantic Web.

Some key features of *ODS Briefcase* are [6]:

- Automatic Metadata Management and Extraction (automatic extraction of file metadata from many file types)
- Powerful Full-Text Search (by metadata, path, filename, content words, mime-type, etc.)
- Flexible Data Access (via SPARQL, among others)
- Open Data Access (easy and transparent integration within any environment)
- Security
- Unified Central storage and access point
- Resource tagging (for categorize the content by user-defined tags)
- Shared Folders View (shows all resources that can be accessed by the user)
- Version Control

Especially for managing Linked Data, *ODS Briefcase* offers several characteristics as upload RDF files, validating their format previously according to a particular syntax, like XML; edit these files, consume uploaded data to the server by means of SPARQL [8], showing query results in various formats (HTML, XML, JSON, Javascript, NTriples, RDF/XML or spreadsheet).

In Fig 2, we may see the *OpenLink Briefcase* interface and the *textual_example_LD.rdf* file stored in the *AK_data* folder.

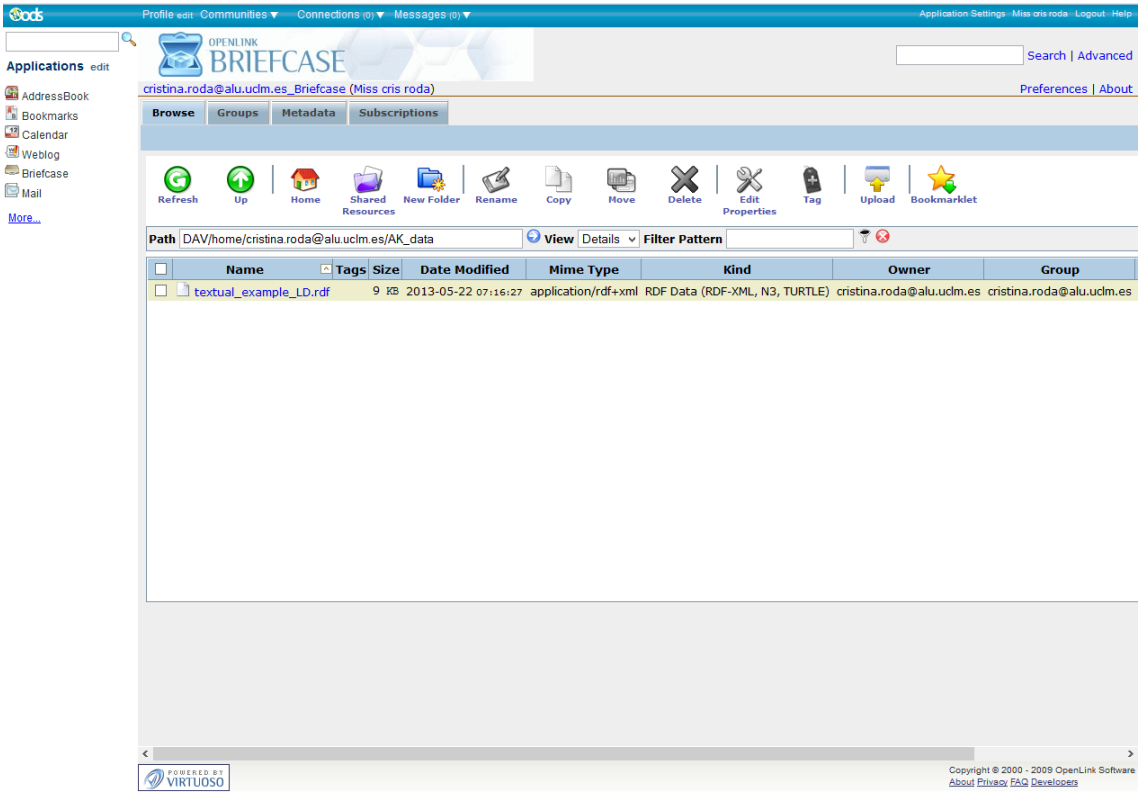


Fig 2 – *OpenLink Briefcase*

In Fig 3 and Fig 4**¡Error! No se encuentra el origen de la referencia.**, we can see the *OpenLink Virtuoso SPARQL Query* interface and the query results in HTML format, respectively.

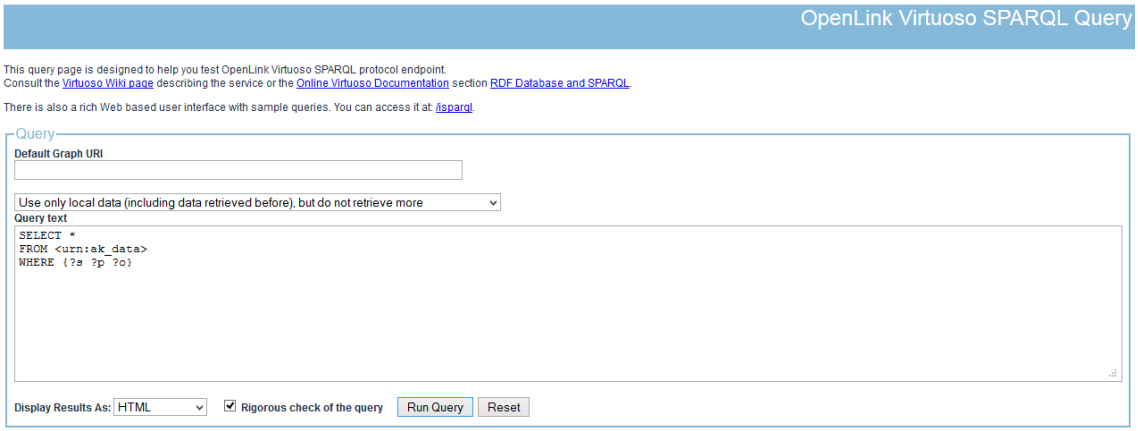


Fig 3 – *OpenLink Virtuoso SPARQL Query*

s	p	o
http://archk.tk/DD4	http://archk.tk /dd/attributes#timeStamp	2012-05-01T13:55:33.280368+01:00
http://archk.tk/DD4	http://archk.tk /dd/kinds#classification	Existence
http://archk.tk/DD4	http://archk.tk /dd/relation#constrains	http://archk.tk/DD7
http://archk.tk/DD4	http://archk.tk /dd/relation#constrains	http://archk.tk/DD8
http://archk.tk/DD4	http://archk.tk/property#type	DesignDecision
http://archk.tk/DD4	http://archk.tk /dd/attributes#author	Cris
http://archk.tk/DD4	http://archk.tk /dd/attributes#category	Structural
http://archk.tk/DD4	http://archk.tk /dd/attributes#decision	Defining classes of the mobile application for managing the on-site ticketing process
http://archk.tk/DD4	http://archk.tk/dd/attributes#risk	0
http://archk.tk/DD4	http://archk.tk /dd/attributes#scope	Universal
http://archk.tk/DD4	http://archk.tk/dd/attributes#state	Approved
http://archk.tk/DD4	http://archk.tk /dd/attributes#rationale	Rationale4
http://archk.tk/DD5	http://archk.tk /dd/attributes#timeStamp	2012-05-01T13:55:33.280368+01:00
http://archk.tk/DD5	http://archk.tk /dd/kinds#classification	Existence
http://archk.tk/DD5	http://archk.tk/property#type	DesignDecision
http://archk.tk/DD5	http://archk.tk /dd/attributes#author	Cris
http://archk.tk/DD5	http://archk.tk /dd/attributes#category	Structural
http://archk.tk/DD5	http://archk.tk /dd/attributes#decision	Defining classes for managing the main services offered by the web application
http://archk.tk/DD5	http://archk.tk/dd/attributes#risk	0
http://archk.tk/DD5	http://archk.tk /dd/attributes#scope	Universal
http://archk.tk/DD5	http://archk.tk/dd/attributes#state	Approved

Fig 4 – SPARQL query results in HTML format

In addition to this, we can create SPARQL queries based on a graph representation and run them using *OpenLink iSPARQL* [9]. Fig 5 shows the *OpenLink iSPARQL* interface. Notice that the data source (URL) has to be given, in our case *urn:ak_data*.

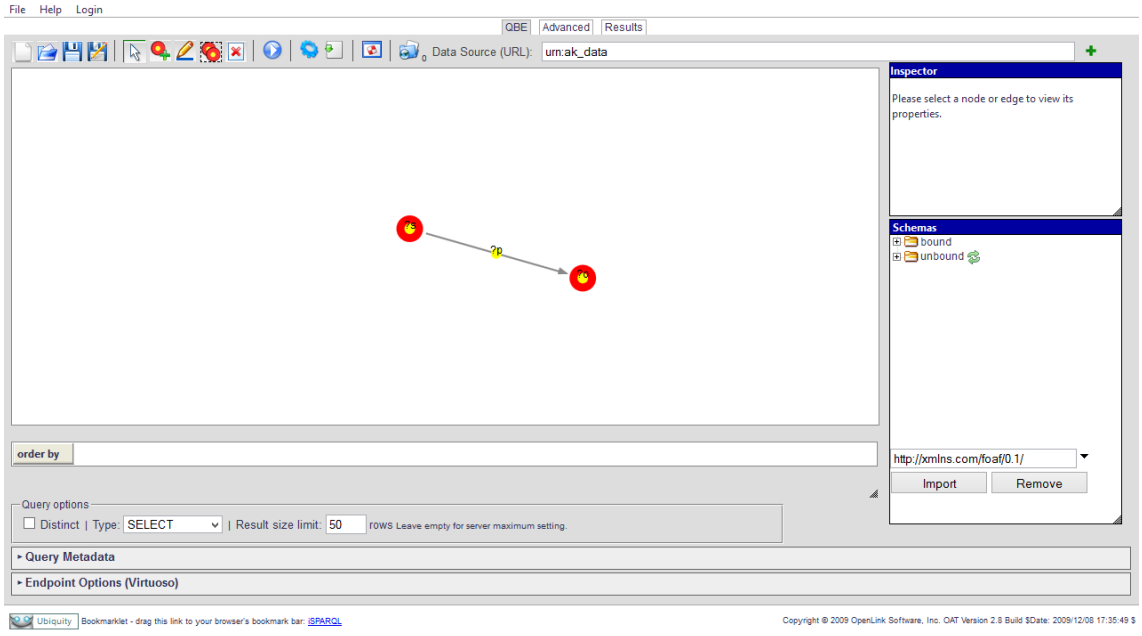


Fig 5 – *OpenLink iSPARQL*

4.1.1 Architectural Knowledge as Linked Data: step by step with Virtuoso

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using Virtuoso server and *ODS Briefcase*.

1. First of all, we have to download the open version of Virtuoso Universal Server and install it in our machine (http://virtuoso.openlinksw.com/download/). At the time of writing, the latest release is 7.0, and it is available for Windows, Linux and Mac OS. We have to be registered to make the download, but registration is totally free.

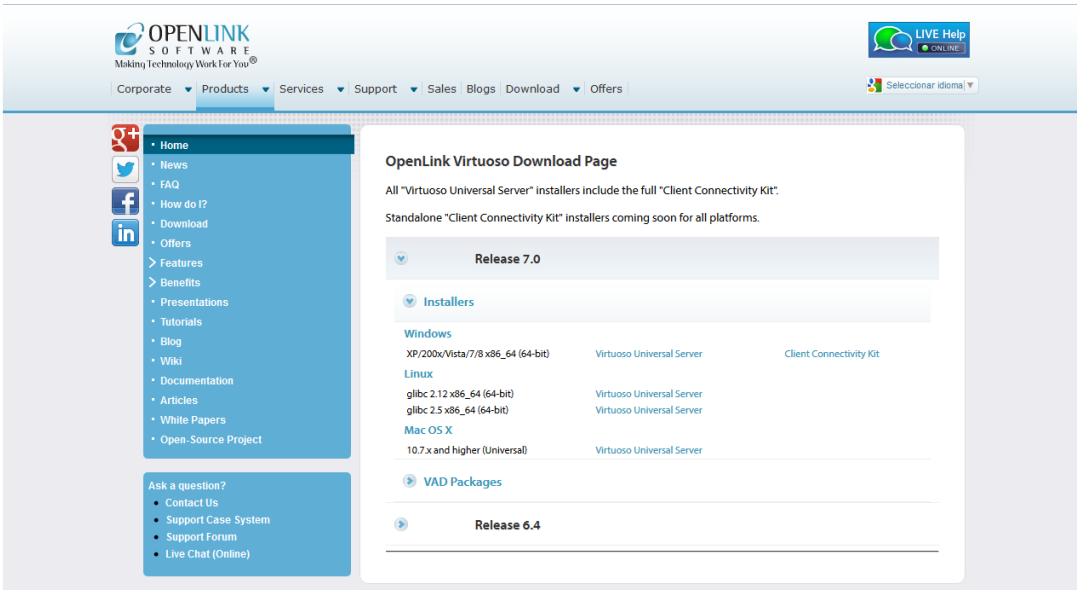


Fig 6 – OpenLink Virtuoso Download Page

2. With our credentials, now we can go to http://my.openlinksw.com/ods/ and sign in. Then, we are allowed to use *ODS Briefcase* by clicking the *Briefcase* link from the left vertical navigator bar.

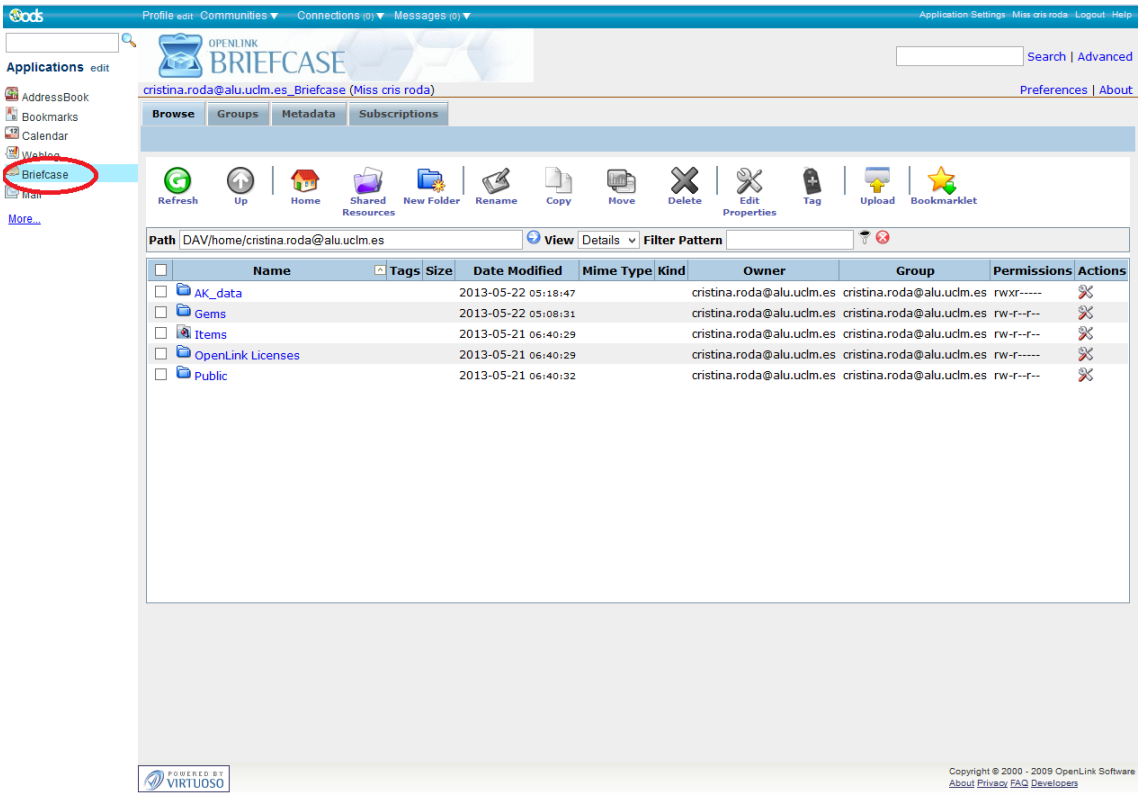


Fig 7 - *OpenLink Briefcase* overall interface

3. We can create a new folder to upload our RDF file into it. Click in “New Folder” and fill in the required information, selecting the type “RDF Upload Folder”, as you see in Fig 8. Go to tab “RDF Upload”, and specify the “Graph name”, for example, *urn:ak_data*, as you see in Fig 9. Finally, click the “Create” button.

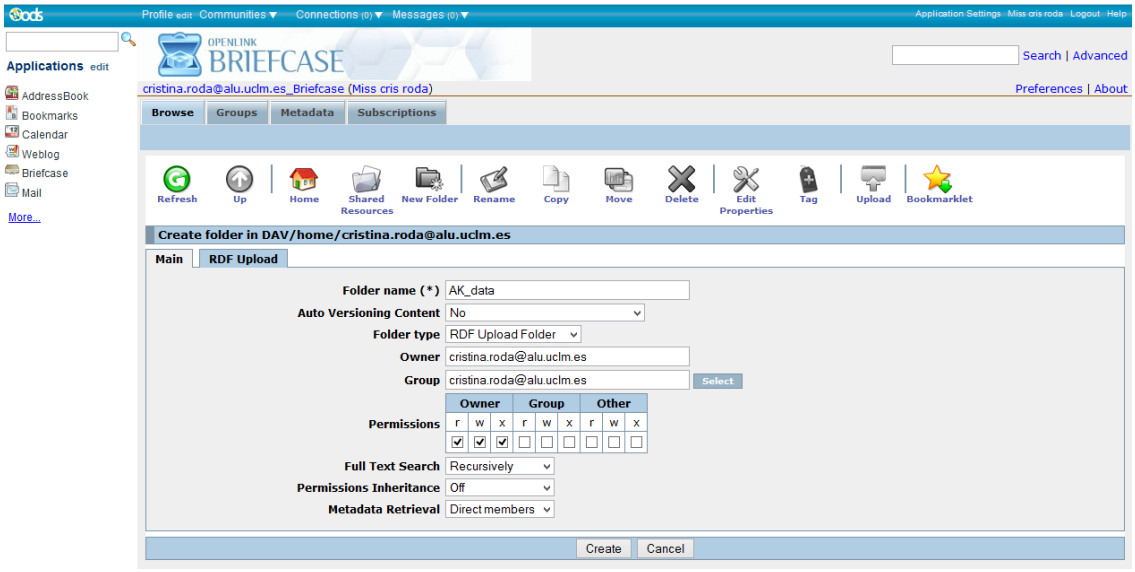


Fig 8 – Create a new folder with *ODS Briefcase* (1)

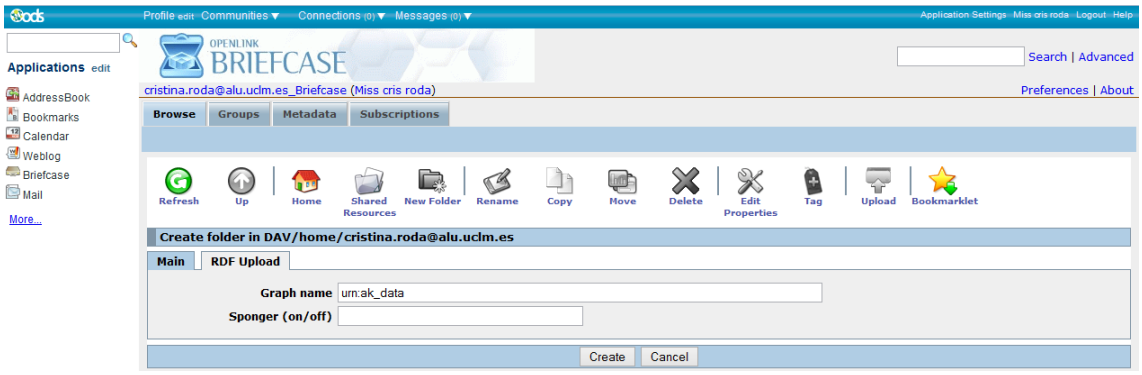


Fig 9 – Create a new folder with *ODS Briefcase* (2)

4. Once the new folder has been created, you can access it by clicking its name link. Then you may upload a new file to it. Click in the “Upload” option and specify the required information, selecting the file source from your computer, as you see in Fig 10. Finally, click the “Upload” button and a new RDF file should be created with the specified metadata.

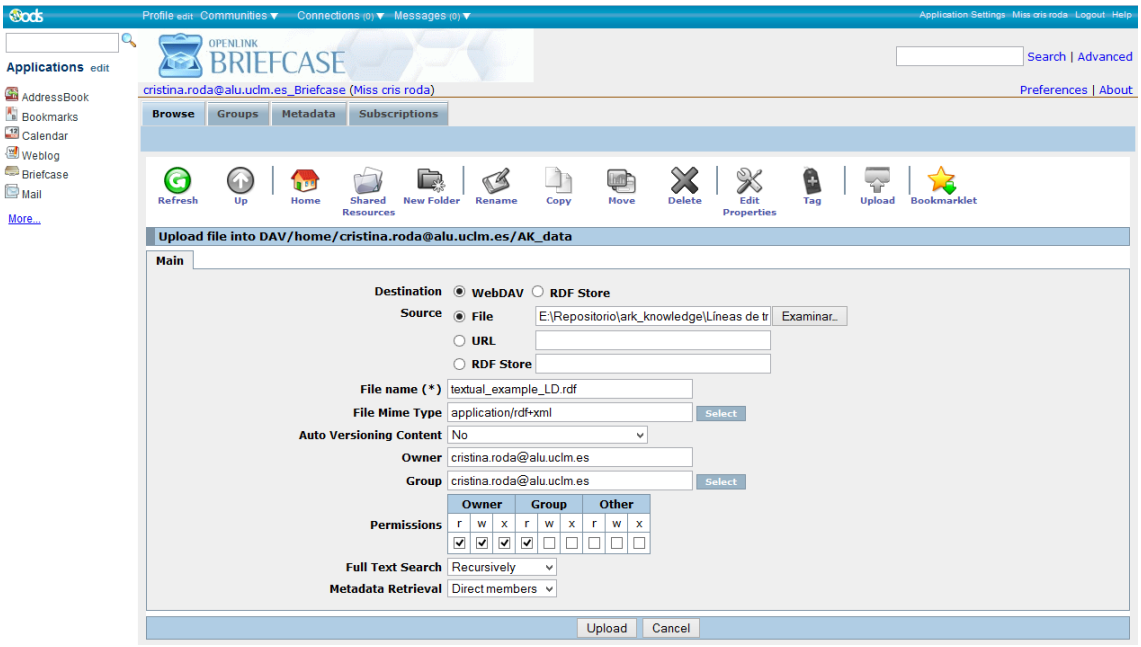


Fig 10 – Upload an RDF file with *ODS Briefcase*

5. We can check the inserted triples, going to *OpenLink Virtuoso SPARQL Query* endpoint: http://my.openlinksw.com/sparql/. There, we enter a query in the "Query text" area and press the "Run Query" button to execute it, as you see in Fig 11. The results of this query are the same as in Fig 4.

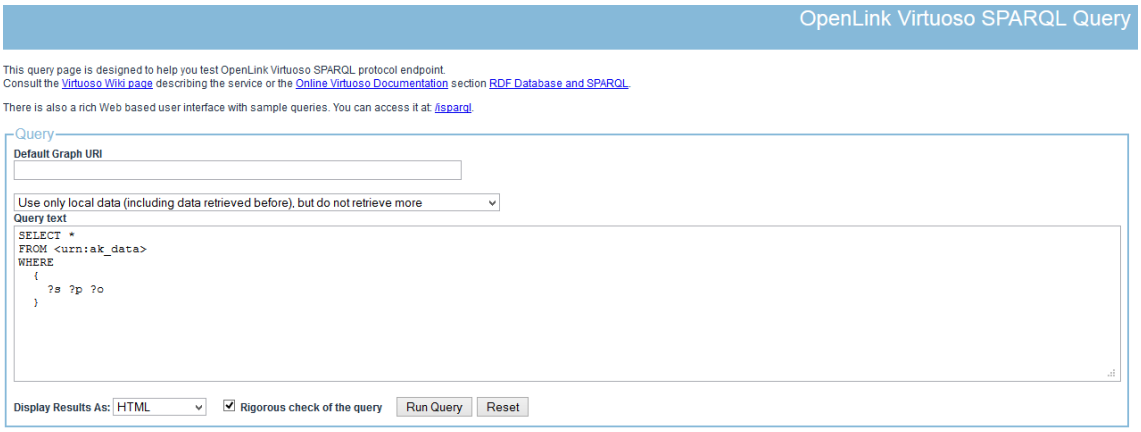


Fig 11 – A simple query using *OpenLink Virtuoso SPARQL Query*

For more information about these steps, you can visit this link.

4.2 Linked Media Framework (LMF)

Linked Media Framework [10] is an easy-to-setup server application that packages Semantic Web technologies to offer advanced services. This framework consists of two main elements: LMF Core and LMF Modules.

The LMF Core component is a Linked Data server that allows exposing data following the Linked Data Principles. In addition to the Linked Data Server, the LMF Core also offers a highly configurable Semantic Search service and a SPARQL endpoint.

Otherwise, we can enhance these LMF Modules from Linked Media Framework [10]:

- *LMF Semantic Search* that offers a highly configurable Semantic Search service.
- *LMF Linked Data Cache* that implements a cache to the Linked Data Cloud that is transparently used when querying the content of the LMF. In case a local resource links to a remote resource in the Linked Data Cloud and this relationship is queried, the remote resource will be retrieved in background and cached locally.

In addition to this, LMF implements its own authentication and authorization mechanism. In this manner, there are three pre-defined profiles: *simple* (allows read access from everywhere and write access only from localhost or other local interfaces), *standard* (allows read access from everywhere and write access only for authenticated users with manager role), and *restricted* (allows access only for authenticated users).

By default, LMF will use the *simple* profile, allowing only access from localhost. If we want to change this profile, we have to go to Security-profiles and then press *Save*.

We know that search over resources will always be domain-specific and will need to take into account the schema of the data. Therefore, the semantic search component of LMF provides only very simple and straightforward search functionality by default for three typical generic cases of metadata: *RDF* (rdfs:label, rdfs:comment), *DC* (dc:title, dc:description) and *SKOS* (skos:prefLabel, skos:altLabel, skos:definition).

However, in order to adapt the search component to our specific domain, LMF Admin Interface offers the possibility to define a *RDF Path Program*, i.e. a set of rules that map index fields to RDF properties or paths of RDF properties. For example, the following program (rdf) defines four fields (title, summary, tag, type):

```
@prefix hg : <http://www.holygoat.co.uk/owl/redwood/0.1/tags/> ;
title   = rdfs:label :: xsd:string ;
summary = rdfs:comment :: lmf:text ;
tag     = hg:taggedWithTag / hg:name :: xsd:string ;
type    = rdf:type :: xsd:anyURI ;
```

In the most simple case (e.g. title), the rule maps an index field to exactly one RDF property. In more complex cases, the rule allows to follow a path of RDF properties; e.g. in the "tag" field, the rule would start at the current resource and follow the *hg:taggedWithTag* property, and from there it will follow the *hg:name* property and store it in the index. (Learn more about *RDF Path Program* in http://code.google.com/p/lmf/wiki/ModuleSemanticSearch and http://code.google.com/p/ldpath/).

4.2.1 Architectural Knowledge as Linked Data: step by step with LMF

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *Linked Media Framework*.

1. First of all, we have to download LMF in http://code.google.com/p/lmf/downloads/list. At the time of writing, the latest standalone release is 2.3.1. It is recommended to download a standalone JAR file in order to install all LMF components and Apache Tomcat automatically.

2. Install the JAR file double-clicking on it, with default options. Notice that the installation path on windows systems may not contain whitespaces, which is a serious bug of Tomcat application server.
3. Once we have installed LMF, we have to start the LMF server. Go to Salzburg NewMediaLab folder, then Linked Media Framework folder and click in "Start Linked Media Framework" icon.

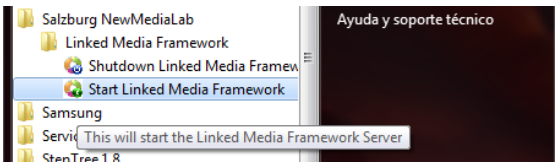


Fig 12 – Starting the LMF server

4. Now, we can access the administration interface by pointing our browser to http://*{your_host_name}*:8080/LMF and going to Administration-Linked Media Framework.

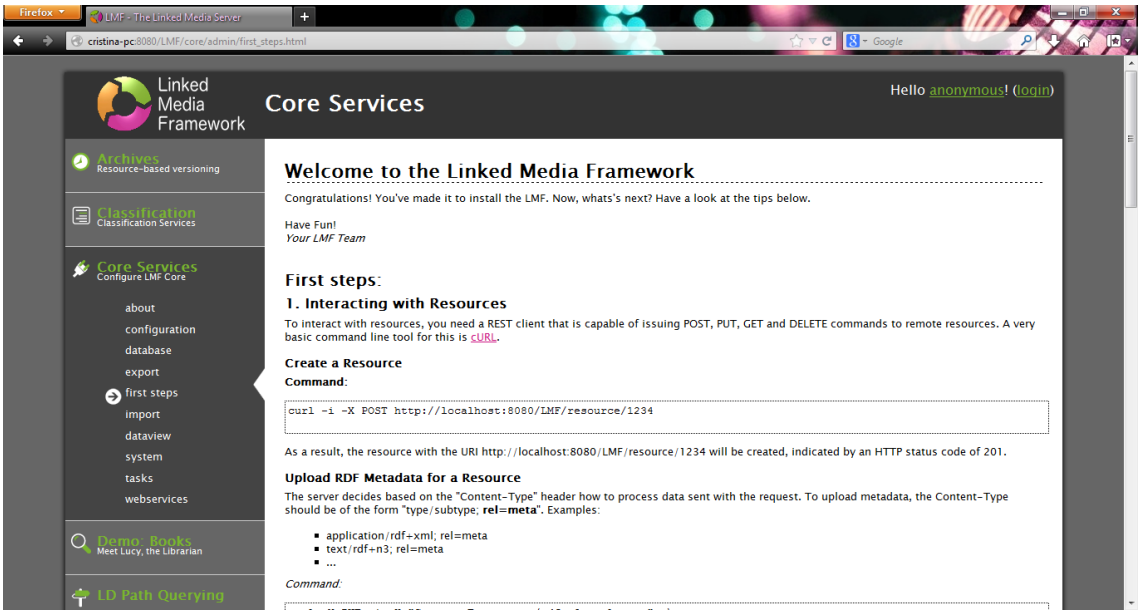


Fig 13 – The administration interface of LMF

5. Then, go to Core Services-import in order to upload an RDF file into LMF server. First, select the input source-type as *File*, choose the RDF file and click the *Import!* Button. A pop-up window notifies that the import was made successfully.

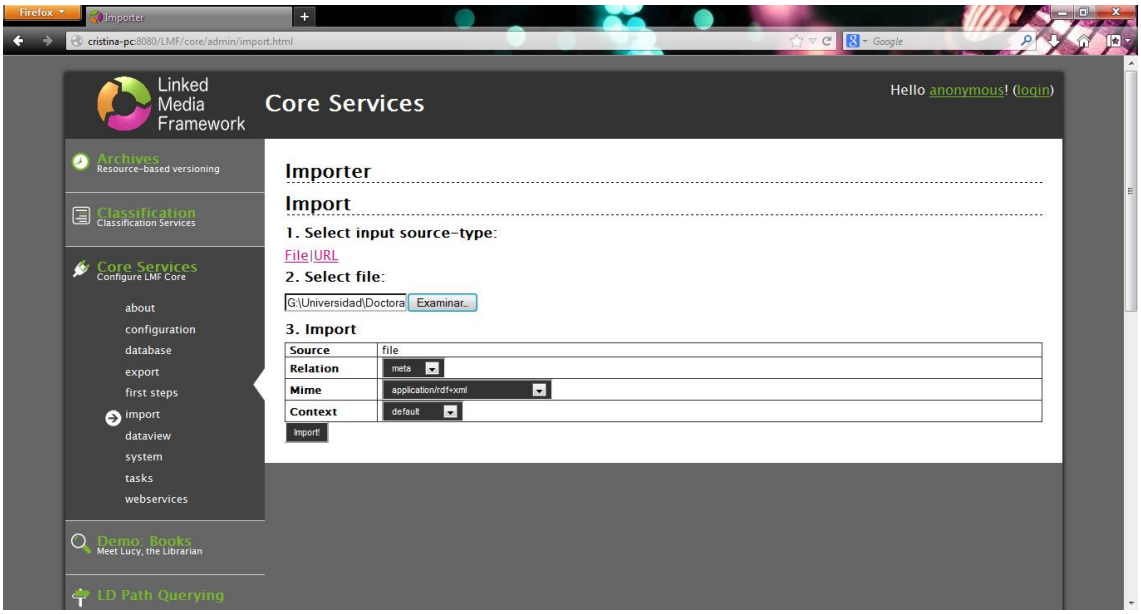


Fig 14 – Adding an RDF file into LMF

Then, we can go to Core Services-dataview and see the RDF data already imported to LMF.

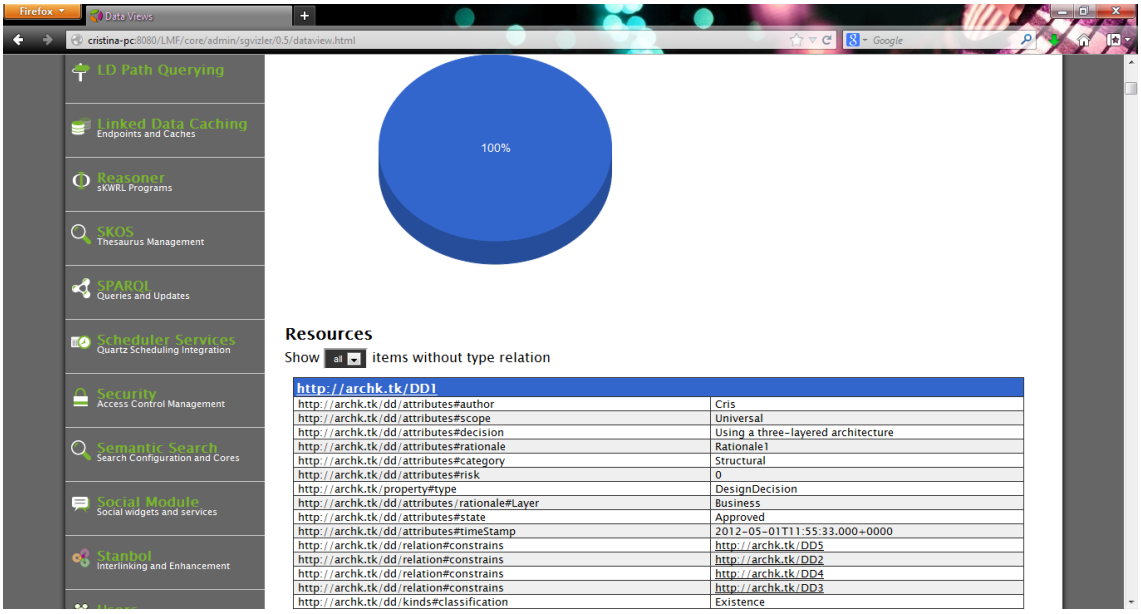


Fig 15 – Data view with LMF

By clicking each resource, for example here *http://archtk.tk/DD1*, we can see its local description, as you see in Fig 16.

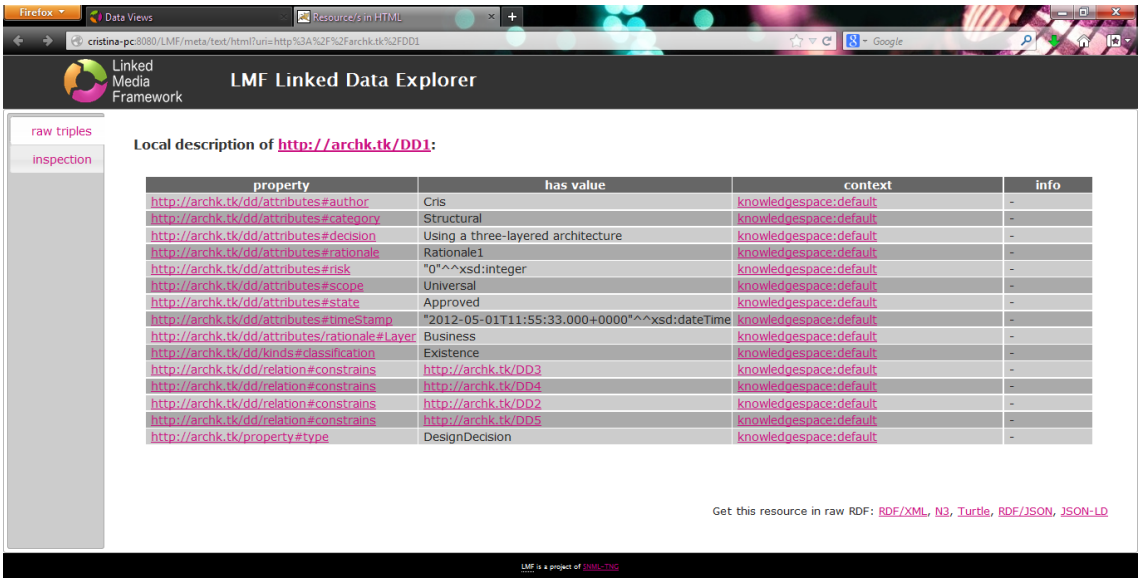


Fig 16 – LMF Linked Data Explorer

6. Finally, we can execute SPARQL queries in order to retrieve some RDF data from LMF server. Notice that LMF offers three different manners to make these queries.

   We can use *Flint SPARQL Editor*, going to SPARQL-flint sparql editor.

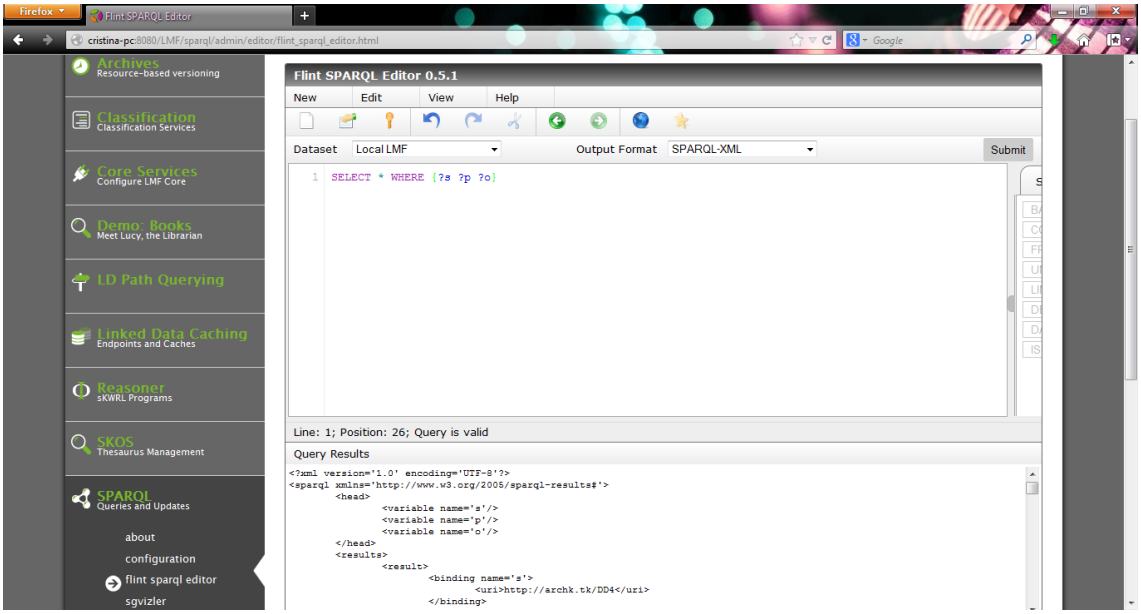


Fig 17 – *Flint SPARQL Editor* within LMF

Or we can use a Javascript SPARQL result set visualizer called *Sgvizler* that allows us to enter custom SPARQL queries and visualize their results in different kinds of charts. Go to SPARQL- sgvizler to use it. In Fig 18, we can see the output of the query at the bottom of the screen in a table format.

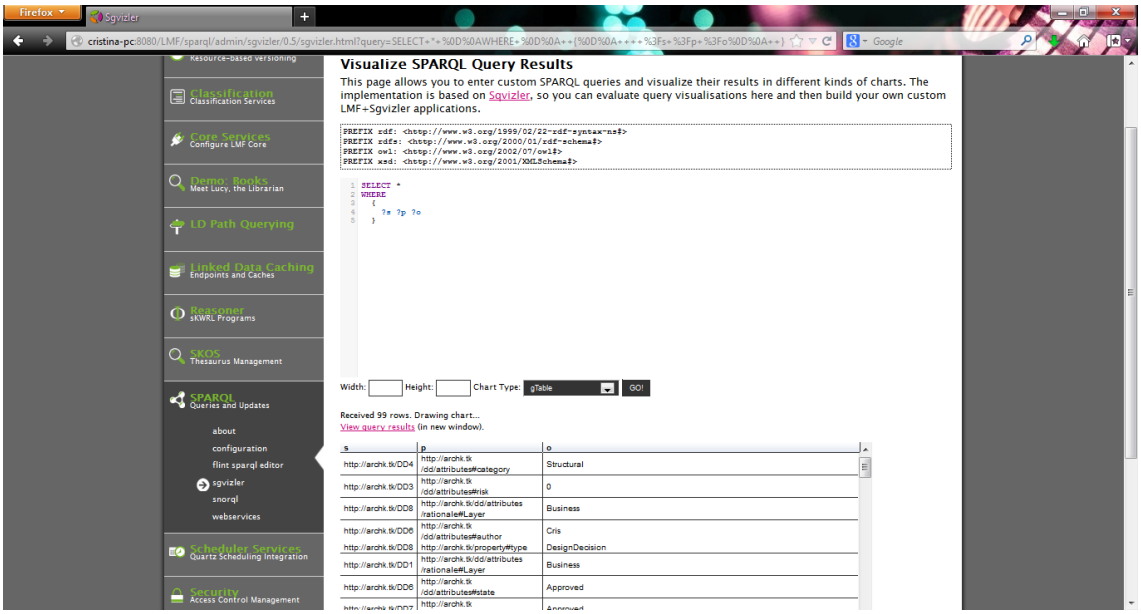


Fig 18 – *Sgvizler* SPARQL visualizer within LMF (1)

Notice that Sgvizler will show our data properly or report an error, depending on compatibility between our data and the chart type selected. Specifically, our Linked Data can only be represented with gOrgChart, gTable (Fig 18), sDefList, sList and sText chart types. In Fig 19, we can see an example of an error when we pretend to visualize our results as a bar chart (gBarChart type): "Data column(s) for axis #0 cannot be of type string".

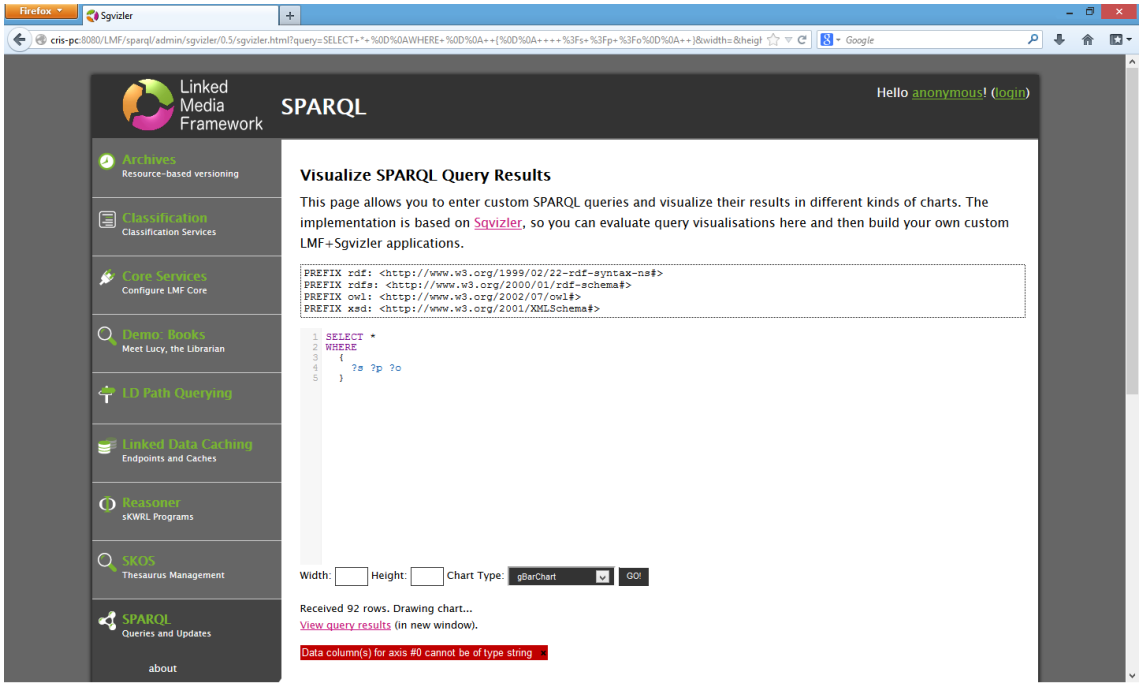


Fig 19 - *Sgvizler* SPARQL visualizer within LMF (2)

Finally, we can use *Snorql*, which is the SPARQL explorer for DBpedia[1], a project aiming to extract structured content from the information created as part of the Wikipedia project. Go to SPARQL-snorql to use Snorql.

[1] DBpedia official website: http://dbpedia.org/About

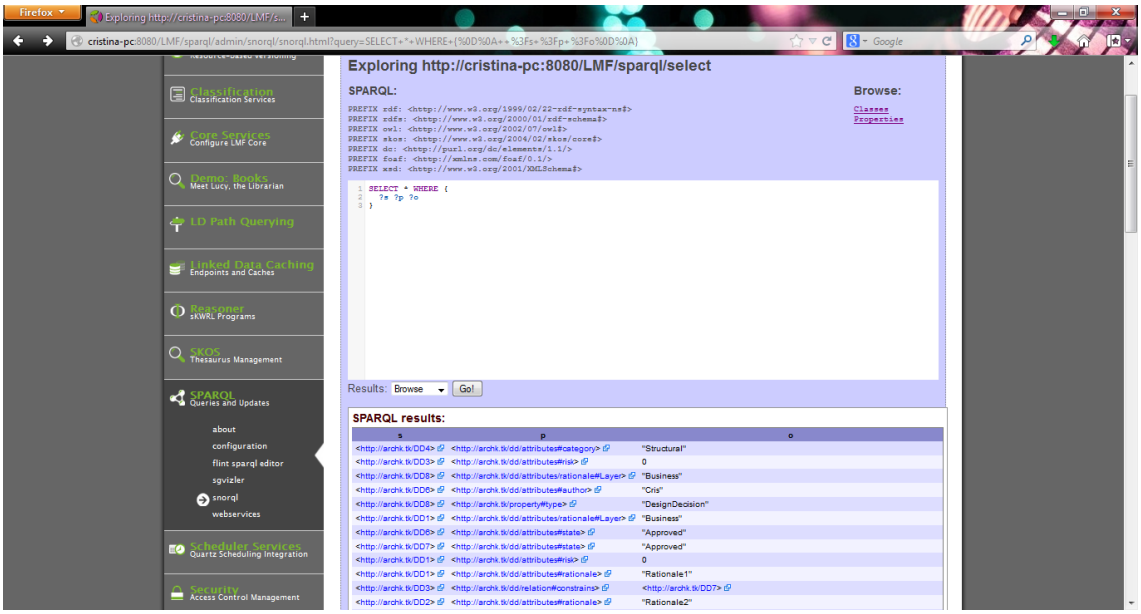


Fig 20 – *Snorql* within LMF

4.3 Apache Jena and Fuseki

In general, *Apache Jena* [11] is a Java framework for building Semantic Web applications. In particular, *Jena* is a Java API for Semantic Web applications which can be used to create and manipulate RDF graphs. Therefore, Jena provides a collection of tools and Java libraries that allow you to develop semantic web and linked-data applications, tools and servers. This Jena framework includes the following features:

- An API for reading, processing and writing RDF data in XML, N-triples and Turtle formats
- An ontology API for handling OWL and RDFS ontologies
- A rule-based inference engine for reasoning with RDF and OWL data sources
- Stores to allow large numbers of RDF triples to be efficiently stored on disk
- A query engine compliant with the latest SPARQL specification
- Servers to allow RDF data to be published to other applications using a variety of protocols, including SPARQL

Thus, in order to provide support for RDF manipulation, Jena has object classes to represent properties, graphs (models), resources and literals. In addition, it provides constant classes for well-known schemas, such as RDF, RDFS, RDFa, Dublin Core or OWL, and also has some methods for reading and writing RDF as XML. Apart from that, Jena allows you to control the namespaces used on output with its prefix mappings; retrieve from a model the resource object, given its URI; access the properties of a resource, using some specific methods; query a model; or manipulate models as a whole by means of three operations: union, intersection and difference. So the primary use of Jena is to help you write Java code that handles RDF and OWL documents or descriptions.

On the other hand, *Fuseki* is a SPARQL server that offers the following services:

- SPARQL Query, SPARQL Update and file upload to a selected dataset
- Validators for SPARQL query and update and for non-RDF/XML formats

Notice that query outputs can be in several formats, such as JSON, XML, Text, CSV or TSV.

4.3.1 Architectural Knowledge as Linked Data: step by step with Apache Jena

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *Apache Jena*.

The aim of this tutorial is to provide a guide to prepare the Eclipse Platform for using Jena, not to explain in detail how to use their objects and classes themselves. To do this, visit http://jena.apache.org/tutorials/rdf_api.html.

1. First of all, we have to download and install Eclipse. Go to http://www.eclipse.org/downloads/ and download the latest version of the *Eclipse Classic* Platform. At the time of writing, the latest release is 4.2.2. Then, install it on your computer.
2. Open the Eclipse platform and create a new Java project. First, go to File-New-Project.

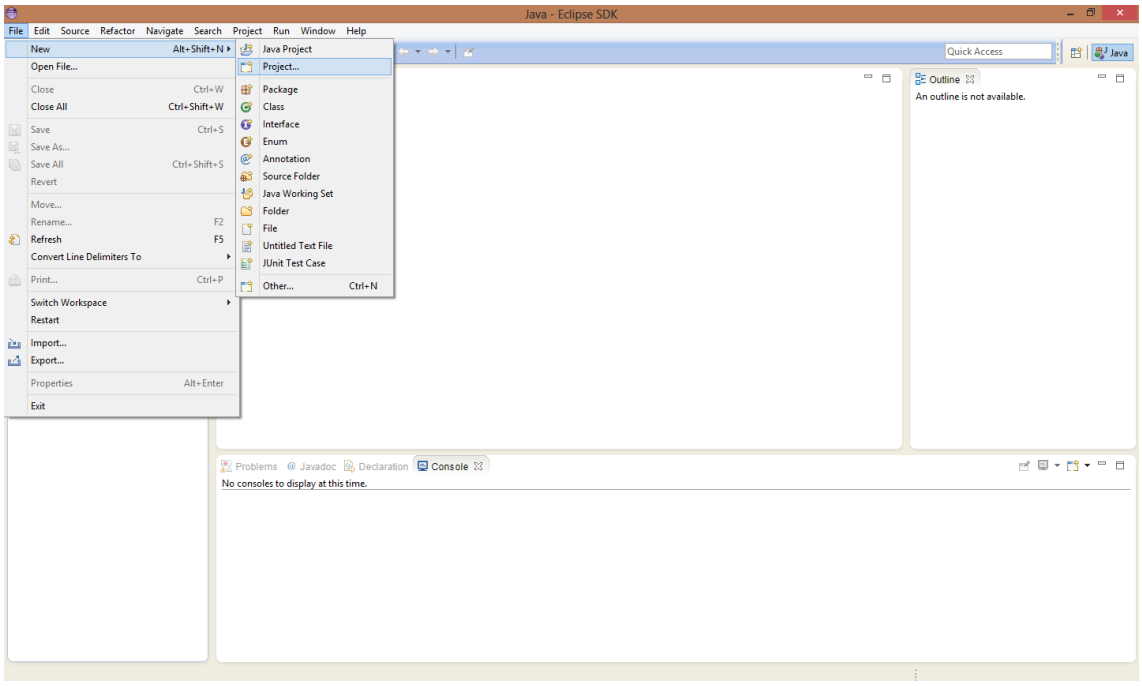


Fig 21 – Creating a new Java project with *Eclipse* (1)

Then select “Java Project” and click *Next*.

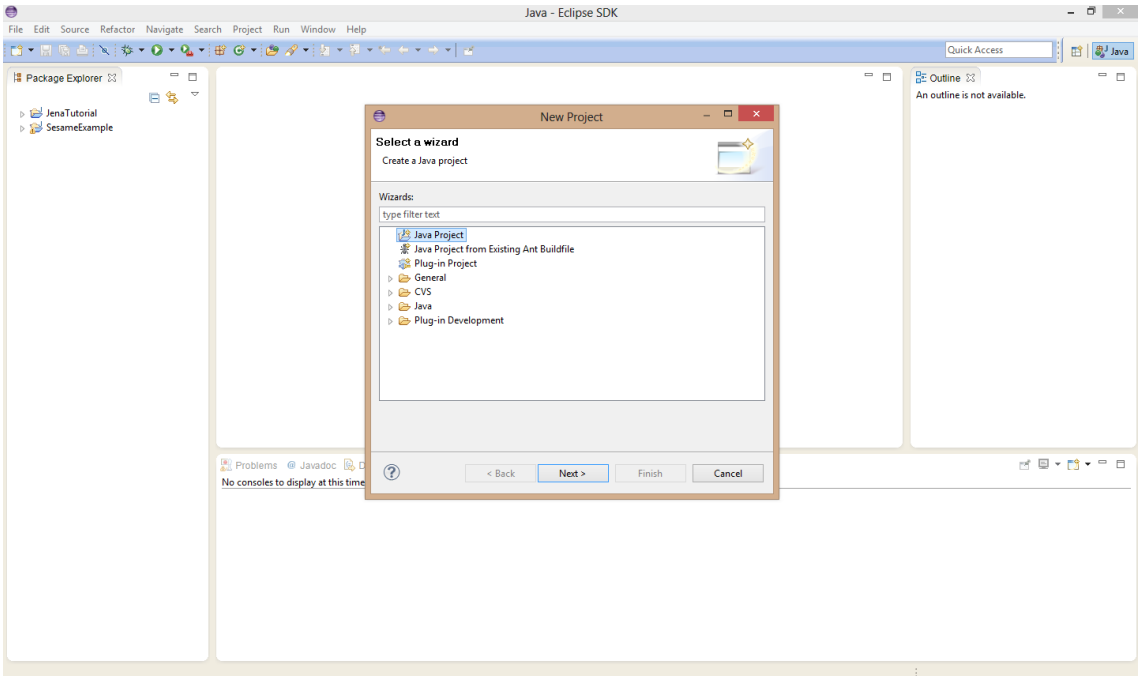


Fig 22 - Creating a new Java project with *Eclipse* (2)

Give a name to the project, leaving the default options and click *Finish*.

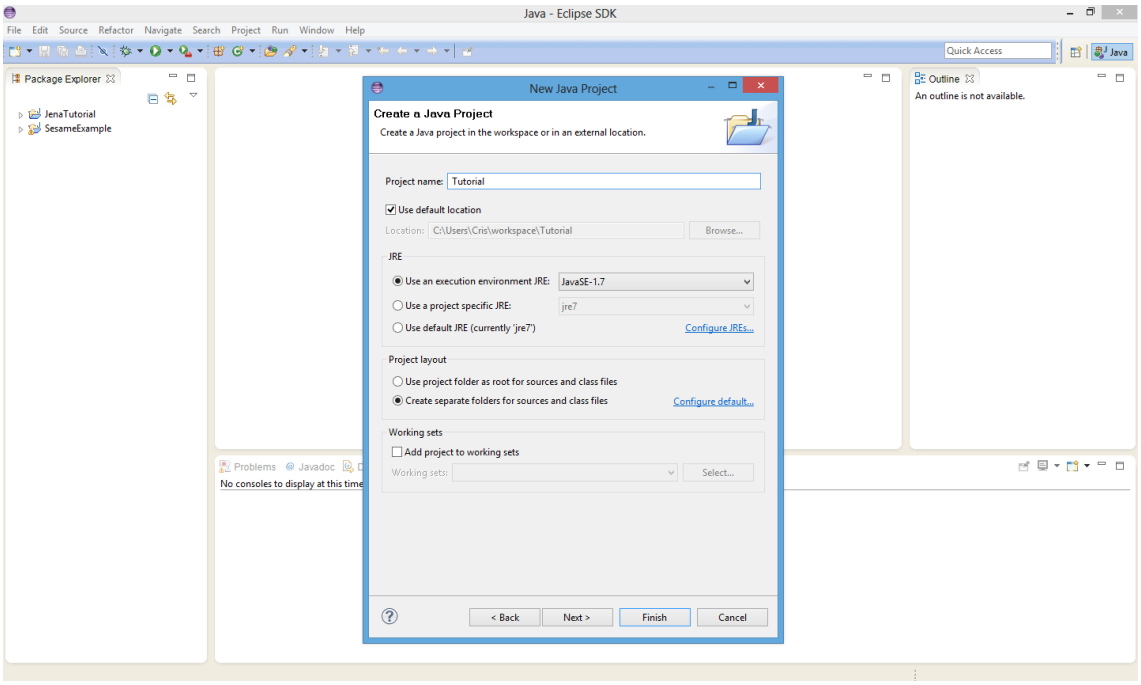


Fig 23 – Creating a new Java project with *Eclipse* (3)

3. Now, we have to download the Jena libraries in order to use them within Eclipse. Go to http://sourceforge.net/projects/jena/files/Jena/Jena-2.6.4/ to download them and unzip the file downloaded to be ready to use in Eclipse.
4. Then, we can define a *user library* in Eclipse, i.e. a declaration of a library that we can reference from any project, which contains the Jena libraries. Go to Window-Preferences.

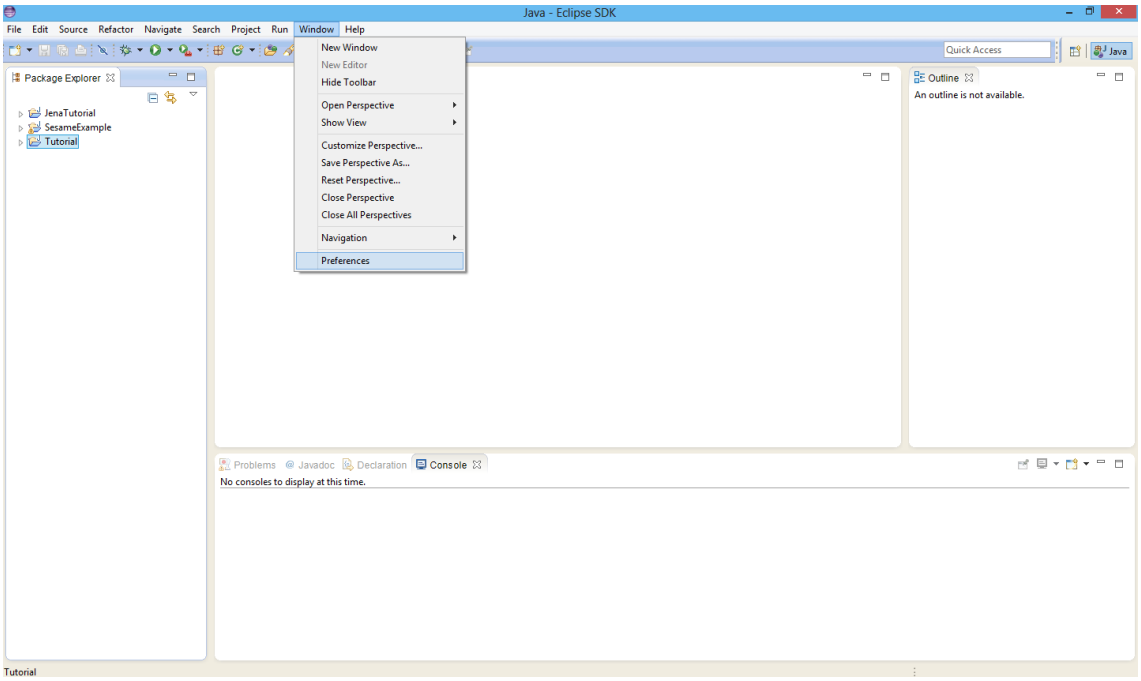


Fig 24 – Adding *Jena* libraries into *Eclipse* (1)

Then go to Java-Build Path-User Libraries and click the "New…" button.

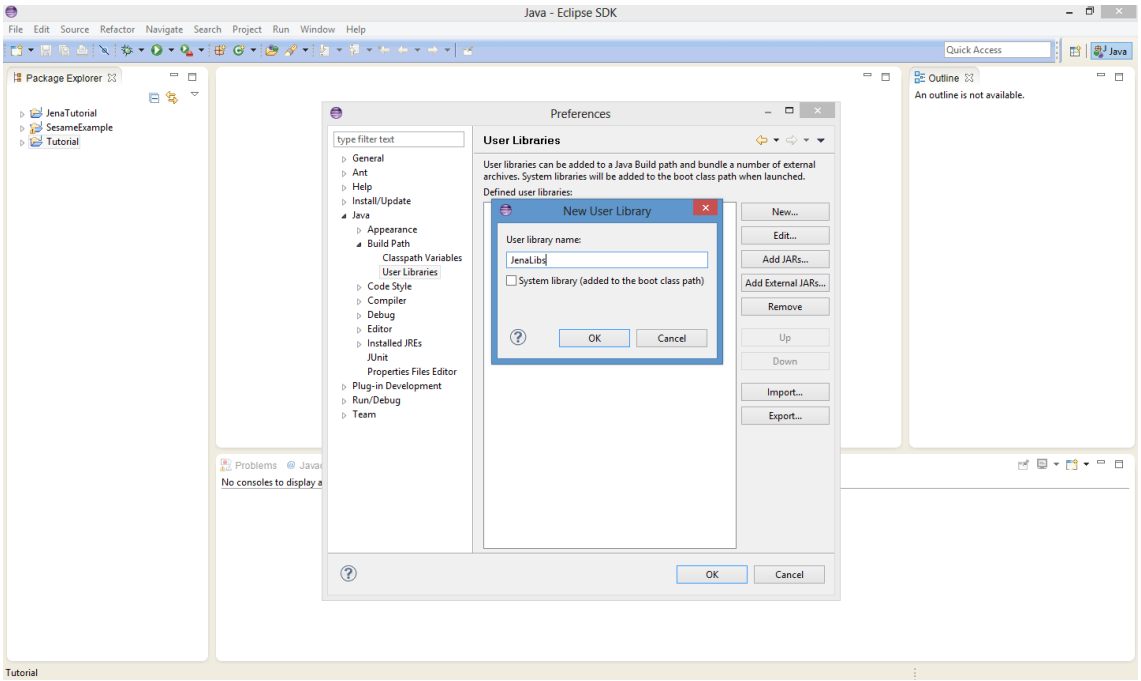


Fig 25 - Adding *Jena* libraries into *Eclipse* (2)

Give a name to the user library, for example, *JenaLibs*, and click *OK*. Then, select the library that you have already created, and press "Add External JARs…" button.

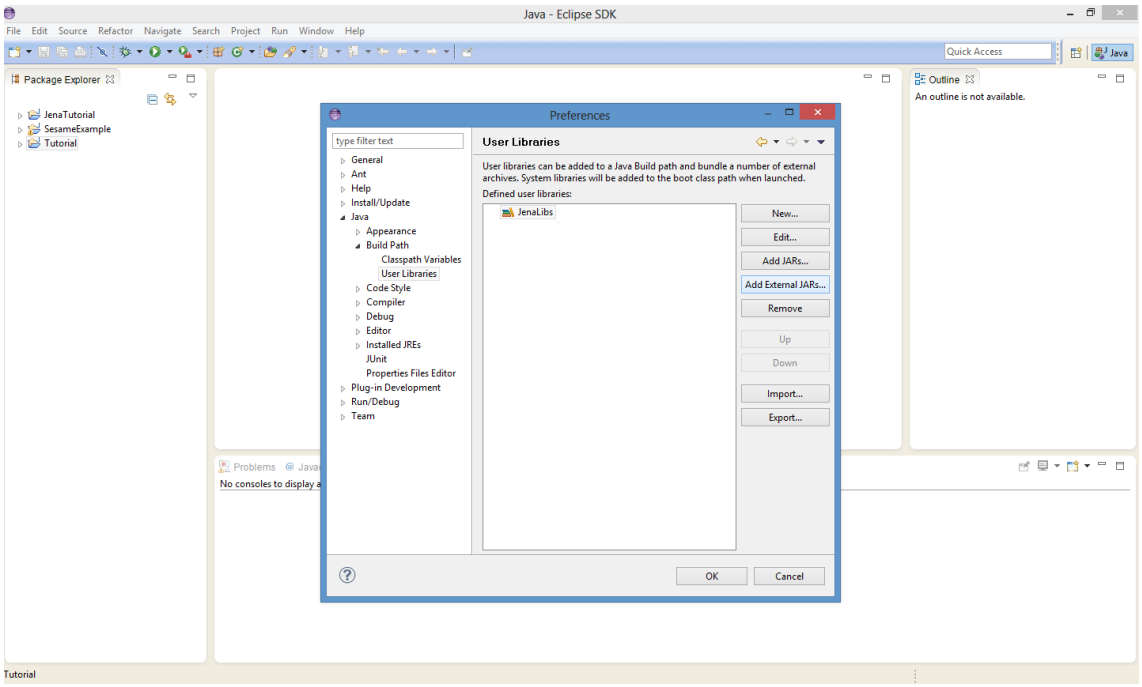


Fig 26 - Adding *Jena* libraries into *Eclipse* (3)

Now we have to add all the .jar files from Jena libraries which are in the *lib/* directory of our Jena install directory.

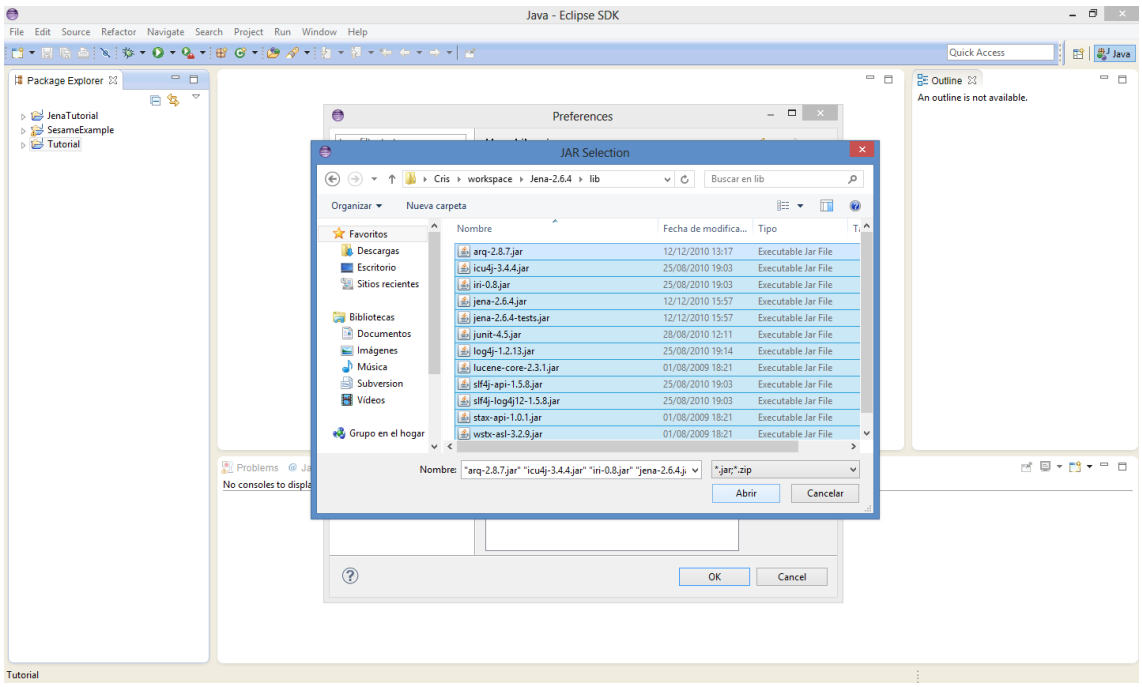


Fig 27 - Adding *Jena* libraries into *Eclipse* (4)

The result is that we have already created a user library called *JenaLibs* that contains the Jena libraries.

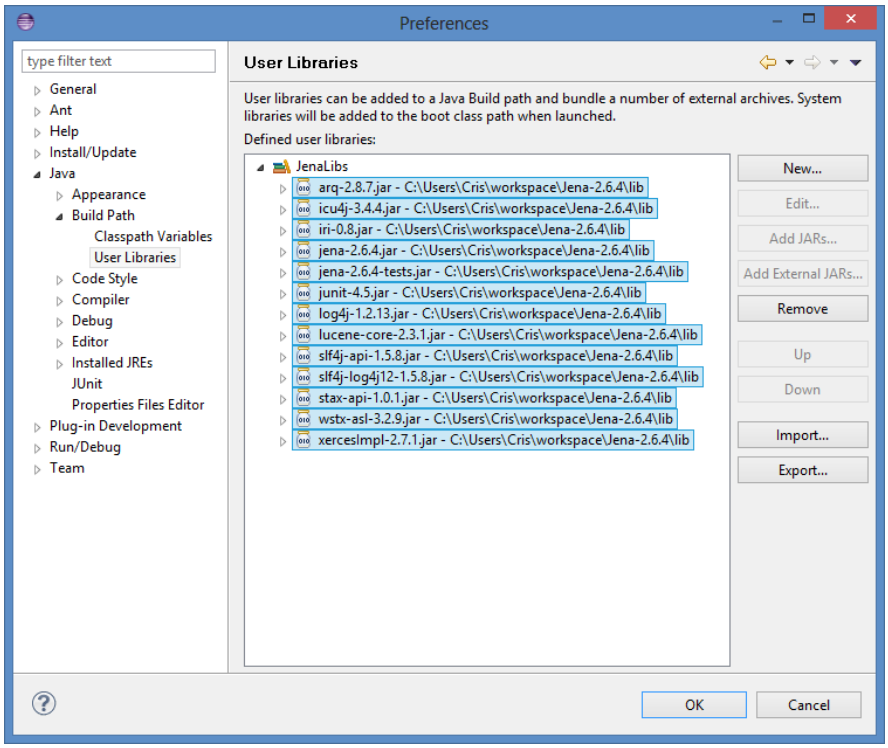


Fig 28 - Adding *Jena* libraries into *Eclipse* (5)

5. To use them into our Java project, we right-click on it in the explorer window and navigate to the Build Path menu option in order to add the *JenaLibs* library.

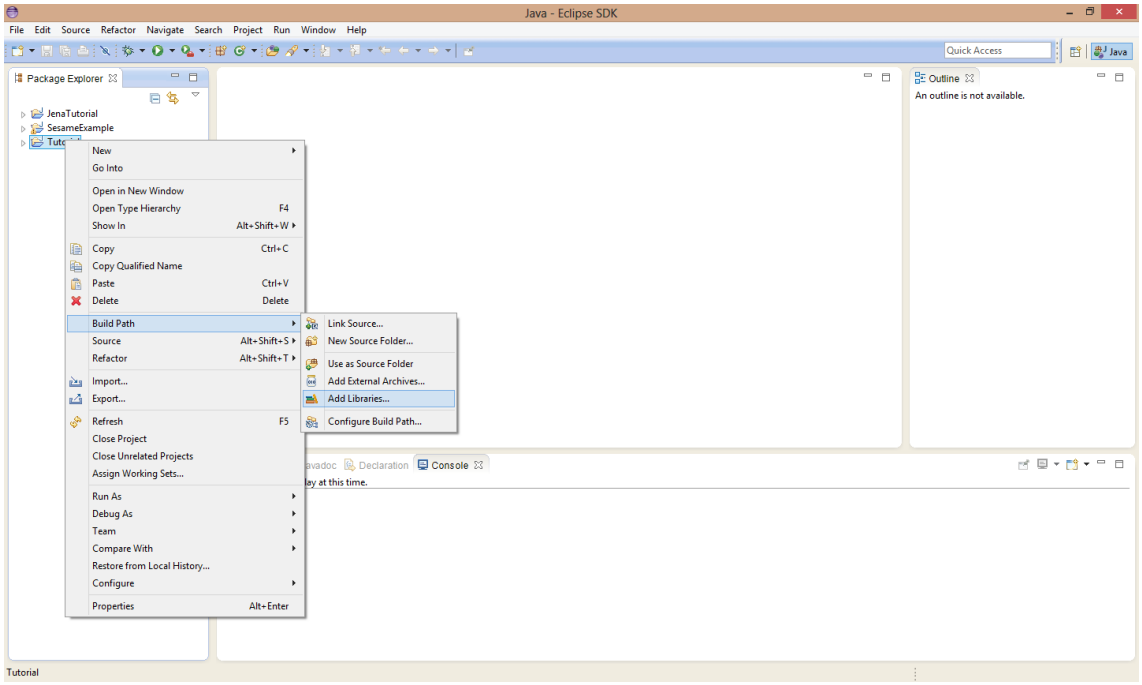


Fig 29 - Adding *Jena* libraries into our Java project (1)

Then, select the option “User Library” and click *Next*.

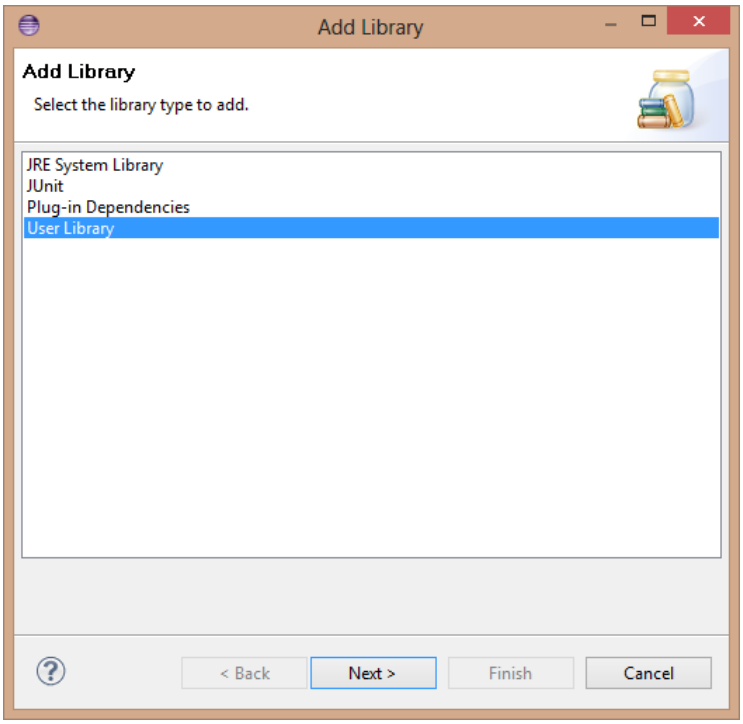


Fig 30 - Adding *Jena* libraries into our Java project (2)

Select *JenaLibs* and click *Finish* in order to add it to the classpath of our project.

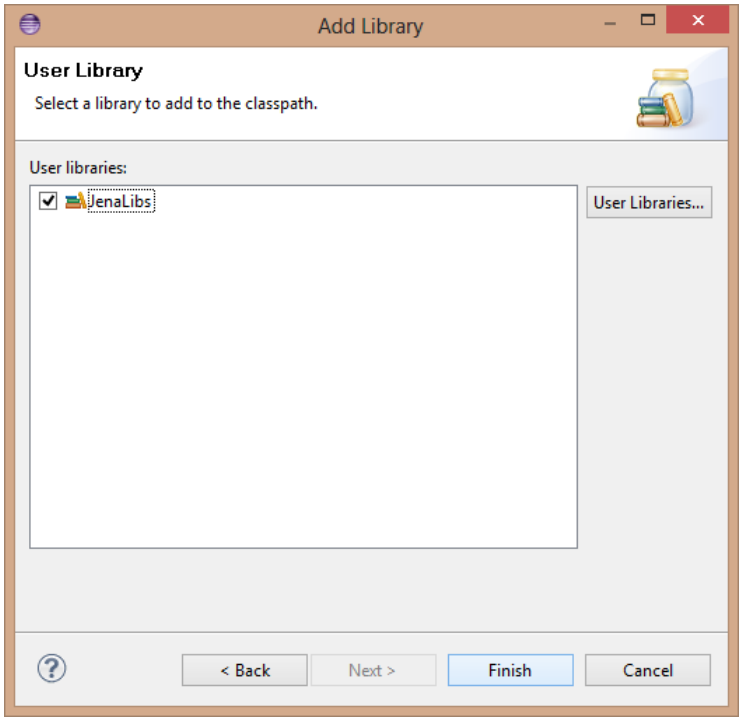


Fig 31 - Adding *Jena* libraries into our Java project (3)

And now, we can use Jena within Eclipse (see this tutorial http://www.iandickinson.me.uk/articles/jena-eclipse-helloworld/ for more detail).

4.3.2 Architectural Knowledge as Linked Data: step by step with Fuseki

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *Fuseki*.

1. First of all, we download the *Jena-Fuseki* package from http://www.apache.org/dist/jena/binaries/. It is a ZIP file that we have to unzip in order to use it.
2. Once we have this file unzipped, we can run the Fuseki server, writing the following line in the console: *fuseki-server --update --mem /ds*, where *--update* option allows us to update the dataset */ds*, and *--mem* option creates it as an empty and non-persistent dataset.

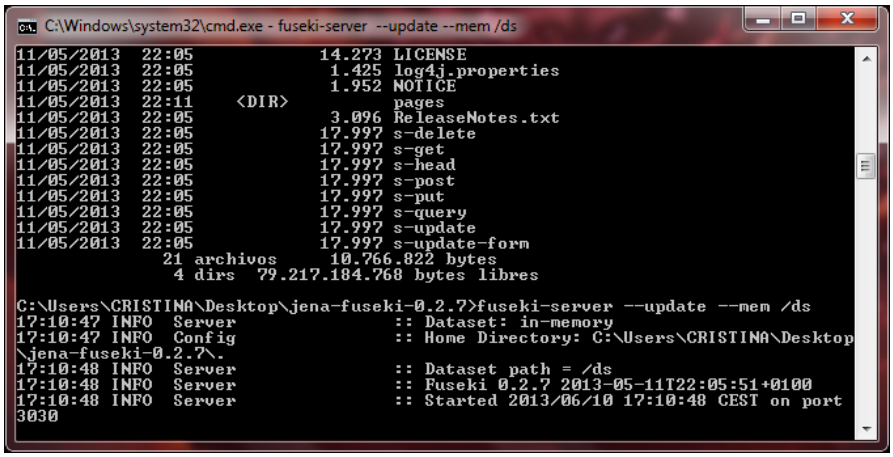


Fig 32 – Starting *Fuseki* server

Notice that if we run Fuseki with the *--loc=DIR* option instead of *--mem*, we can point it to an existing TDB dataset (and if it doesn't exist it will be created). As TDB datasets on disk are *persistent* anytime, when we run Fuseki with the *--loc* flag set to the same path (in this case *DIR*), it will use the same dataset and keep changes between sessions.

Since you'll have created the same dataset you can use Java code with the TDB jars to access this dataset from code directly without going via Fuseki.

3. As the server is started, we can go to http://localhost:3030/ which is the Fuseki interface to manage our data.

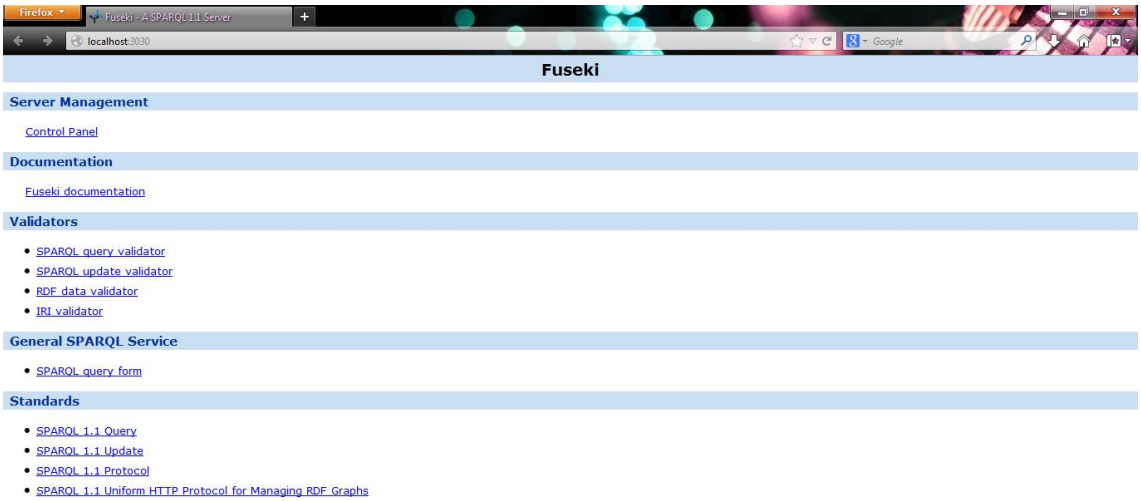


Fig 33 – *Fuseki* interface

4. To upload our RDF data to the Fuseki server, we have to go to *Control Panel* and select our dataset, */ds*. Then, in the *Fuseki Query* interface, we select the RDF file to upload into the server and click *Upload*.

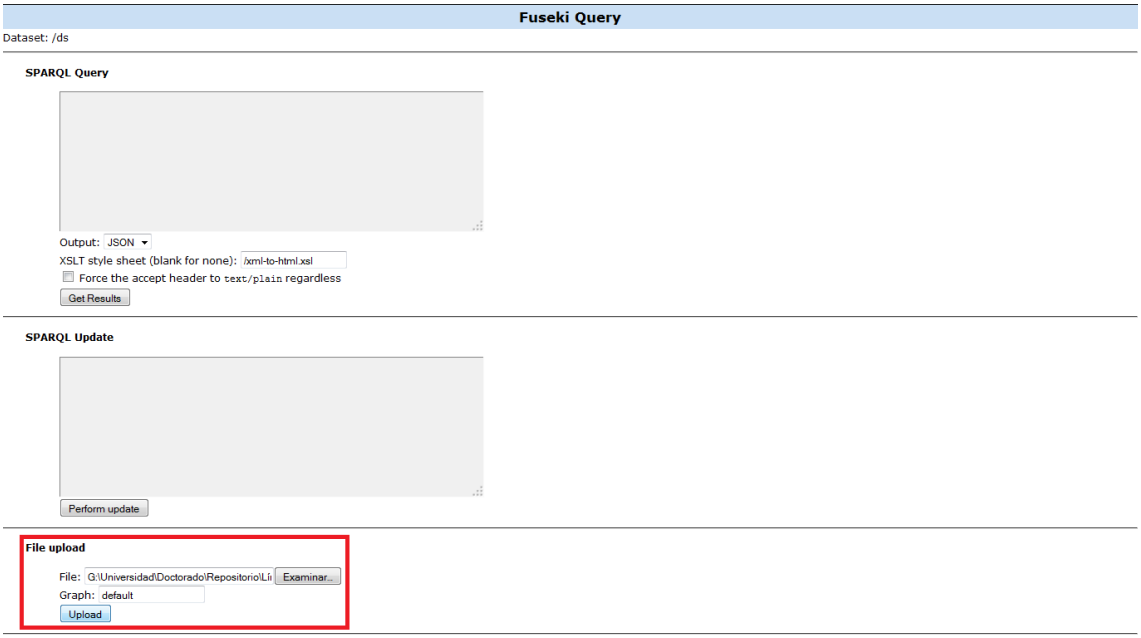


Fig 34 – Uploading an RDF file into *Fuseki* server

5. Once our RDF file is uploaded into the Fuseki server, we can make queries to manage this data. Go to http://localhost:3030/sparql.tpl and write in the *SPARQL Query* text area the following query: "SELECT * WHERE { ?s ?p ?o }". Then select the output type and click *Get Results*.

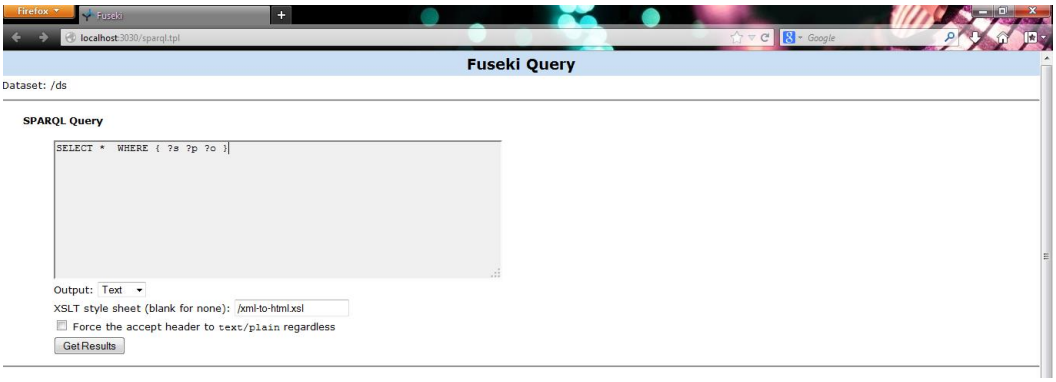


Fig 35 – Querying our data with *Fuseki* and SPARQL

Finally, the results of the query are presented, in this case, in text format:

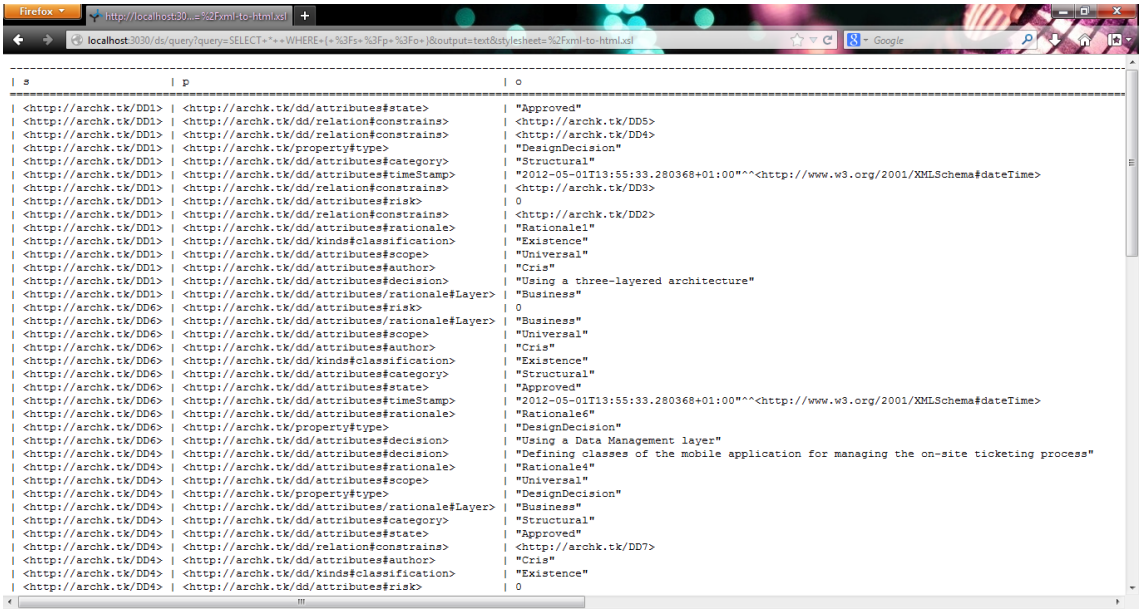


Fig 36 – Query results with *Fuseki* and SPARQL

4.4 TopBraid Suite

TopBraid Suite [12] is a compendium of products that offers semantic technology to help user in several scopes of applicability, like connect data, systems and infrastructure or build flexible applications from linked data models.

All components of the suite work within an open architecture platform built specifically to implement W3C standards for integration and combination of data drawn from diverse sources (see Fig 37). These components are *TopBraid Composer*, for design data, *TopBraid Ensemble*, to assemble data, and *TopBraid Live*, to interact with data. They will be explained next.

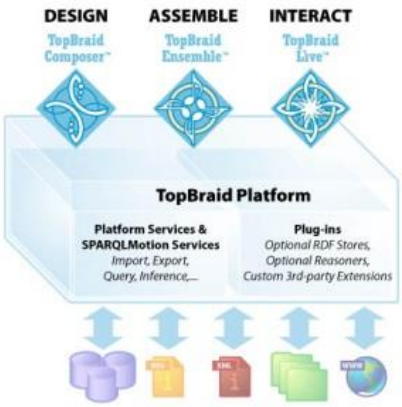


Fig 37 – TopBraid Suite Platform

TopBraid Composer (TBC) [13] is an Eclipse plug-in that offers an enterprise-class modelling environment for developing Semantic Web ontologies, building Semantic applications and converting data and models to/from RDF or OWL. Therefore, this platform provides complete support for developing and managing ontologies and Linked Data.

In that way, these are the main features of TopBraid Composer:

- Editing Ontologies and RDF Data
- Developing Semantic Web Applications
- Application Development Tools
- Importing Data Sources
- Exporting Data Sources
- Working with XML and Table files
- Developing Web Data (Microdata, RDFa)
- Data Integration

In addition, TopBraid Composer is fully compliant with W3C standards, such as RDF, RDFS, OWL or SPARQL. In this manner, all resources in the RDFS vocabulary are available in TopBraid Composer, as well as all OWL language constructs. TBC also ensures not to have syntactic errors in RDF data created in a text editor.

TBC provides an SPARQL tab to execute SPARQL queries and we can also save query results into a file. Furthermore, TBC can be used with Subversion (SVN) and allows opening some file formats within its environment, like RDF/XML, Turtle, N3, N-triples, Excel documents and XML.

On the other hand, we have *TopBraid Ensemble*, [12] a semantic web application assembly toolkit for rapid configuration and delivery of dynamic business applications, so it allows us to create model-driven applications. This tool provides a quick route from an ontology and RDF data to a working web application using pre-built model-driven *Rich Internet Application* (RIA) components implemented with Adobe Flex. Notice that to indicate how this application works is beyond the scope of this section, due to it is unlikely that we will use it.

Finally, we have *TopBraid Live* (TBL) [14] which is an enterprise SOA-capable Semantic Web application and Linked Data platform. It is a server for deploying flexible, model-driven applications and dynamic, on-demand integration of data from diverse sources.

So, it has so many benefits, such as:

- *A unified, standards-based data integration process* that combines data and schemas from internal, external, structured and unstructured sources using connectors to relational data, web services, XML files, JSON documents, spreadsheets, flat files, e-mail, content management systems and more.
- *An orchestration engine* for coordinating access to multiple sources (real time, ETL, etc.) including relational databases, XML, spreadsheets and web services.
- *Web Services* – out-of-the-box RESTful Linked Data services plus a comprehensive library of ready-to-use modules and functions for rapid creation of custom composite web services from existing data and systems.
- *Provenance and change management* that completes audit trail of all changes, versioning.
- *Business rules* – Flexible and powerful reasoning integrates RDFS and OWL inferencing with custom, domain-specific business rules.
- *SPARQL Endpoint*, given that all information managed by TopBraid is accessible through SPARQL. In addition to full *SPARQL 1.1* Query Language support, TopBraid Live includes many extensions provided as SPARQL functions.
- *Linked Data*, given that all information managed by TopBraid is available as RESTful web services returning a choice of serializations.
- *Named graph architecture* – TopBraid Live workspace is a quad store fully implementing named graph architecture.
- *Model-based* – fully dynamic, flexible and agile.
- *Rapid application development* – using pre-built UI components, application templates and modules.
- *Support for the entire application development lifecycle* – seamless migration from development in TopBraid Composer to deployment in TopBraid Live.

There are two editions of this product. On the one hand, we have *TopBraid Live Enterprise Server* that provides full Internet and intranet access to the applications developed with TopBraid Suite. This server is deployed on a web container, such as Tomcat and supports multi-user access from distributed clients. Users may access it through web services and HTML and JavaScript applications, built using the TopBraid tools.

On the other hand, we have *TopBraid Live Personal Server* which is included in the Maestro Edition of TopBraid Composer (TBC-ME) in order to replicate Enterprise server features on a local machine within an IDE environment. In this manner, it executes in a single-user environment, providing a powerful tool for designing, developing and testing applications before deploying them to an Enterprise server. You can access the Personal server from any browser, running on the same machine as TBC. Once applications are designed with TBC-ME and the Personal Server, you can deploy them with a click of a button to the Enterprise Server for full Internet and intranet access.

Notice that TopBraid Live provides Federation Services in order to perform federated queries across multiple databases which is an essential feature for our aim.

Fig 38 represents the TopBraid *Enterprise Reference Architecture* (ERA) for the whole TopBraid Suite and its three components (TopBraid Composer, TopBraid Ensemble and TopBraid Live).

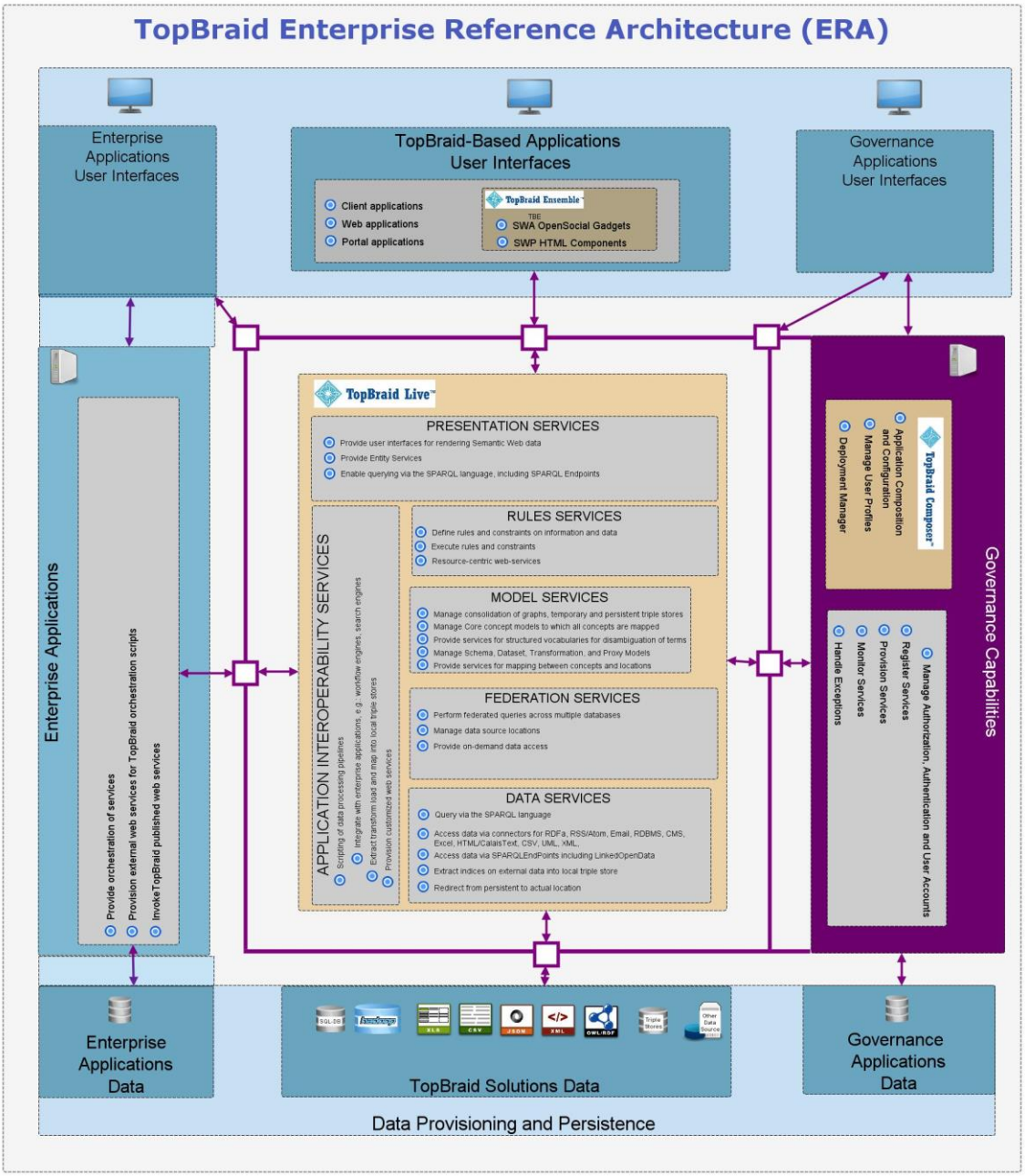


Fig 38 – TopBraid *Enterprise Reference Architecture* (ERA)

4.4.1 Architectural Knowledge as Linked Data: step by step with TopBraid Suite

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *TopBraid Suite*.

Let's start with *TopBraid Composer*.

1. First of all, we have to download TopBraid Composer from http://www.topquadrant.com/products/TB_install.php. Three versions are available: *Maestro Edition*, *Standard Edition* and *Free Edition*, from the most comprehensive version

of TBC to the lightest release that neither includes support and maintenance, nor the graph view to illustrate the RDF graph, respectively. Apart from the different versions, we can select the specific OS between Windows, Mac and Linux.

2. Once we have downloaded TBC, we have to unzip the corresponding file and execute *TopBraid Composer.exe* to open TBC environment.

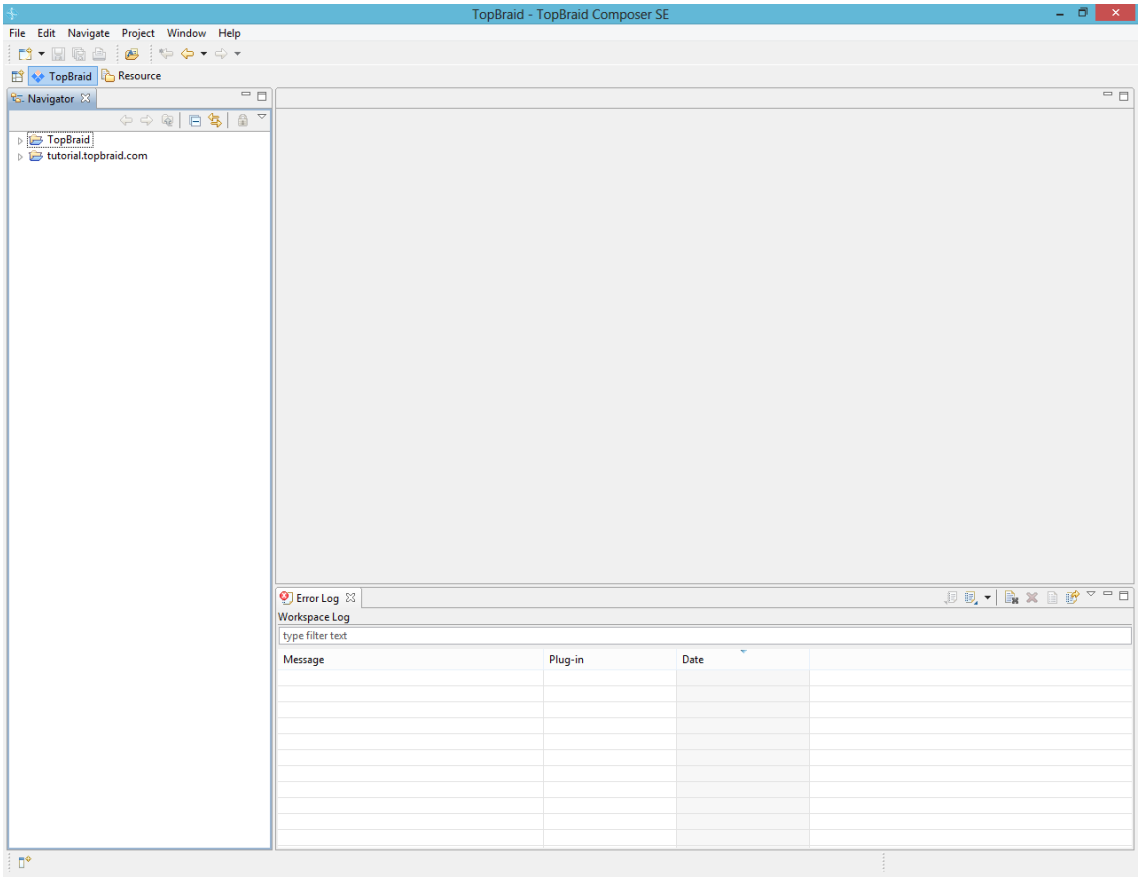


Fig 39 – *TopBraid Composer* overall interface

3. We can create a new project. Go to File-New-Project.

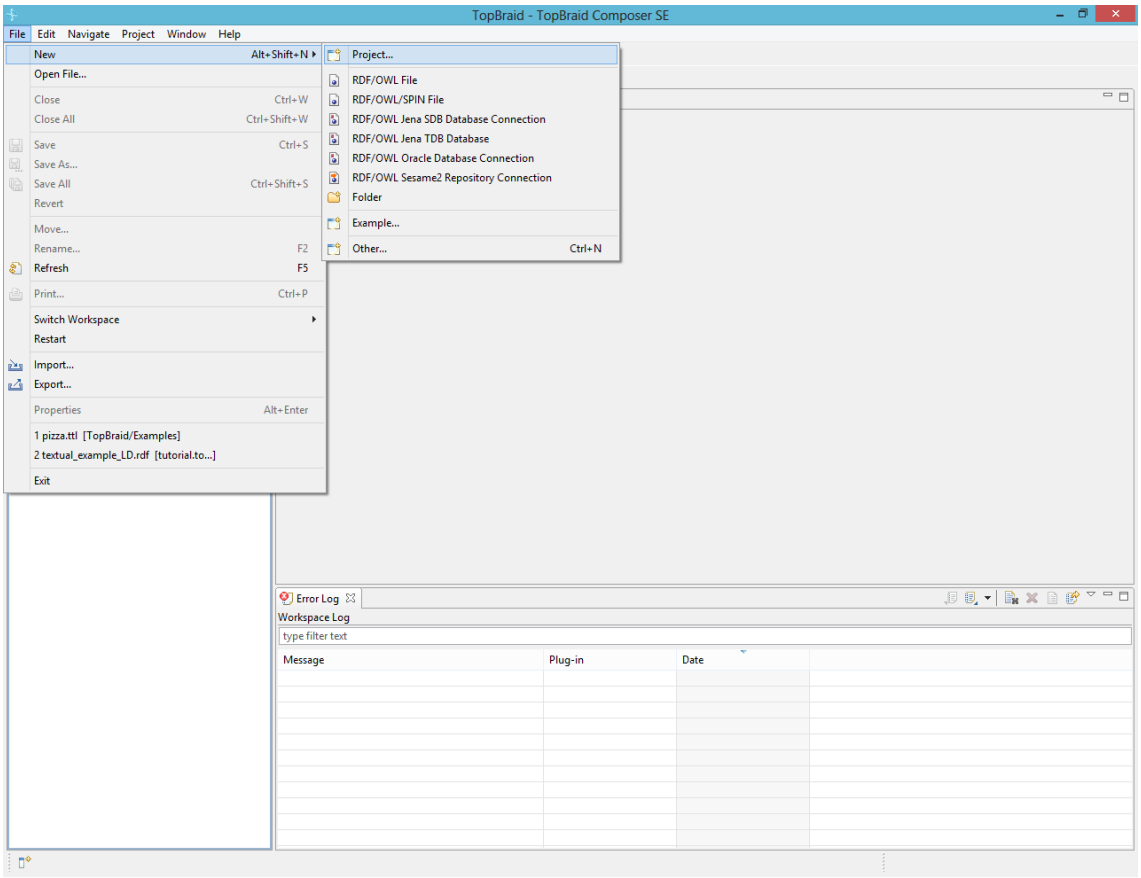


Fig 40 – Creating a new project with TBC (1)

Then, we choose *Project* from the General folder and click *Next*.

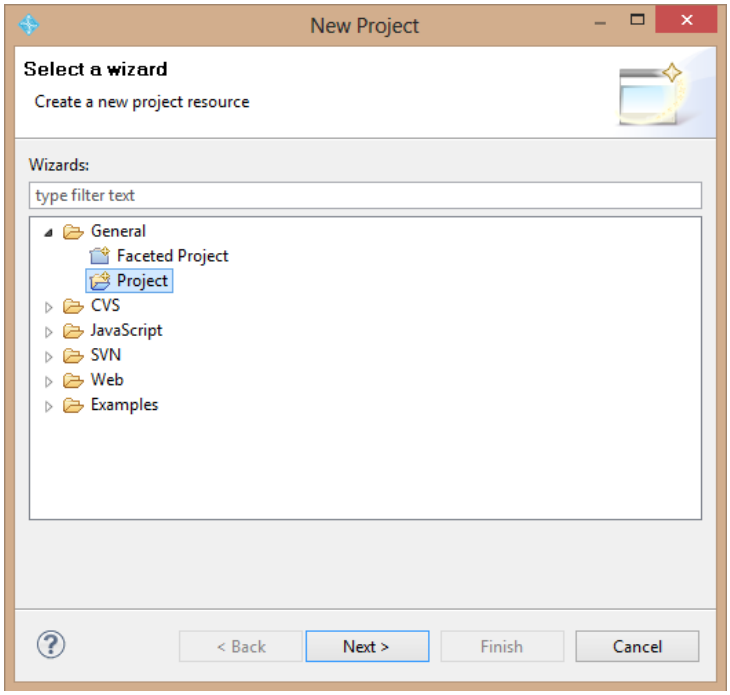


Fig 41 – Creating a new project with TBC (2)

Finally, we give a name to the project and click *Finish* to create it in our workspace.

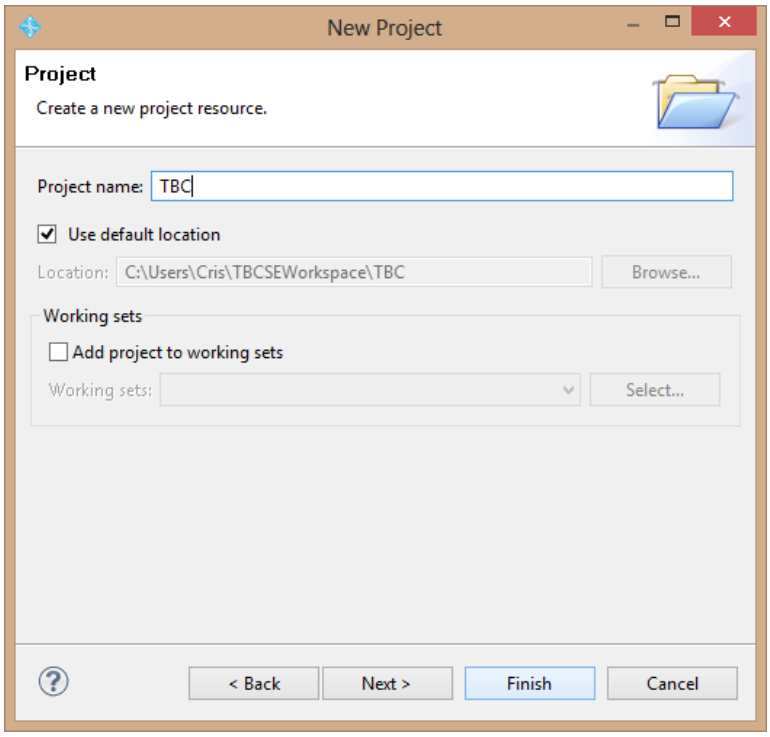


Fig 42 – Creating a new project with TBC (3)

4. Once we have already created our TBC project, we may paste our RDF file into it, so we can manage its RDF data.

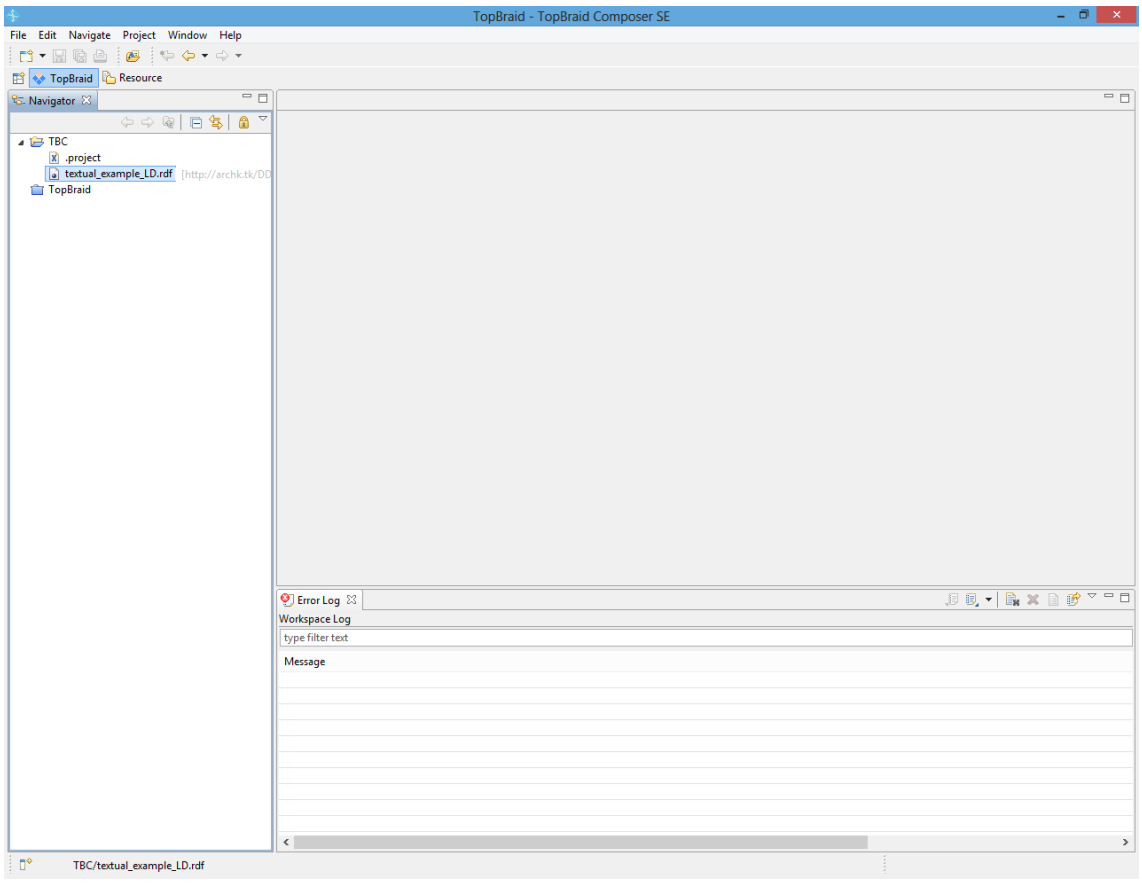


Fig 43 – Adding our RDF data into a TBC project

Then, we are allowed to open this RDF file by double-clicking it, using the editor provided by TBC.

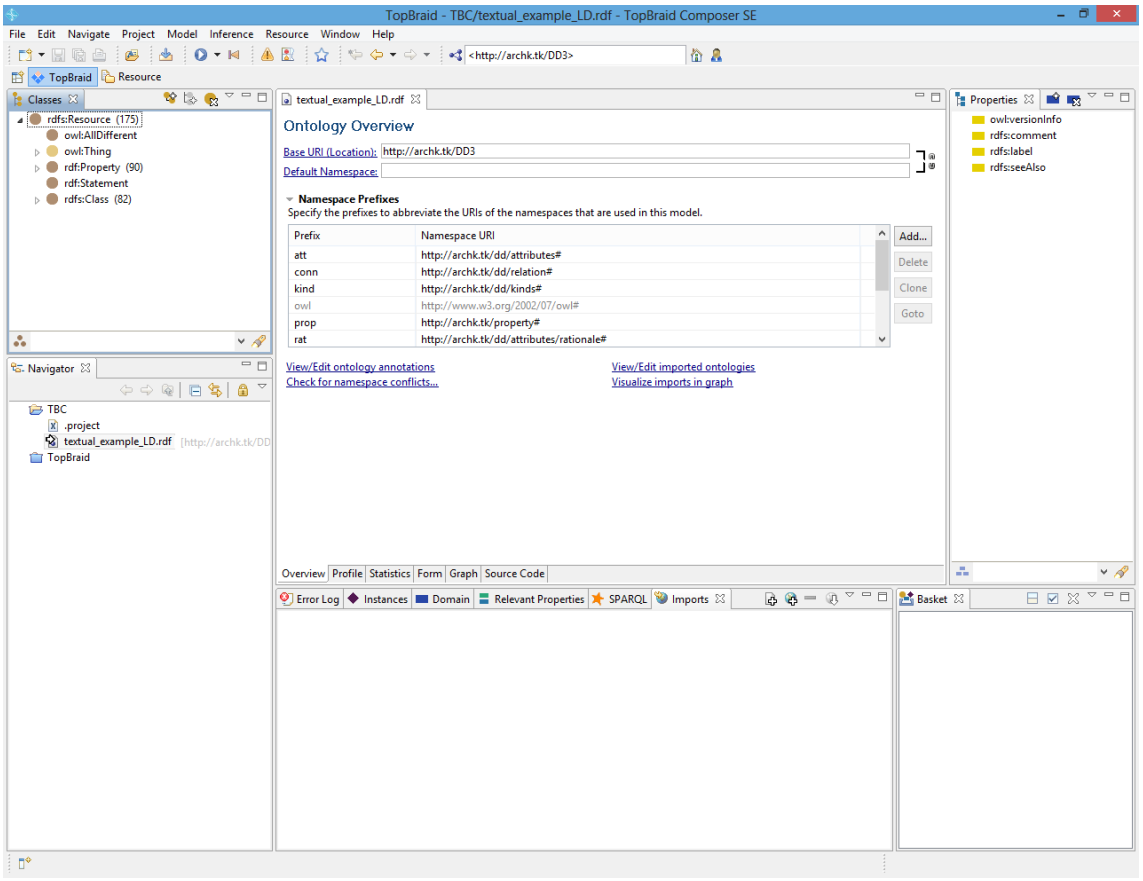


Fig 44 – Opening our RDF file with TBC

5. Once we have opened this file in TBC, we can navigate through its different tabs: Overview, Profile, Statistics, Form, Graph or Source Code.

   For example, the Form tab shows all properties regarding to a specific resource, in this case *<http://archk.tk/DD3>*.

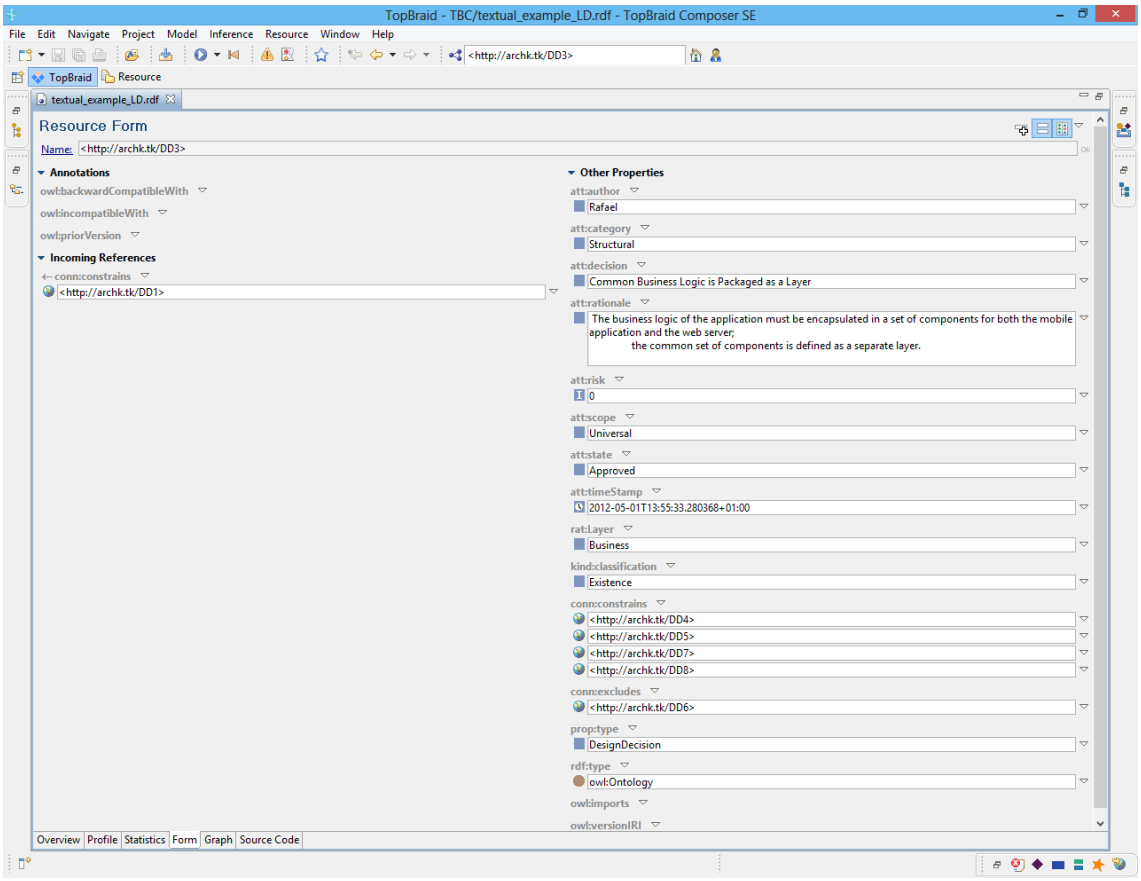


Fig 45 – Form tab in TBC

Another useful component of TBC is the Graph tab (not available in Free Edition of TBC) which shows a graph view of the RDF data, in this case, the AK decision network, including relationships between design decisions.

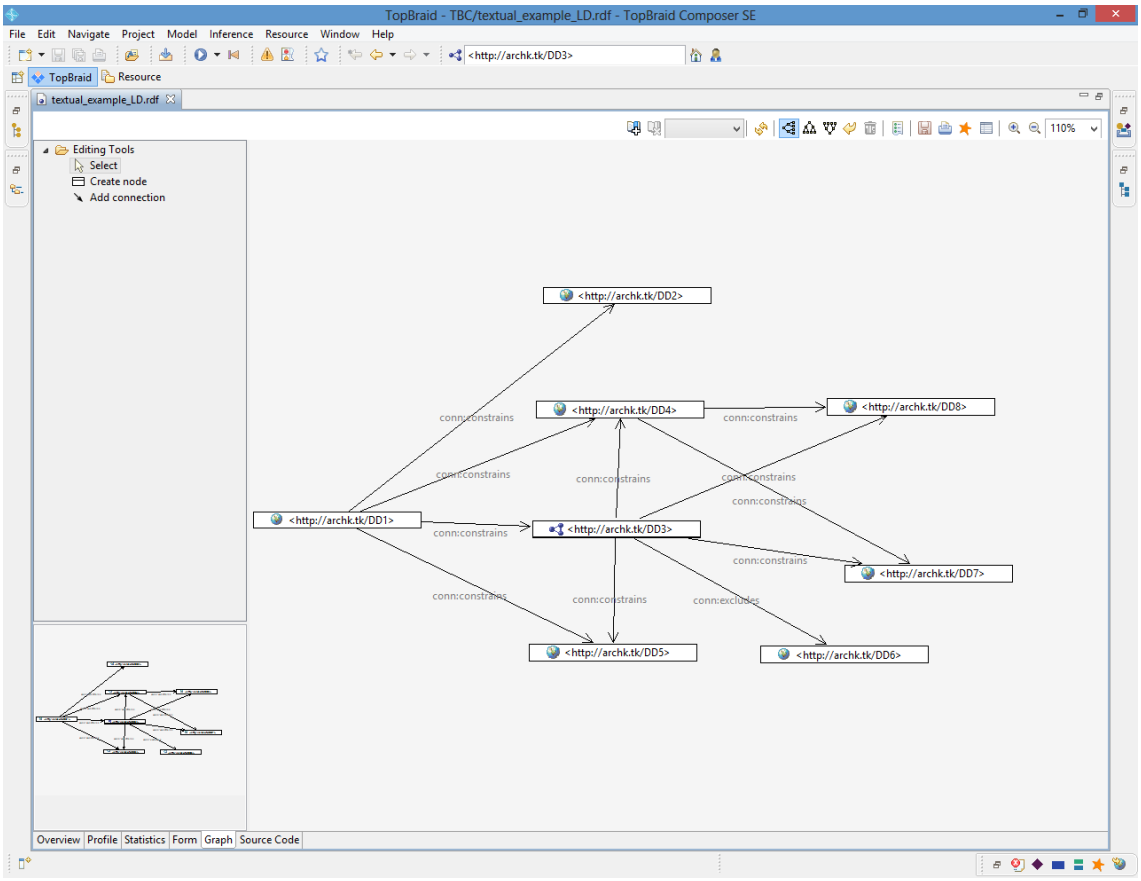


Fig 46 – Graph tab in TBC (1)

We can also expand these nodes in the graphic view in order to show their related information, as you see in Fig 47.

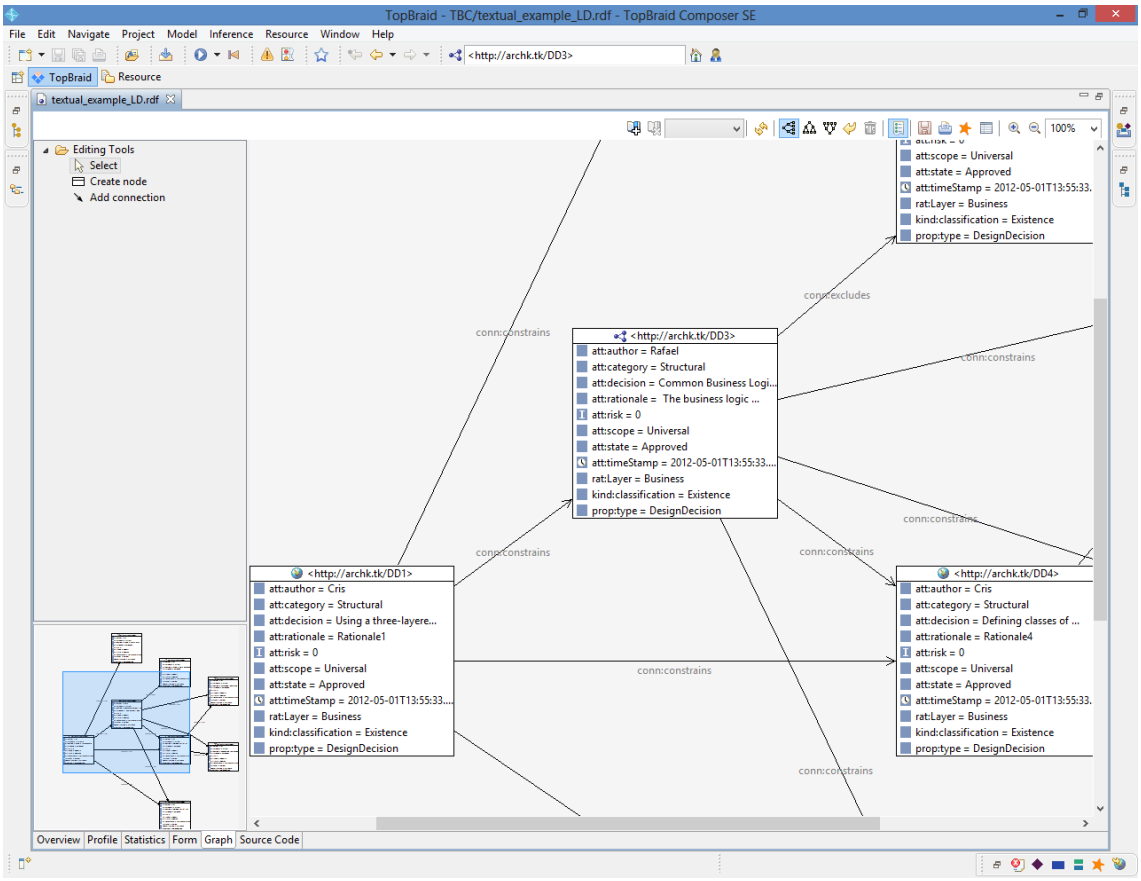


Fig 47 – Graph tab in TBC (2)

6. We may also be interested in execute some SPARQL queries against our RDF data. TBC offers another component which is a SPARQL view that allows us to run this type of queries.

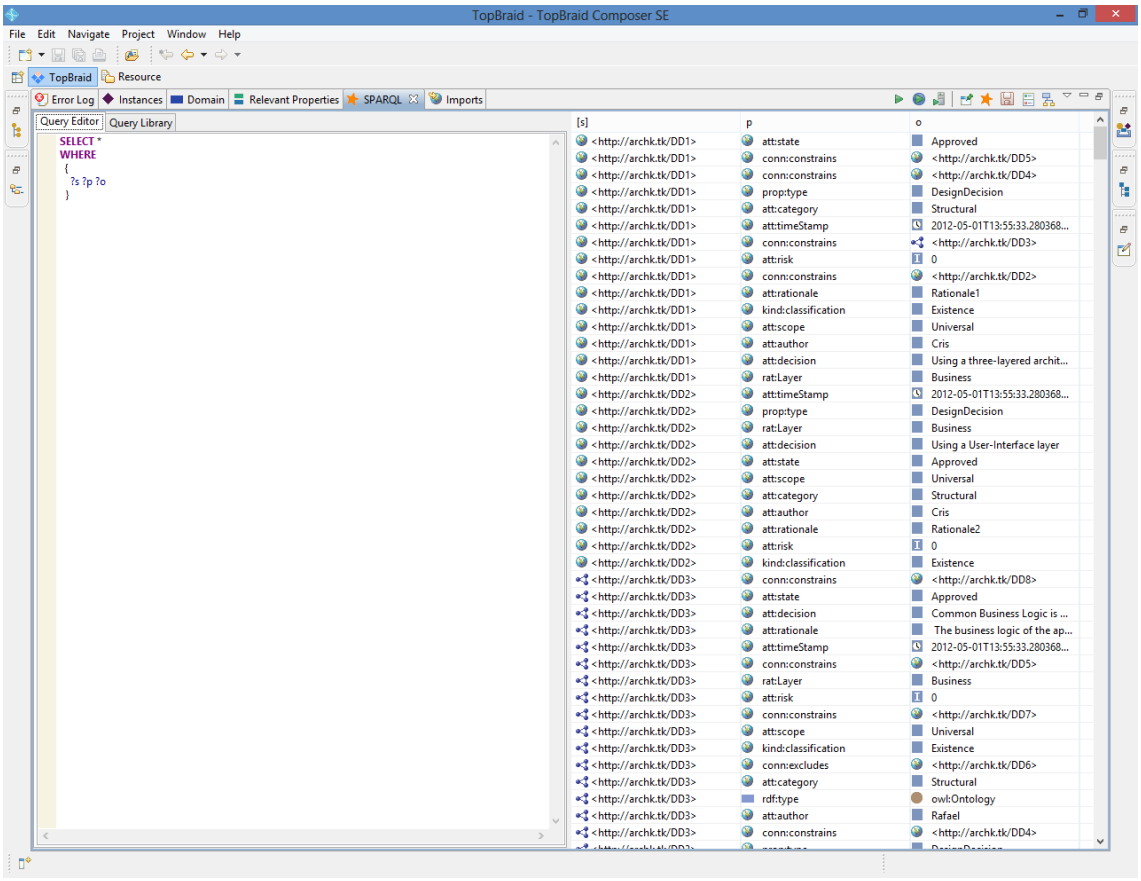


Fig 48 – SPARQL view in TBC

At this time, we have to know that applications developed using TopBraid Composer can be deployed on TopBraid Live whether these applications use a TopBraid Ensemble web-based interface, a custom web-based interface or a programmatic RESTful interface.

Notice that only the Maestro Edition of TopBraid Composer includes the Personal Edition of TopBraid Live so that you can test web-based applications while developing them. When you are ready to deploy these applications for use by multiple users, the Enterprise Edition of TopBraid Live provides a scalable platform for this.

Now, we are going to explain the steps for using *TopBraid Live Personal Server*. Notice that you have to download the TBC Maestro Edition (TBC-ME) in order to use it. We are not going to explain how to use *TopBraid Live Enterprise Server* given that it is not available as free download, but you can contact sales@topquadrant.com for more information and go to http://www.topquadrant.com/docs/tbl/42install/installation.html to follow a complete installation guide.

So, let's continue with TopBraid Live Personal Server within TBC-ME.

1. First of all, open TBC-ME by double-clicking its *TopBraid Composer.exe* file.
2. Then, ensure that the Personal Server is started. It is started by default but if not, go to the *Help* menu and click on the option "Start Personal TopBraid Server". This option should change as "Stop Personal TopBraid Server" when this server is started.

3. At this point, TBC-ME is launching the TopBraid Live Personal Server in background at http://localhost:8083. You can use http://localhost:8083/tbl to access the admin console (see Fig 49).

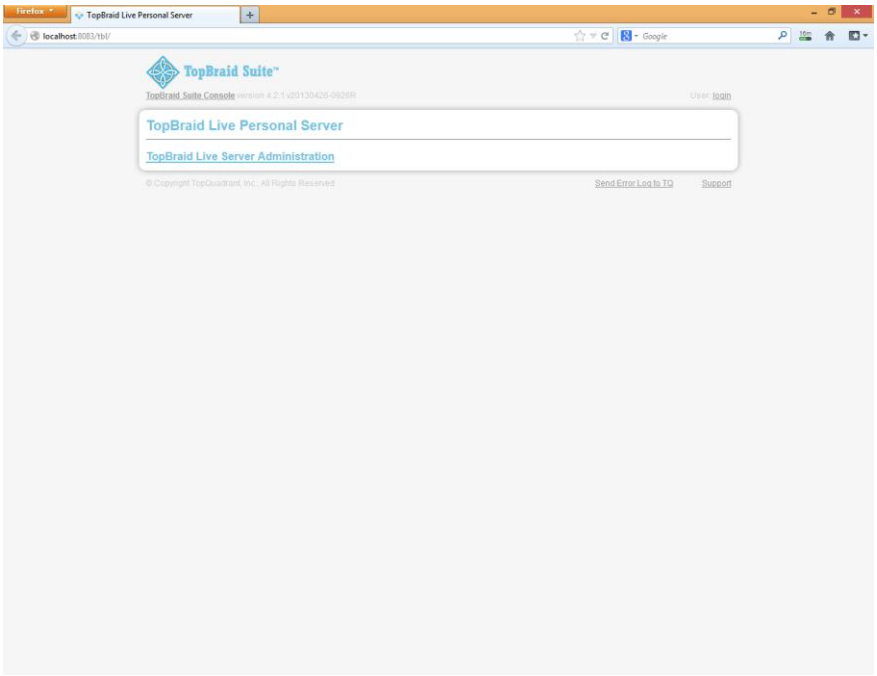


Fig 49 – *TopBraid Live Personal Server* admin interface

4. Then, we can click on the *TopBraid Live Server Administration* link in order to see the different options offered by this application.

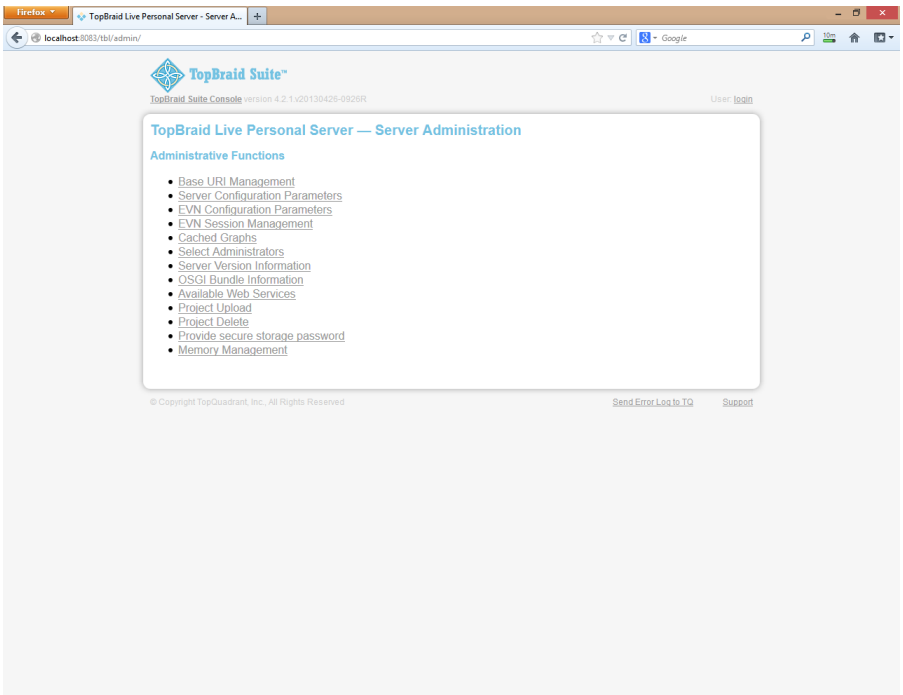


Fig 50 – *TopBraid Live Personal Server* admin options

For example, if we go to the first one, *Base URI Management*, we can see the different projects that we have in our workspace and their own files:

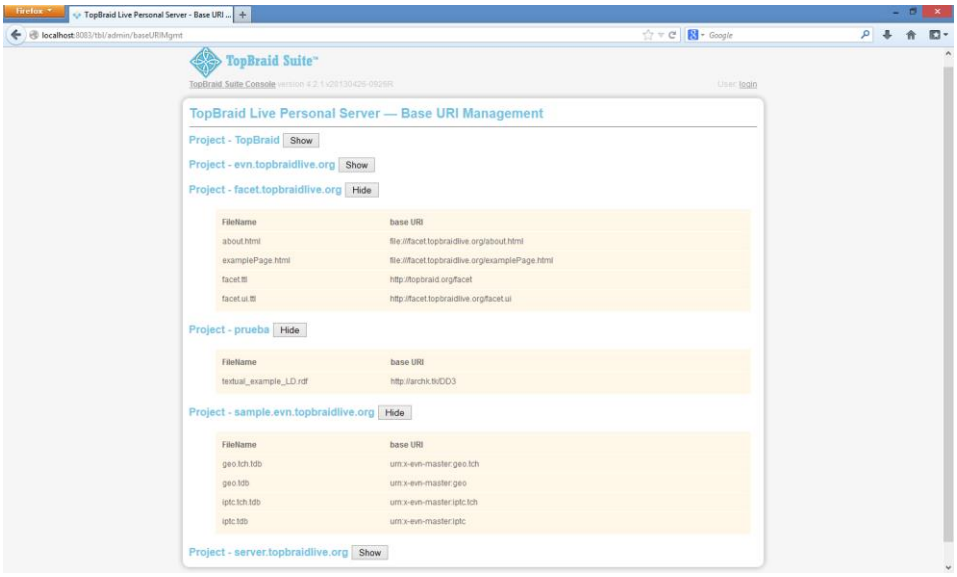


Fig 51 – Base URI Management (TBL Personal Server)

Another administrative function is *Server Configuration Parameters* that allows you to visualize and edit the current server configuration.

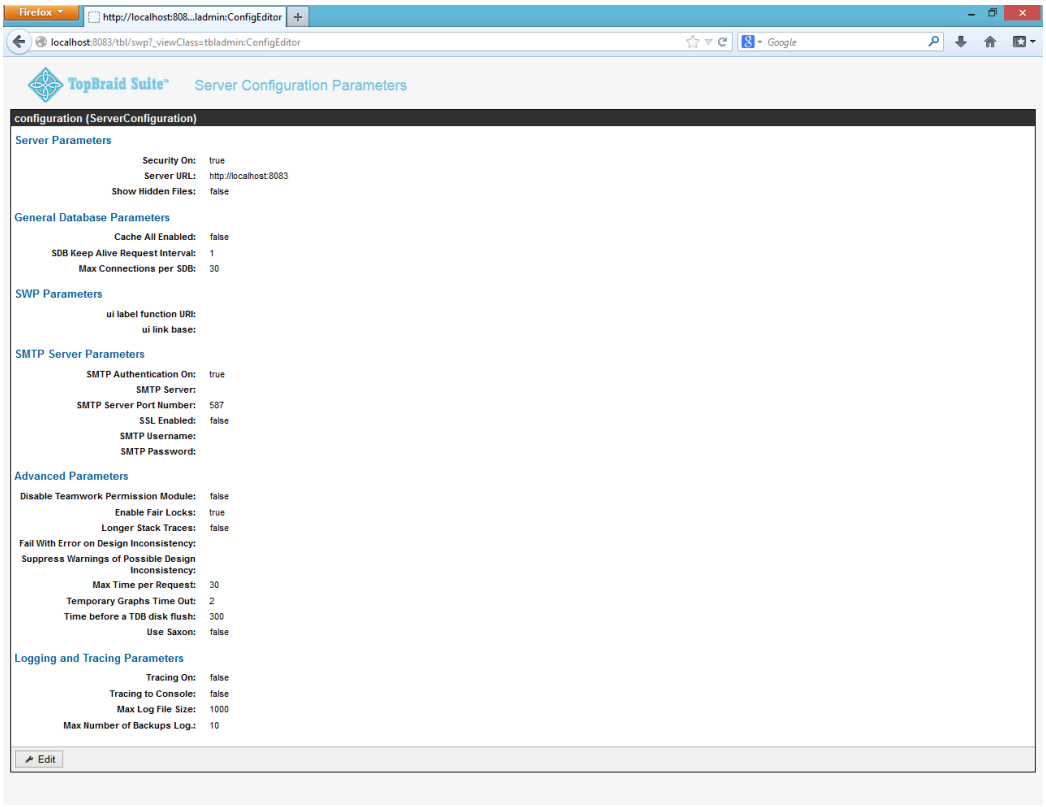


Fig 52 – Server Configuration Parameters (TBL Personal Server)

As another example, we may go to *Project Delete* in order to eliminate one or more projects in our workspace. Notice that not all the projects can be removed, as you see in Fig 53.

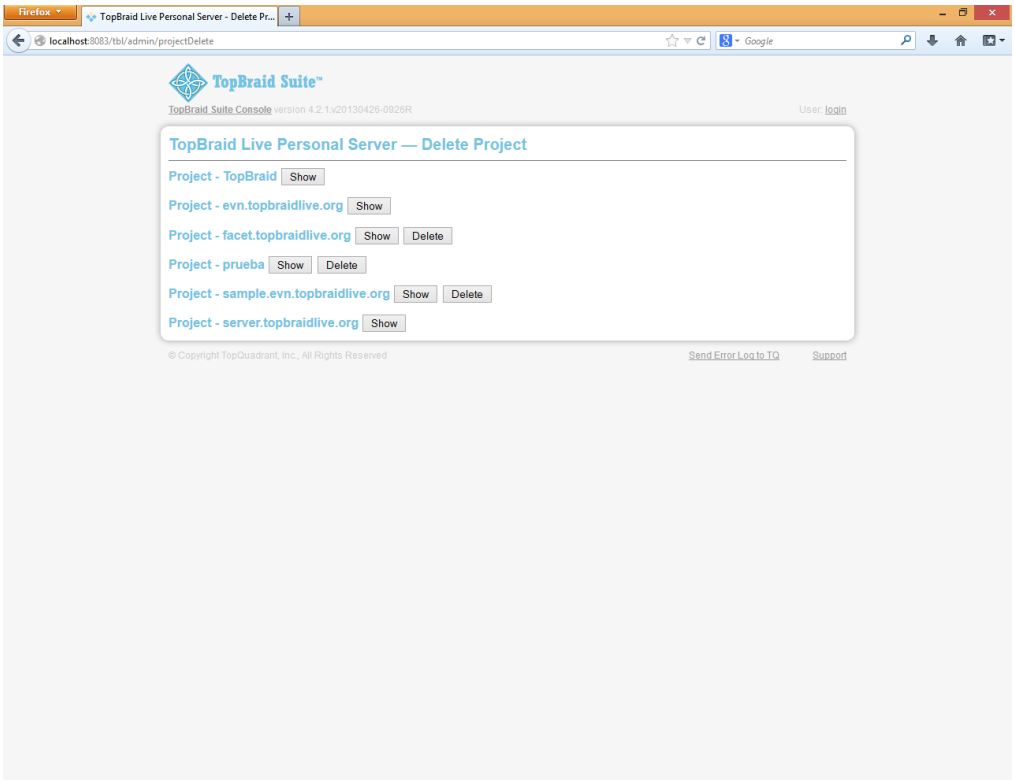
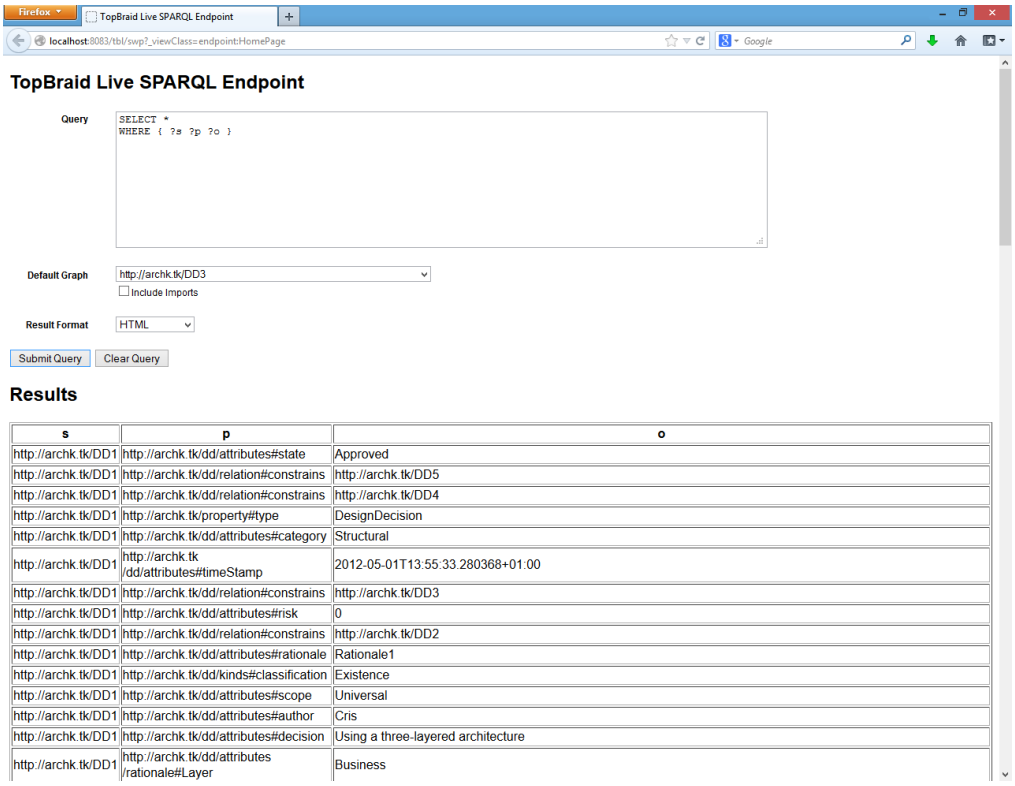


Fig 53 – Project Delete (TBL Personal Server)

5. If we want to launch SPARQL queries using this server, we can go to http://localhost:8083/tbl/sparql and execute them, for example: *SELECT * WHERE { ?s ?p ?o}*. The output of this query is shown in Fig 54. Notice that the result format may be in XML, HTML, Text/CSV and Text/TSV.



s	p	o
http://archk.tk/DD1	http://archk.tk/dd/attributes#state	Approved
http://archk.tk/DD1	http://archk.tk/dd/relation#constrains	http://archk.tk/DD5
http://archk.tk/DD1	http://archk.tk/dd/relation#constrains	http://archk.tk/DD4
http://archk.tk/DD1	http://archk.tk/property#type	DesignDecision
http://archk.tk/DD1	http://archk.tk/dd/attributes#category	Structural
http://archk.tk/DD1	http://archk.tk /dd/attributes#timeStamp	2012-05-01T13:55:33.280368+01:00
http://archk.tk/DD1	http://archk.tk/dd/relation#constrains	http://archk.tk/DD3
http://archk.tk/DD1	http://archk.tk/dd/attributes#risk	0
http://archk.tk/DD1	http://archk.tk/dd/relation#constrains	http://archk.tk/DD2
http://archk.tk/DD1	http://archk.tk/dd/attributes#rationale	Rationale1
http://archk.tk/DD1	http://archk.tk/dd/kinds#classification	Existence
http://archk.tk/DD1	http://archk.tk/dd/attributes#scope	Universal
http://archk.tk/DD1	http://archk.tk/dd/attributes#author	Cris
http://archk.tk/DD1	http://archk.tk/dd/attributes#decision	Using a three-layered architecture
http://archk.tk/DD1	http://archk.tk/dd/attributes /rationale#Layer	Business

Fig 54 – TopBraid Live SPARQL Endpoint

4.5 Sesame

Sesame [15] is an open source Java framework for storing and querying RDF data, similar to Jena (see section 4.3). This framework is fully extensible and configurable with respect to storage mechanisms, inferencers, RDF file formats, query result formats and query languages.

This framework isn't very useful without implementations of various APIs. The core of the Sesame framework is the *RDF Model API* which defines how the building blocks of RDF (statements, URIs, blank nodes, literals, graphs and models) are represented. Another API is the *Repository API* which is the central access point for Sesame repositories. Its purpose is to give a developer-friendly access point to RDF repositories, offering various methods for querying and updating the data in an easy way.

Therefore, Sesame supports SPARQL querying, a memory-based and a disk-based RDF store and RDF Schema inferencers. It also supports most popular RDF file formats and query result formats.

In addition, we can use the Web application *OpenRDF Workbench* that allows us to interact with Sesame. It is possible given that this Web application provides a web interface for querying, updating and exploring the repositories of a Sesame server.

4.5.1 Architectural Knowledge as Linked Data: step by step with Sesame

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *Sesame*.

In particular, we are going to explain how to use OpenRDF Workbench to manage RDF data.

- First of all, we have to download a Java servlet container, like Tomcat or Jetty, in order to set up Sesame as a standalone server. We choose Jetty as it has a very simple installation and is developed by Eclipse Foundation. Go to http://download.eclipse.org/jetty/stable-9/dist/ and select the latest distribution that will be a ZIP file. At the time of writing, the latest version is 9.0.3.v20130506.
- Next, we have to unzip this file and writing the following line in the console in order to start Jetty Java servlet: *java –jar start.jar*. Then, we can confirm that the installation of Jetty has been successful, going to http://localhost:8080 through our browser.

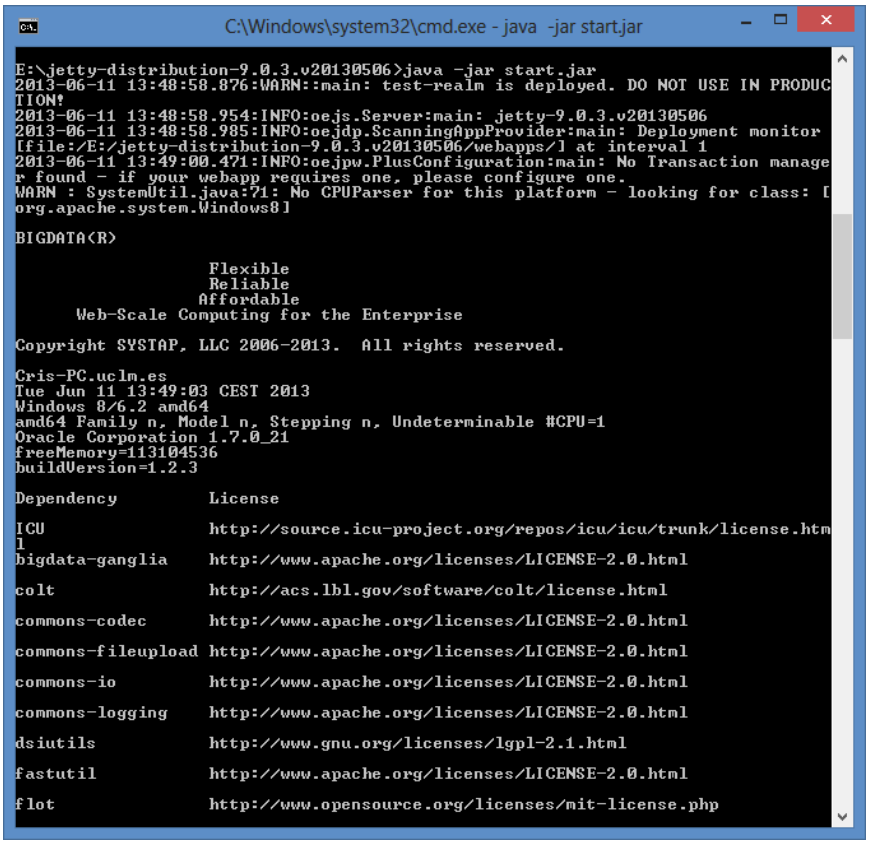


Fig 55 – Starting Jetty

After that, we have to go back to our prompt and hit *Ctrl-C* to stop Jetty because we haven't included the OpenRDF Workbench WAR files yet, which is the next step.

- Download the OpenRDF Workbench latest release from http://sourceforge.net/projects/sesame/files/Sesame%202/ that will be a ZIP file. At the time of writing, the latest version is 2.7.1. Then unzip this file and copy the archives *openrdf-sesame.war* and *openrdf-workbench.war* in order to paste them into the *webapps* folder of our Jetty installation.
- Then, we have to restart Jetty, writing *java –jar start.jar* in the console, always within our Jetty directory.
- Go to http://localhost:8080/openrdf-workbench/repositories/NONE/repositories to access the OpenRDF Workbench Web application. As you can see in Fig 56, there is a repository by default.

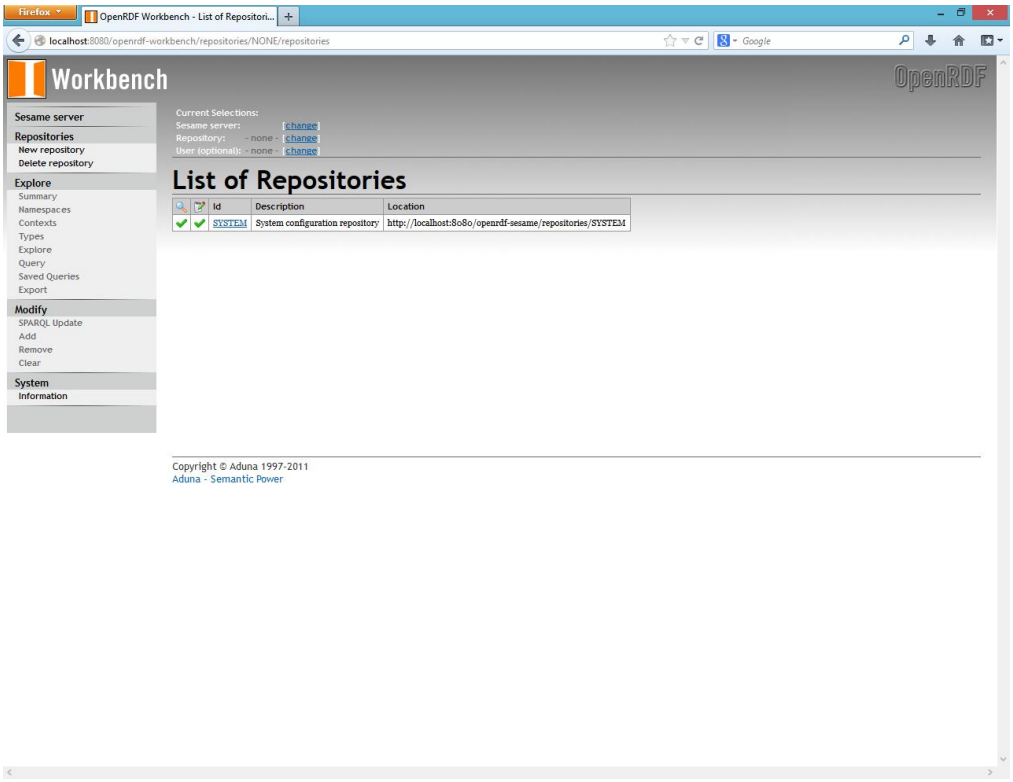


Fig 56 – *OpenRDF Workbench* overall interface

- Then, we can create a repository. Go to Repositories-New repository to create one, give the required information and press *Next* button.

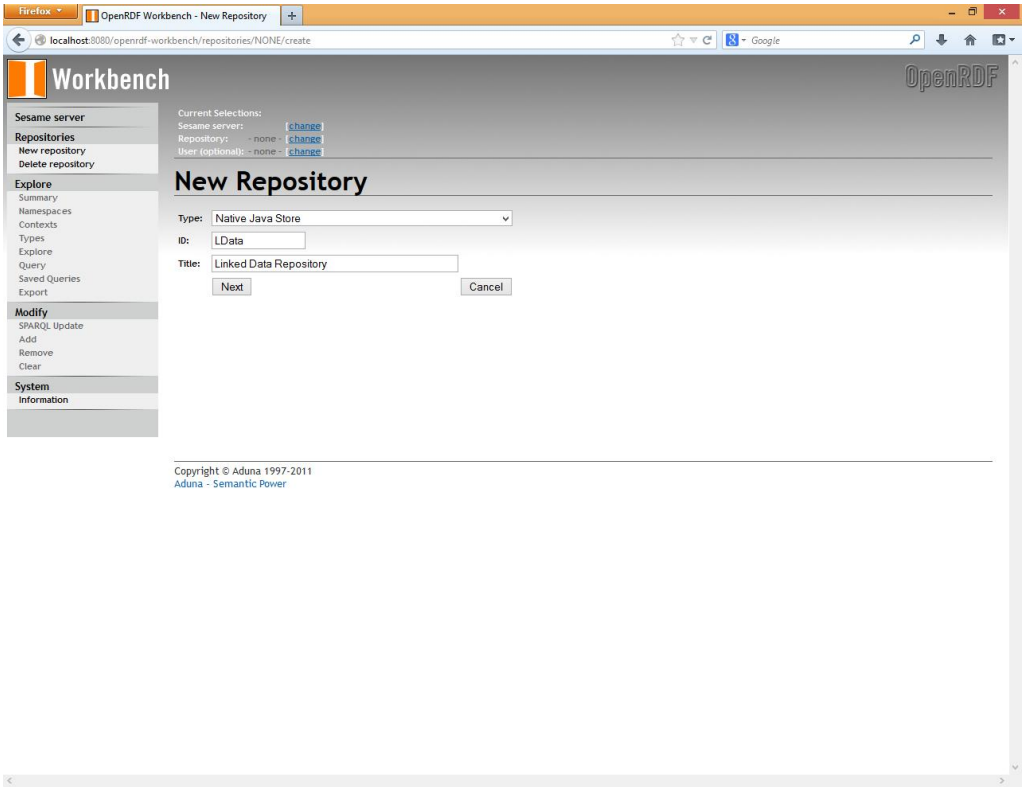


Fig 57 – Creating a new repository with *OpenRDF Workbench* (1)

Then, given the default items, press the *Create* button.

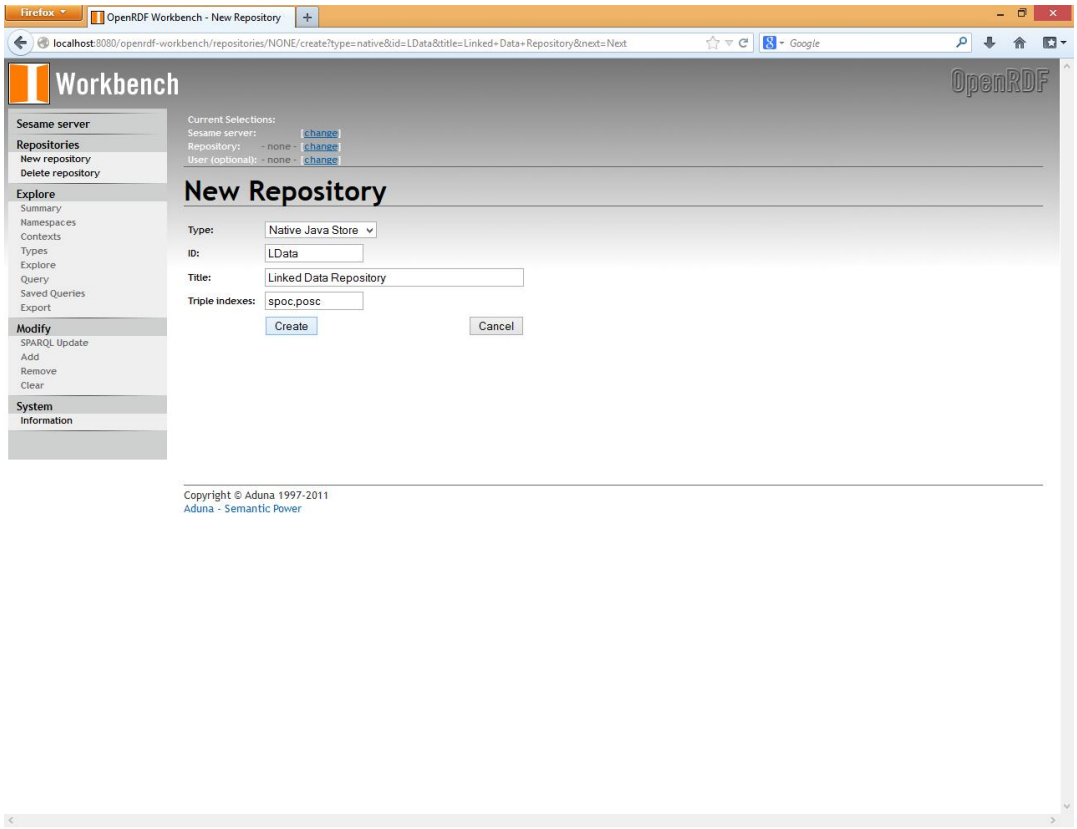


Fig 58 – Creating a new repository with *OpenRDF Workbench* (2)

Fig 59 shows the new repository that we have already created within OpenRDF Workbench.

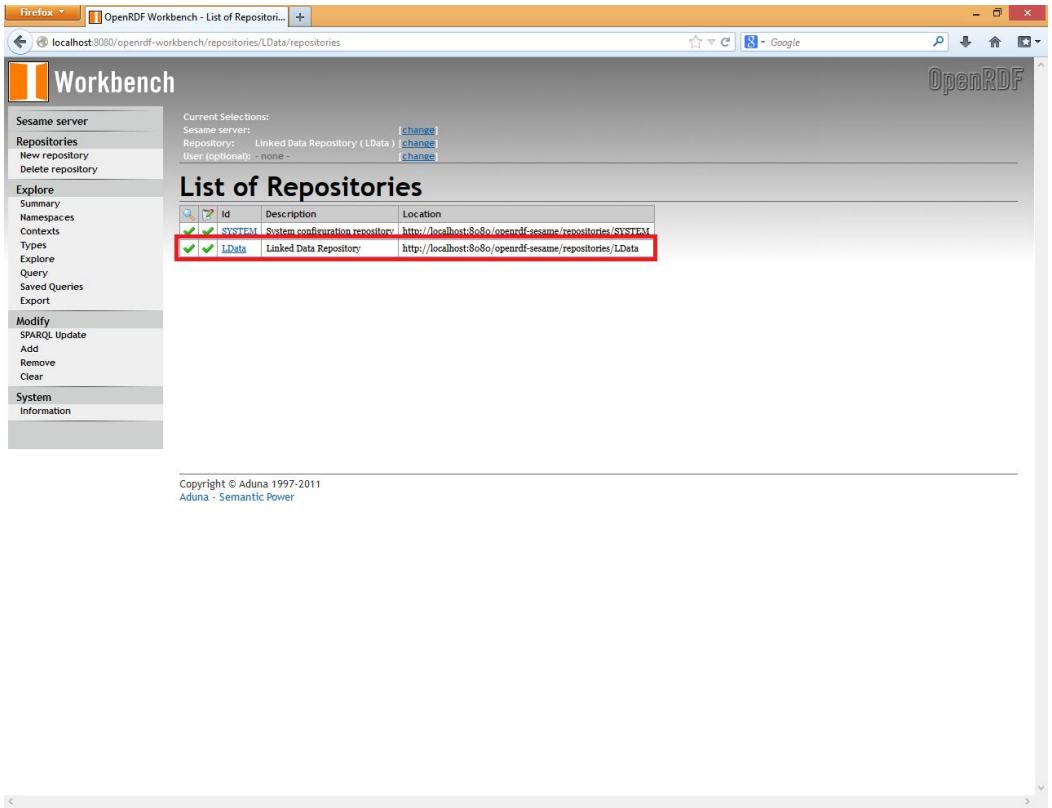


Fig 59 - Creating a new repository with *OpenRDF Workbench* (3)

- We may add RDF data into our repository. Go to Modify-Add, select the RDF file that we want to upload and click *Upload*.

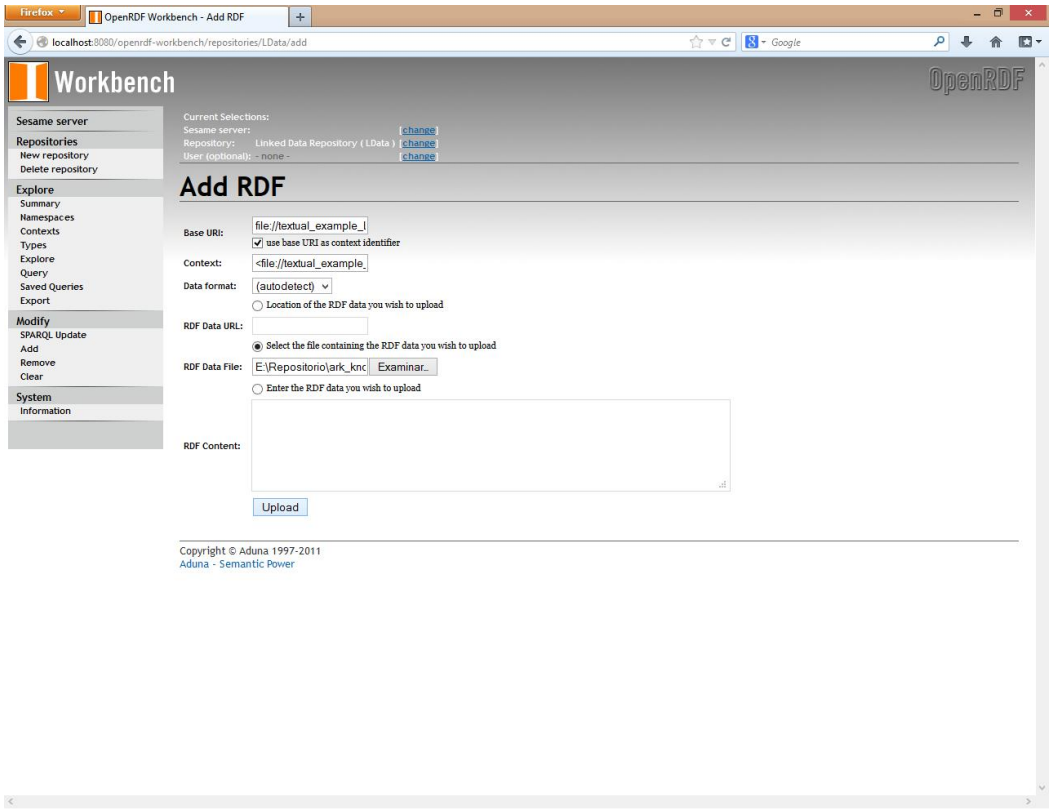


Fig 60 – Adding RDF data with *OpenRDF Workbench*

Then, we can go to Explore-Contexts and click on the RDF file that we have already uploaded in order to see its information.

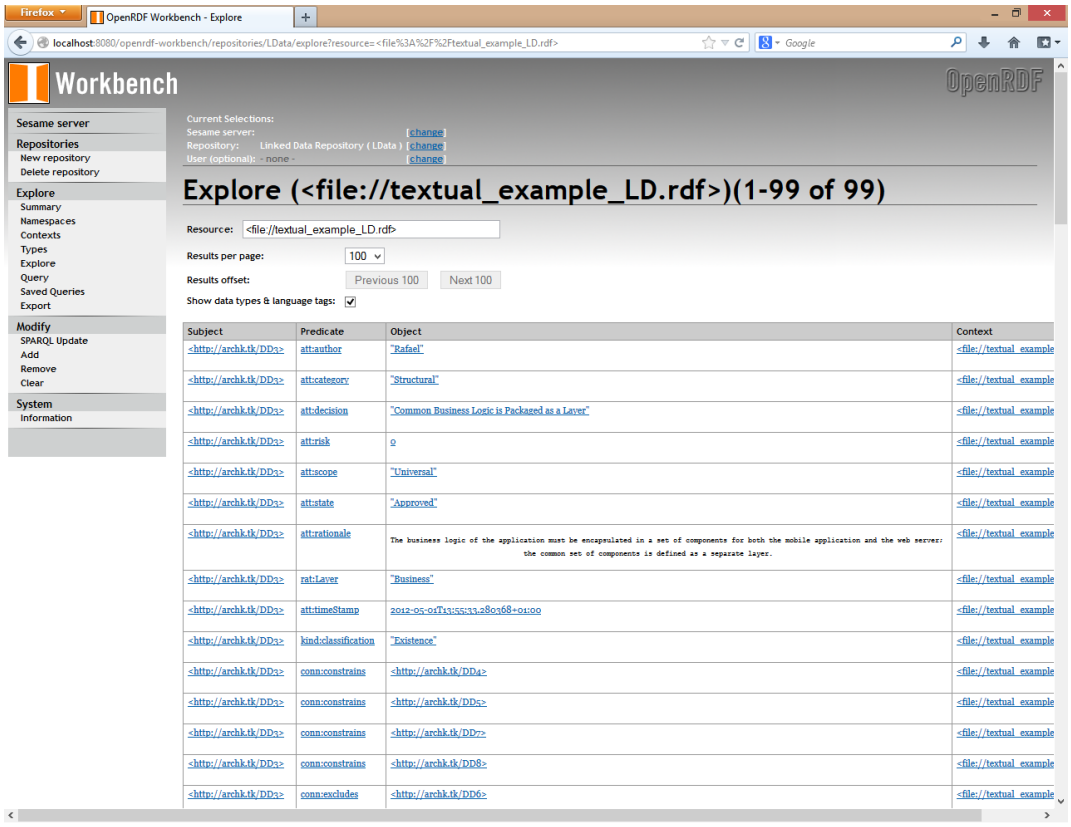


Fig 61 – Exploring RDF data with *OpenRDF Workbench*

- Finally, among other things, we can execute SPARQL queries going to Explore-Query and pressing the *Execute* button.

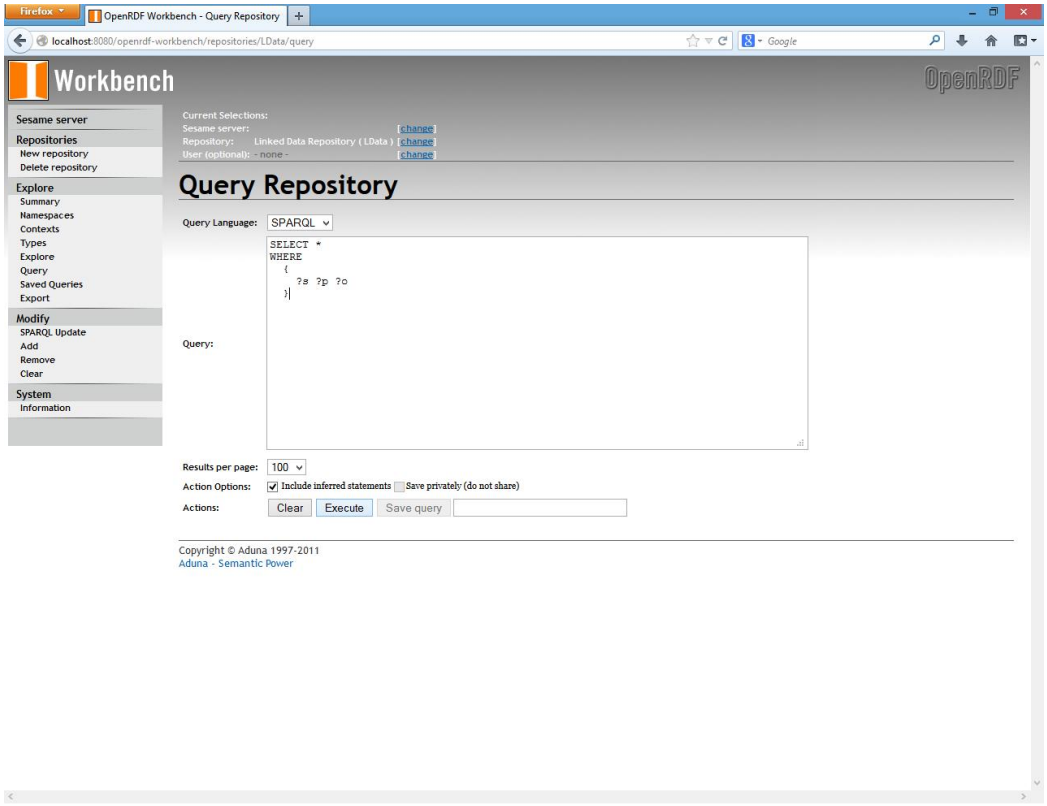


Fig 62 – Executing a SPARQL query with *OpenRDF Workbench*

These are the query results for the previous SPARQL query.

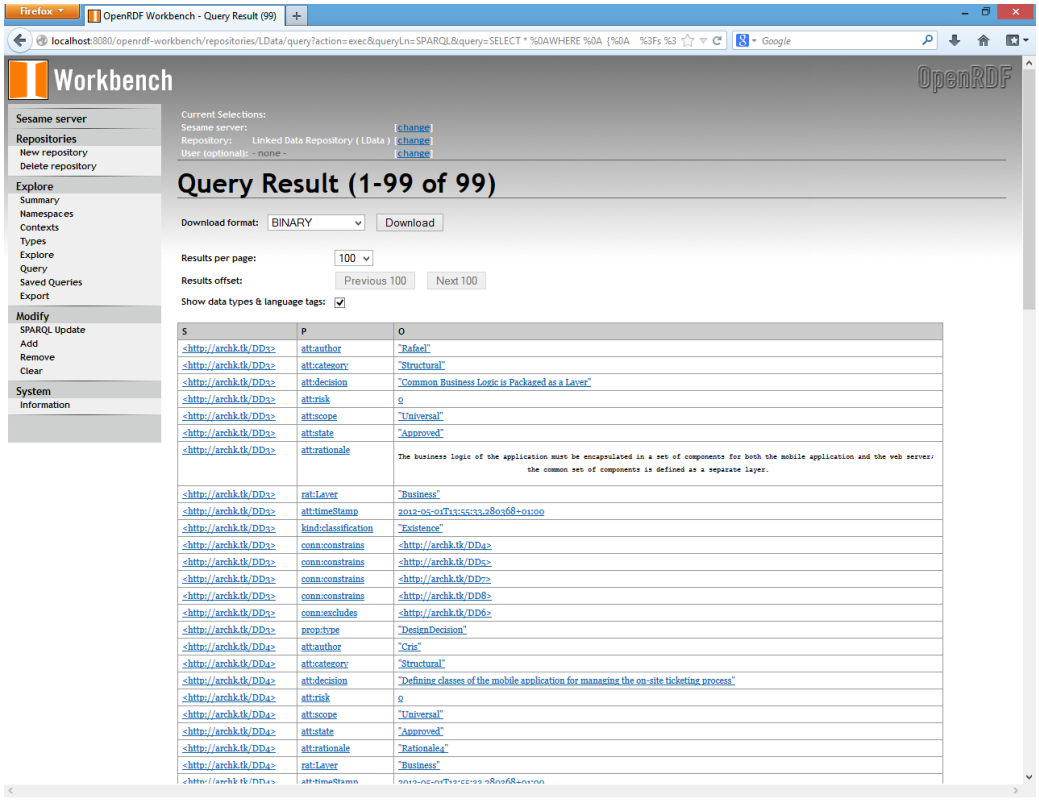


Fig 63 – SPARQL query results with *OpenRDF Workbench*

4.6 Mulgara

Mulgara [16][17] is a scalable open source RDF database written entirely in Java, licensed under the Open Software License 3.0. This tool is like a relational database due to you can store information and retrieve it via a query language. Unlike a relational database, Mulgara is optimized for the storage and retrieval of RDF statements (subject-predicate-object).

These are the general features of this semantic store:

- Native RDF support
- Multiple databases (models) per server
- Simple SQL-like query language (SPARQL or TQL[2])
- Small footprint
- Full text search functionality
- Large storage capacity
- Optimized for metadata storage and retrieval
- Multi-processor support
- Low memory requirements
- Streamed query results

Apart from that, Mulgara provides mechanisms for assuring *reliability* (full transaction support, clustering and store level fail-over, permanent integrity), *connectivity* (Jena, SOAP,

[2] Text Query Language

Software Developers Kit, etc.), *manageability* (near zero administration, web based configuration and monitoring tools) and *scalability* (XA Triplestore engine) of our system.

Thus XA Triplestore, which is the storage engine of Mulgara, provides scalability due to the following features:

- 64-bit data structures, that allow Mulgara to store very large amounts of data, up to the limits imposed by the host operating system
- Multiple sessions with no lock contention, so a single writing session in addition to multiple reading sessions can access the triplestore concurrently without the reading sessions being required to acquire a global lock while processing a query
- On-line backups, so you can modify and query concurrently with a backup operation
- Permanent integrity, thanks to on-disk data structures of the triplestore
- Use of Java NIO (new I/O), that provides transactions, permanent integrity and good performance while still remaining a pure Java implementation

Mulgara is capable of querying any type of data source by using *Resolvers* which accept and process queries against data contained in a file, a database or other data source. In most cases, the data being queried against is not in a meaningful format, i.e. it is not RDF, and so the resolver must convert the data first.

The resolver database class is highly configurable, allowing you to optimally set up Mulgara for the appropriate usage requirements. By inserting different classes into the constructor of the database class you can set it up as a:

- *Heavyweight store* that uses disk input and output as its primary storage, making it *persistent* across executions of the server
- *Lightweight, memory based store* that is faster but subject to memory limitations and *no persistence*

There are five configurable parts for the database with respect to its operation:

1. Persistent Node Pool
2. Persistent String Pool
3. Temporary Node Pool
4. Temporary String Pool
5. System Resolver Factory

The *Persistent String* and *Node Pools* maintain the mappings of node *id* to string representations for all current models in both the System and External models. The *Temporary String* and *Node Pools* are used for storing temporary nodes during a query.

4.6.1 Architectural Knowledge as Linked Data: step by step with Mulgara

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *Mulgara*.

1. First of all, go to http://www.mulgara.org/download.html and download the latest version of Mulgara. At the time of writing, the latest release is 2.1.13. Notice that we are going to download the ZIP file.

2. Then, unzip this file in order to start a Mulgara server. Go to a console terminal and type *java -jar mulgara-2.1.13.jar* to start a Mulgara server with default settings.

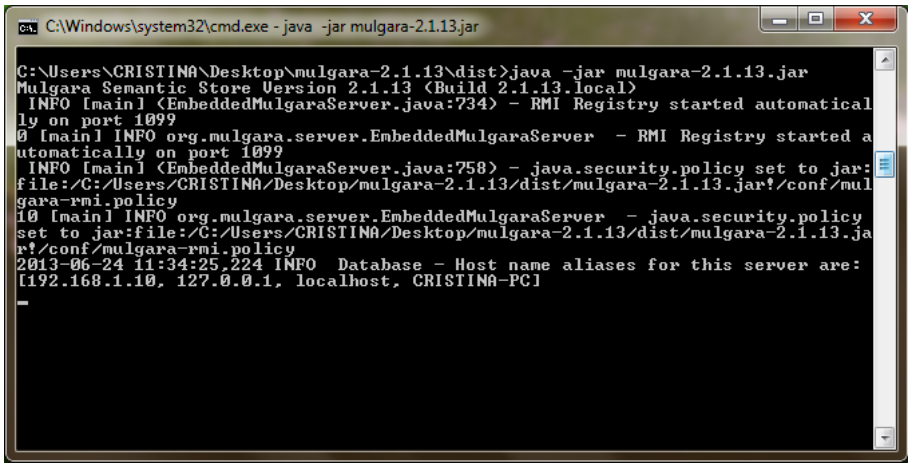


Fig 64 – Starting a Mulgara server

3. A default configuration for a standalone Mulgara server runs a set of web services, including the Web User Interface. So, we can go to http://localhost:8080/ and show the list of services available, as you see in Fig 65.

Fig 65 – *Mulgara* available web services

4. If we click on the *User Interface* link (http://localhost:8080/webui/), a page should appear as below:

Fig 66 – *Mulgara* User Interface

This Web UI page is used for general access to a Mulgara database.

5. Then, we can upload a RDF file in order to manage it later. First, we have to specify a graph URI, i.e. any valid absolute URI, for example: *urn:mydata*. Then, press the *Examinar...* button, choose the RDF file and click *Upload*.

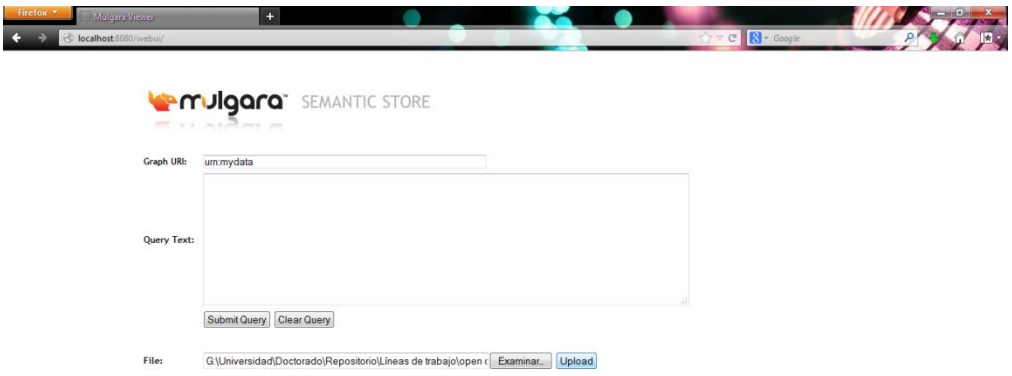


Fig 67 – Uploading a RDF file into *Mulgara* server (1)

It should appear a success message like this:

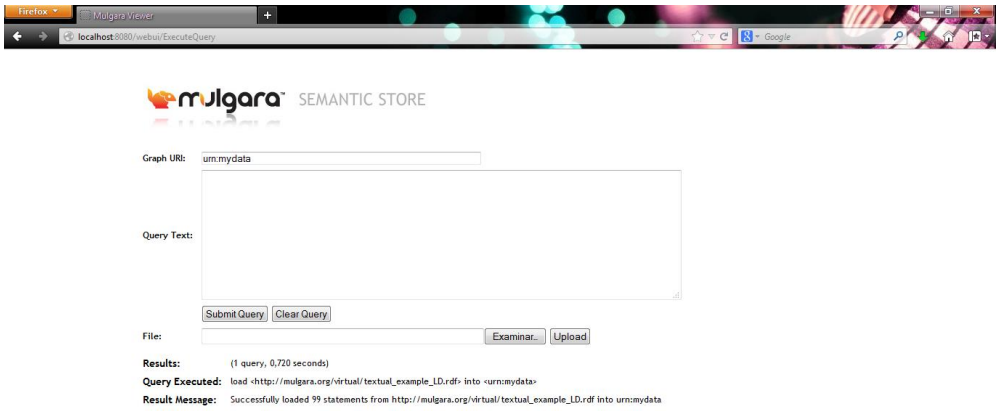


Fig 68 - Uploading a RDF file into *Mulgara* server (2)

6. Once we have uploaded our data, we can query it using, in this case, the SPARQL language. For example, we can write *SELECT * WHERE {?s ?p ?o}* in the *Query Text* area, which contains a single command or query in either TQL or SPARQL languages, and click on *Submit Query* button. Fig 69 shows the results for this SPARQL query.

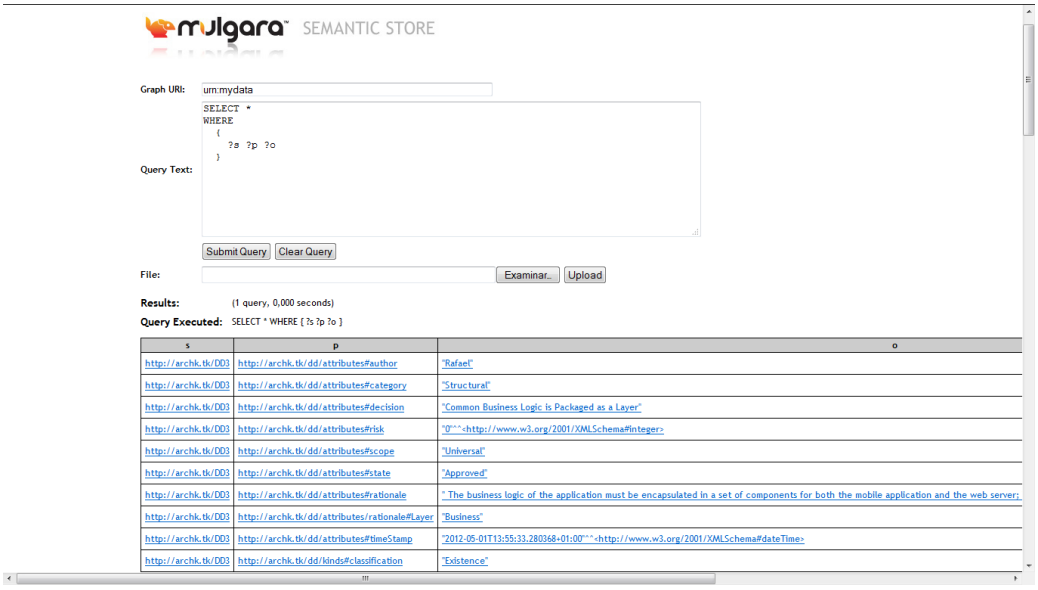


Fig 69 – Querying RDF data with *Mulgara*

4.7 RedStore

RedStore [18] is a lightweight RDF triplestore written in C language, using the Redland library that is a set of free software libraries that provide support for RDF. It supports, in addition to its native persistent or in-memory storage, a variety of storage backend adapters, including MySQL, Postgres, and Virtuoso. In native mode, RedStore uses *hash tables* for persisting RDF data.

This tool has some main features, cited as below:

- SPARQL over HTTP support
- Built-in HTTP server

- Support for a wide range of RDF formats
- Only one runtime dependency: Redland
- Unit and integration test suite

But it has some limitations too:

- Single process/single threaded
- No request timeouts

There are some available *storage modules*[3] for RedStore that you can explicit while starting it, using the *-t options* directive:

- *Hashes* that provides in-memory or persistent storage via Sleepycat/Berkeley DB (BDB) -> *-t hash-type='bdb'*
- *Memory* that provides a simple and fast in-memory store with no persistence
- *File* that provides an in-memory model initialised from the RDF/XML content in a file
- *MySQL* that is compiled in when MySQL 3 or 4 is available and provides persistent storage using the MySQL open source database
- *PostgreSQL* that is based on MySQL store and is compiled in when PostgreSQL is available. This store provides persistent storage using the PostgreSQL open source database
- *SQLite* that provides persistent storage via the SQLite relational database when available and supports SQLite V2 and V3
- *Tstore* that provides persistent storage via the AKT Triplestore when available
- *URI* that provides an in-memory model initialised from the RDF/XML content in a URI
- *Virtuoso* that is capable of efficiently handle large amounts of data with persistence

So, as an example, if you want to start RedStore on port 8080 using a persistent hashes store, you have to type this command: *redstore -p 8080 -s hashes -t "hash-type='bdb'"*. Notice that you have to install a Berkeley DB in order to use the hash-type *bdb* (go to http://linux.softpedia.com/progDownload/Berkeley-DB-Download-75.html).

4.7.1 Architectural Knowledge as Linked Data: step by step with RedStore

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *RedStore*.

1. First of all, notice that RedStore is not available for Windows environments, but for Mac OS X and Linux. In this case, I'm going to use a Linux environment like Ubuntu in order to try out RedStore. If we don't have a machine with Ubuntu as the Operating System, we can download *VirtualBox* from https://www.virtualbox.org/wiki/Downloads for emulating a Linux Operating System.

Notice that we can use, for example, *Cygwin* for simulating a Linux bash terminal, instead of a virtual machine environment, but it is not recommended because, in this case, Cygwin doesn't work properly during the RedStore installation.

[3] http://librdf.org/docs/api/redland-storage-modules.html

2. Then, go to http://releases.ubuntu.com/ in order to download an Ubuntu disk image. At the time of writing, the latest release is Ubuntu 13.04. This image has to be set within VirtualBox, so we can follow this tutorial http://www.psychocats.net/ubuntu/virtualbox for install it.
3. Once we have installed Ubuntu inside VirtualBox, start this virtual machine and open a terminal. Then, we are going to follow this tutorial in order to install RedStore http://wiki.filteredpush.org/wiki/RedStore_Installation_on_firuta.
4. Within the Linux terminal, write the following lines in order to download RedStore and its dependencies into a *tmp* folder:

```
cd Desktop
mkdir tmp
cd tmp
wget http://download.librdf.org/source/raptor2-2.0.6.tar.gz
wget http://download.librdf.org/source/rasqal-0.9.28.tar.gz
wget http://download.librdf.org/source/redland-1.0.15.tar.gz
wget http://www.aelius.com/njh/redstore/redstore-0.5.4.tar.gz
```

   Then, unpack these archives:

```
tar -zxvf raptor2-2.0.6.tar.gz
tar -zxvf rasqal-0.9.28.tar.gz
tar -zxvf redland-1.0.15.tar.gz
tar -zxvf redstore-0.5.4.tar.gz
```

5. Next, we have to install the dependencies for the Autogen script:

```
sudo su
apt-get install automake
apt-get install libtool
apt-get install gtk-doc-tools
```

6. Now, within the Raptor directory, run the Autogen script, install Libxml, which is the XML parser to be used by Raptor, compile and install it:

```
cd raptor2-2.0.6
./autogen.sh
apt-get install libxml2-dev
./configure
make
make install
```

7. Next, we must compile Rasqal, but first we need the latest version of Flex and Bison:

```
apt-get install flex
apt-get install bison
```

8. Now we can run the Autogen script followed by configure, make and install for Rasqal:

```
cd ..
cd rasqal-0.9.28
./autogen.sh
./configure
make
make install
```

9. Now for Redland:

```
cd ..
cd redland-1.0.15
./autogen.sh
./configure
make
make install
```

10. Finally, we can install RedStore given that its dependencies are taken care of:

```
cd redstore-0.5.4
./configure
make
make install
ldconfig
```

11. Once we have installed RedStore, we can start it on port 8086:

    ```
    redstore -p 8086 -b localhost
    ```

    If we go to http://localhost:8086/, we can see the main page of RedStore as in Fig 70.

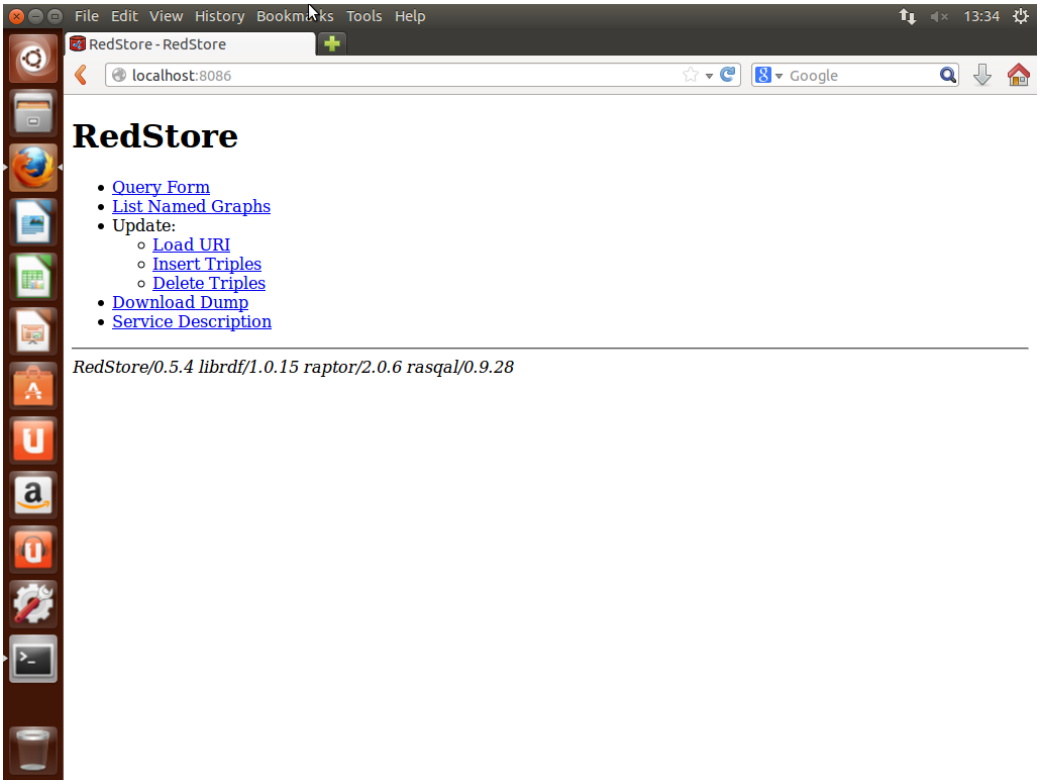


Fig 70 – Main page of RedStore

12. Then, if we go to *Insert Triples*, we can update our dataspace with our RDF data, selecting RDF/XML as the triple syntax, indicating a base URI and pressing the *Submit Query* button, as you see in Fig 71 and Fig 72.

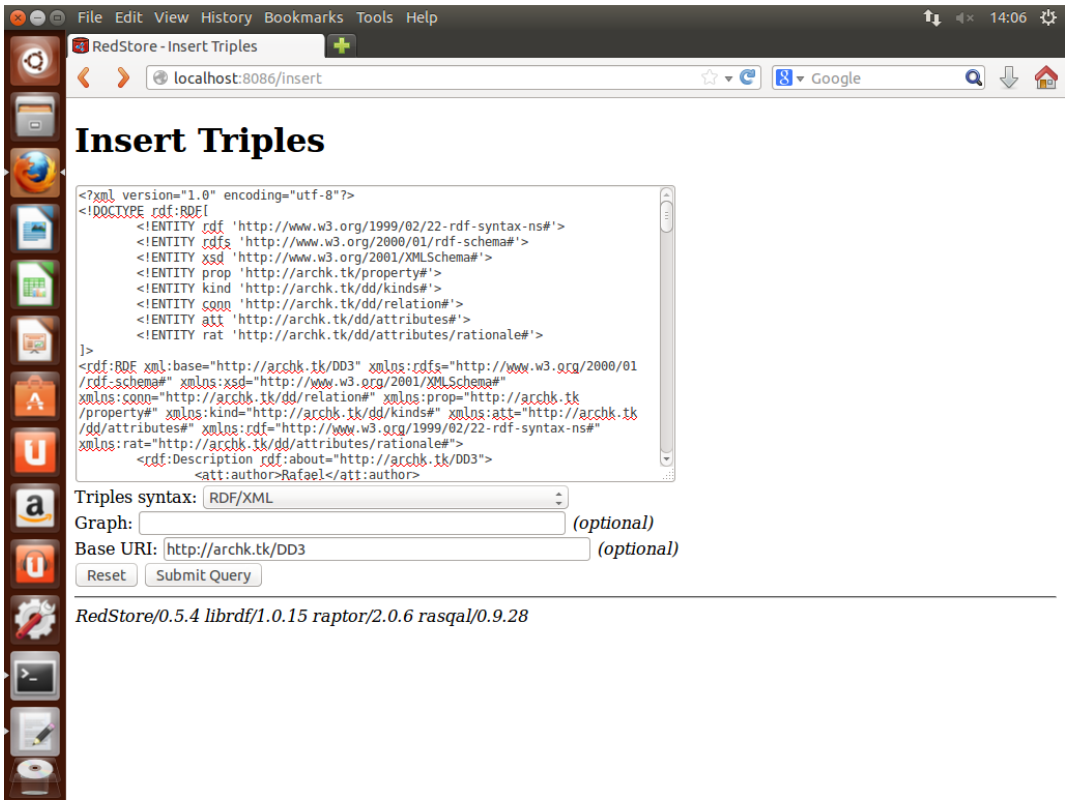


Fig 71 – Inserting triples with RedStore (I)

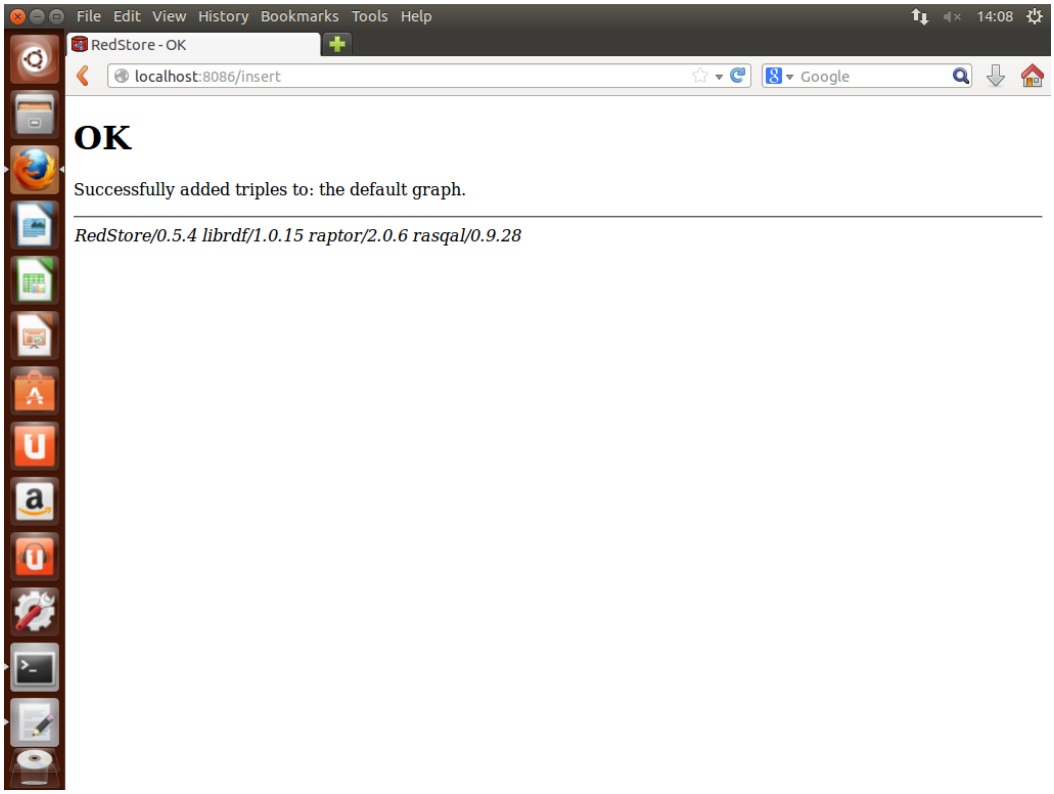
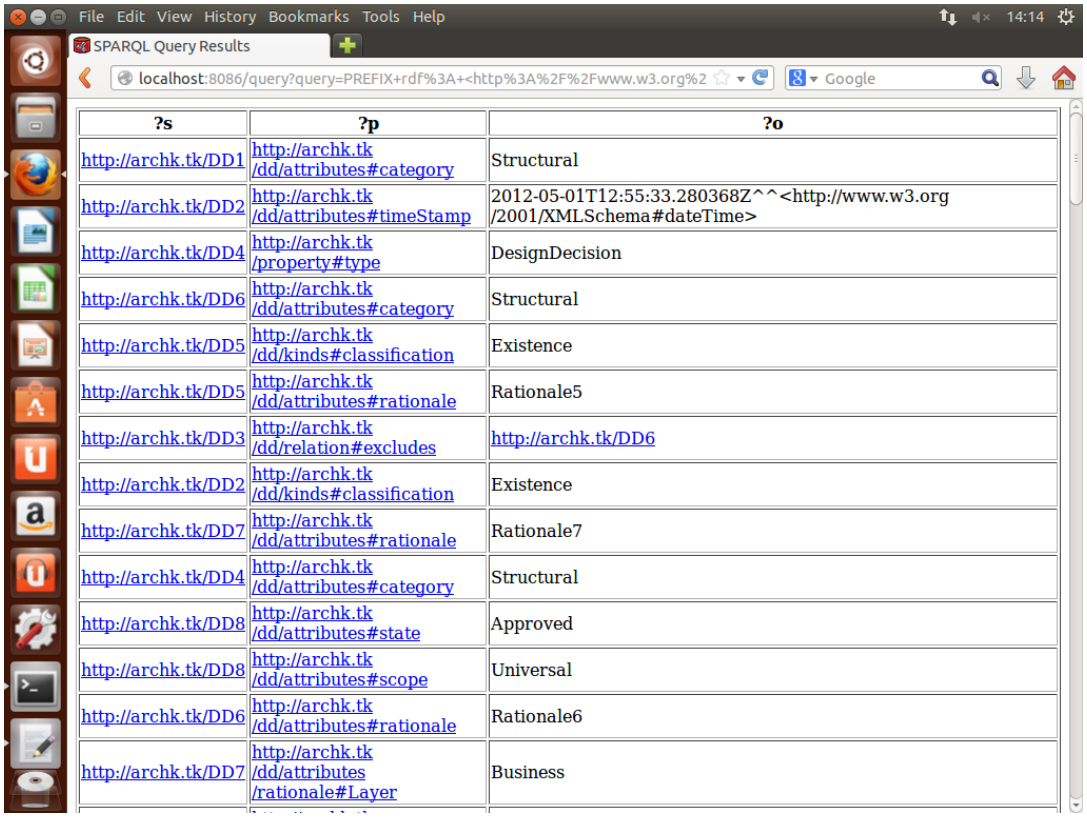


Fig 72 – Inserting triples with RedStore (II)

13. Finally, if we go back to the main menu, we can choose the *Query Form* option and make a SPARQL query, as you see in Fig 73 and Fig 74.

Fig 73 – Making a SPARQL query within RedStore (I)



?s	?p	?o
http://archk.tk/DD1	http://archk.tk /dd/attributes#category	Structural
http://archk.tk/DD2	http://archk.tk /dd/attributes#timeStamp	2012-05-01T12:55:33.280368Z^^<http://www.w3.org /2001/XMLSchema#dateTime>
http://archk.tk/DD4	http://archk.tk /property#type	DesignDecision
http://archk.tk/DD6	http://archk.tk /dd/attributes#category	Structural
http://archk.tk/DD5	http://archk.tk /dd/kinds#classification	Existence
http://archk.tk/DD5	http://archk.tk /dd/attributes#rationale	Rationale5
http://archk.tk/DD3	http://archk.tk /dd/relation#excludes	http://archk.tk/DD6
http://archk.tk/DD2	http://archk.tk /dd/kinds#classification	Existence
http://archk.tk/DD7	http://archk.tk /dd/attributes#rationale	Rationale7
http://archk.tk/DD4	http://archk.tk /dd/attributes#category	Structural
http://archk.tk/DD8	http://archk.tk /dd/attributes#state	Approved
http://archk.tk/DD8	http://archk.tk /dd/attributes#scope	Universal
http://archk.tk/DD6	http://archk.tk /dd/attributes#rationale	Rationale6
http://archk.tk/DD7	http://archk.tk /dd/attributes /rationale#Layer	Business

Fig 74 - Making a SPARQL query within RedStore (II)

4.8 Callimachus

Callimachus [19] is a framework for data-driven applications based on Linked Data. It allows web developers to quickly and easily create web applications based on Linked Data given that they only need a web browser to create a data-driven application.

In addition, Callimachus builds on either Sesame (section 4.5) or Mulgara (section 4.6) for RDF storage, AliBaba (a RESTful object-RDF library) and uses a revolutionary template-by-example technique for viewing and editing resources. One of the interesting aspects of Callimachus is that templates are parsed to build SPARQL from RDFa markup and then filled with query results.

4.8.1 Architectural Knowledge as Linked Data: step by step with Callimachus

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *Callimachus*.

1. First of all, we have to make sure that it is installed a JDK and JRE in our machine, and that the version is correct. In this case, we need JDK 1.6 or 1.7. We can type java –version in the command prompt in order to see what version is locally installed. Go to http://www.oracle.com/technetwork/es/java/javase/downloads/index.html for downloading them.
2. Once we have installed JDK and JRE in our computer, we have to set the environment variables for JDK_HOME (pointing to our JDK directory), JAVA_HOME (pointing to JDK our directory) and JRE_HOME (pointing to our JRE directory).
3. Next step is to download Callimachus. Go to http://callimachusproject.org/get-started.xhtml?view and download the latest version which is nowadays 1.1.2. It is a ZIP file that you have to unzip into an empty target directory.
4. Then, copy the file *etc/callimachus-defaults.conf* to a new file called *etc/callimachus.conf*, and open it in order to make some necessary changes, like remove the '#' before the PORT and ORIGIN variables. We can modify the values of these variables for make them more appropriate to our system.
5. Create a file called *etc/mail.properties* to allow for mail functionality within Callimachus.
6. Then, from a console, initialize Callimachus typing *callimachus-setup.bat* within the *bin* folder. After a few seconds, a new window is opened in our browser, as you can see in Fig 75. Fill in the information required and press "Yes, sign me up!" button.

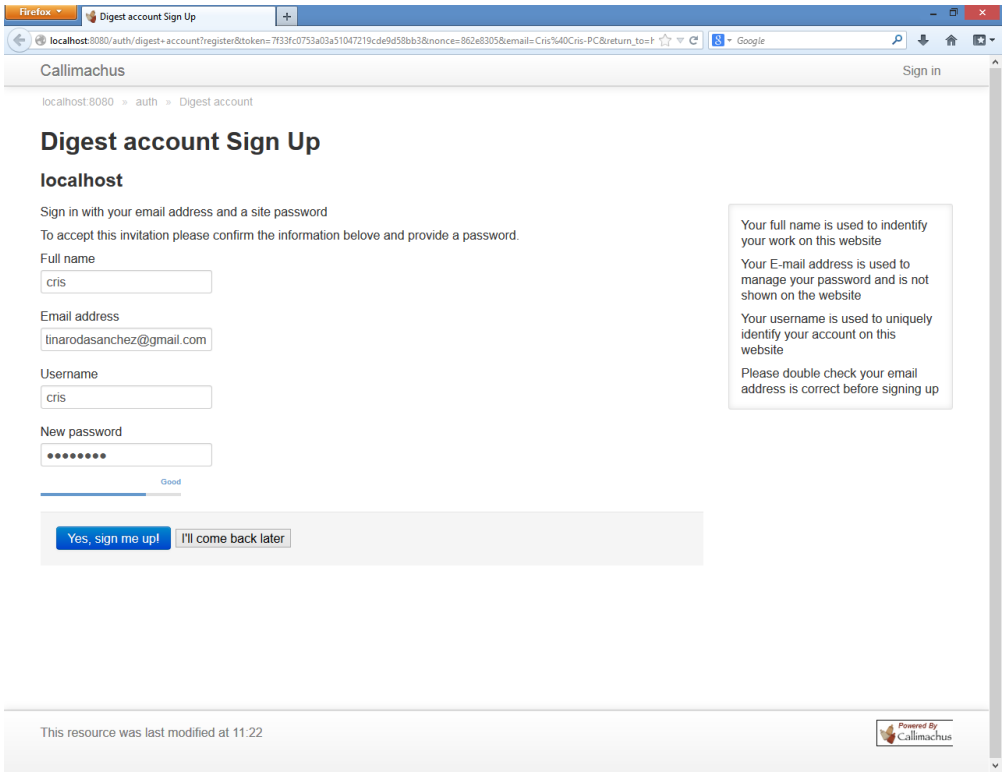


Fig 75 – Assigning a password to my Callimachus account

7. Then, a new sign-in window is showed. We have to write our credentials and press the "Sign in" button in order to access the Callimachus client application. Fig 77 shows the main interface, once we have signed in.

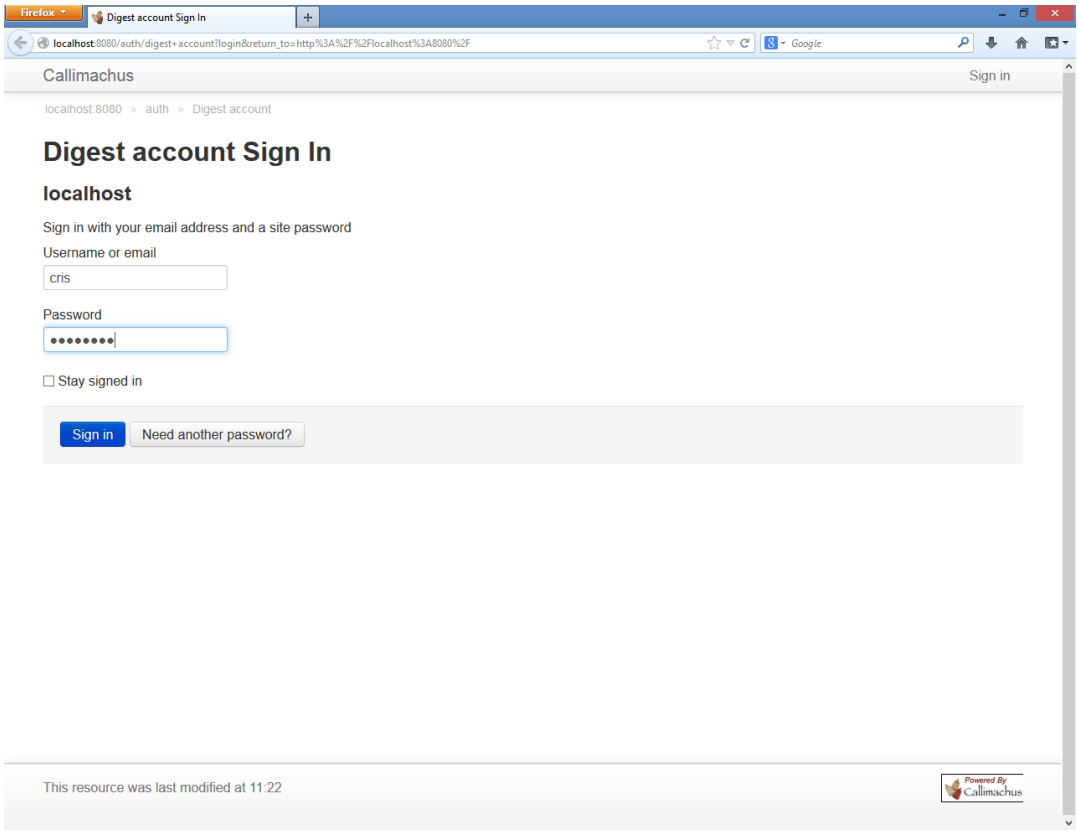


Fig 76 – Signing in Callimachus

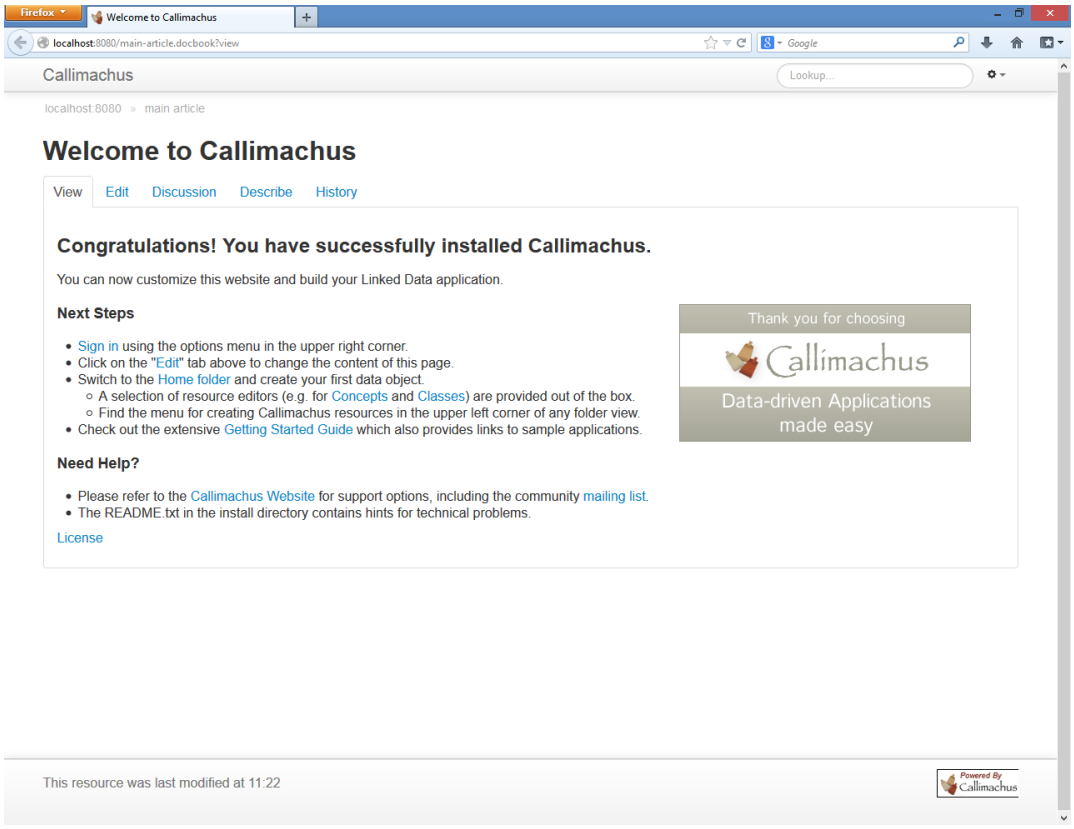


Fig 77 – Main interface in Callimachus

8. Now, we can import linked data into Callimachus. Navigate to the Home Folder link and create a new folder for our data, pressing the “Create” button.

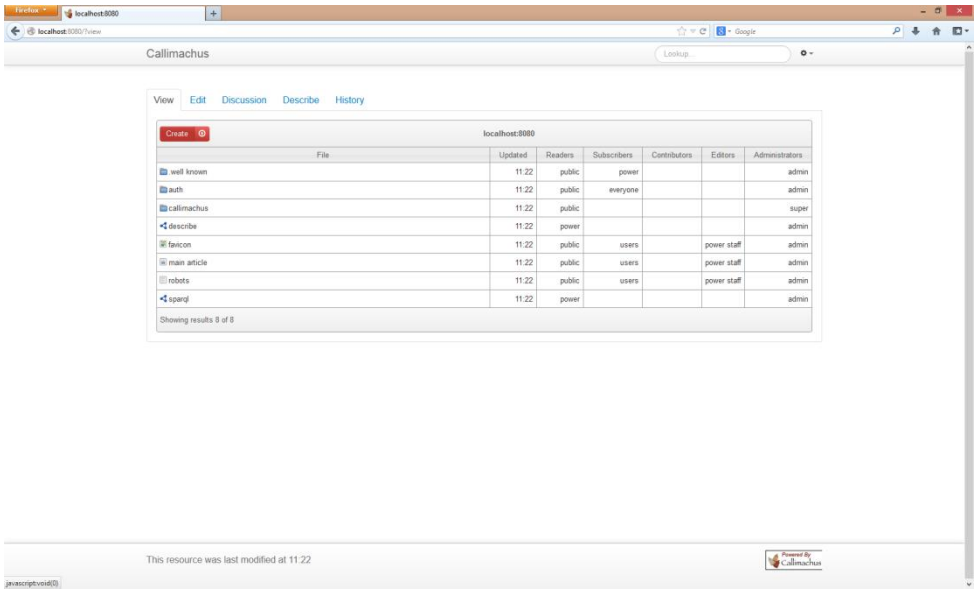


Fig 78 – Managing linked data into Callimachus (I)

Choose the *Folder* option.

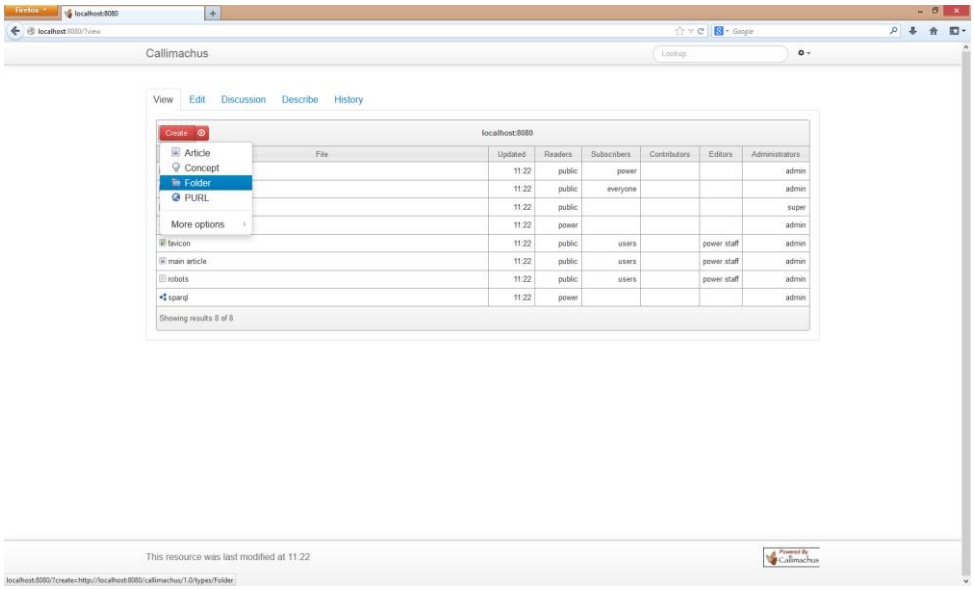


Fig 79 - Managing linked data into Callimachus (II)

Give a folder name and press the “Create” button.

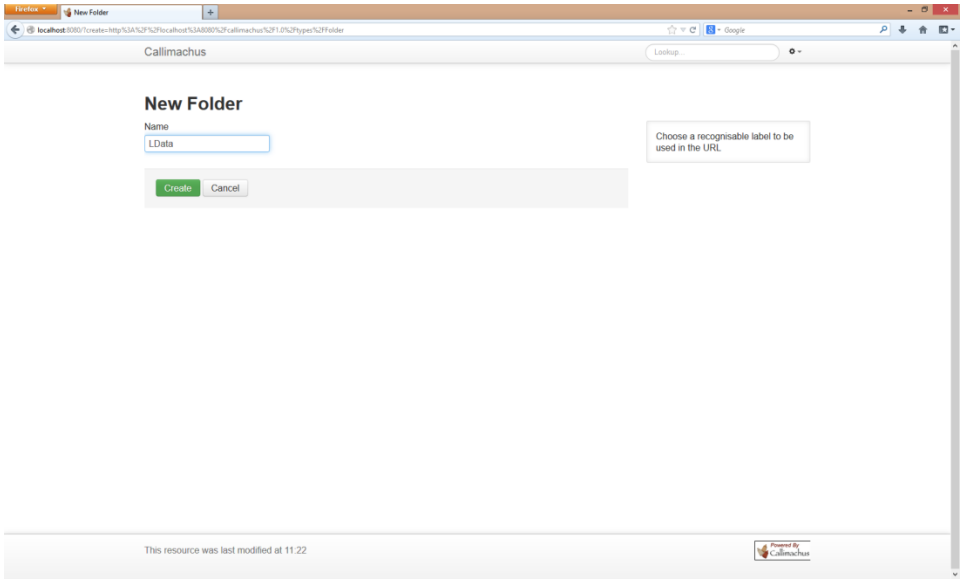


Fig 80 - Managing linked data into Callimachus (III)

Then, click on the *Upload* option in order to upload the linked data file.

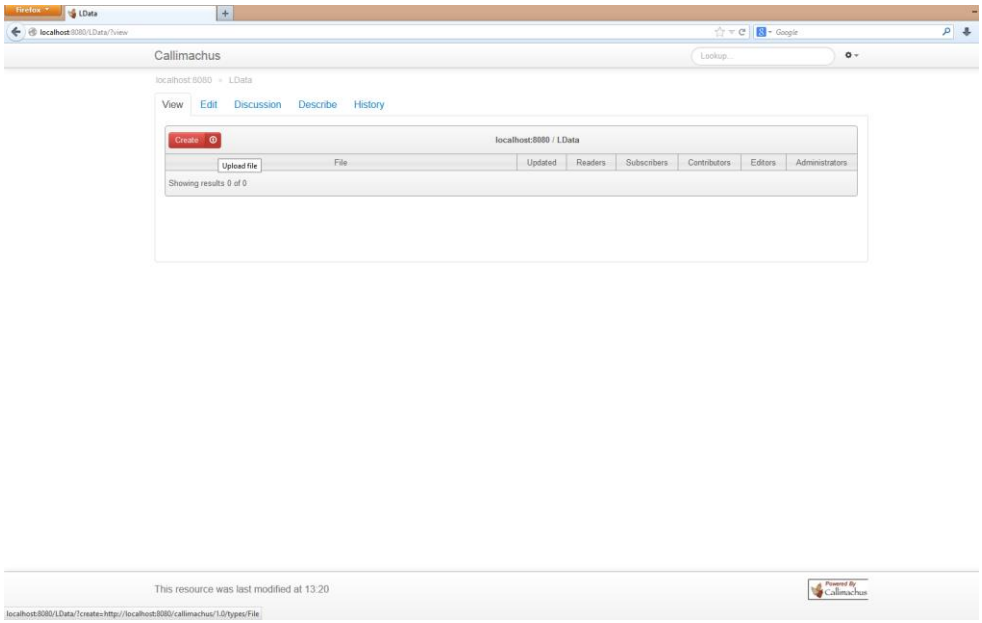


Fig 81 - Managing linked data into Callimachus (IV)

Then, select the corresponding RDF file and click “Upload”.

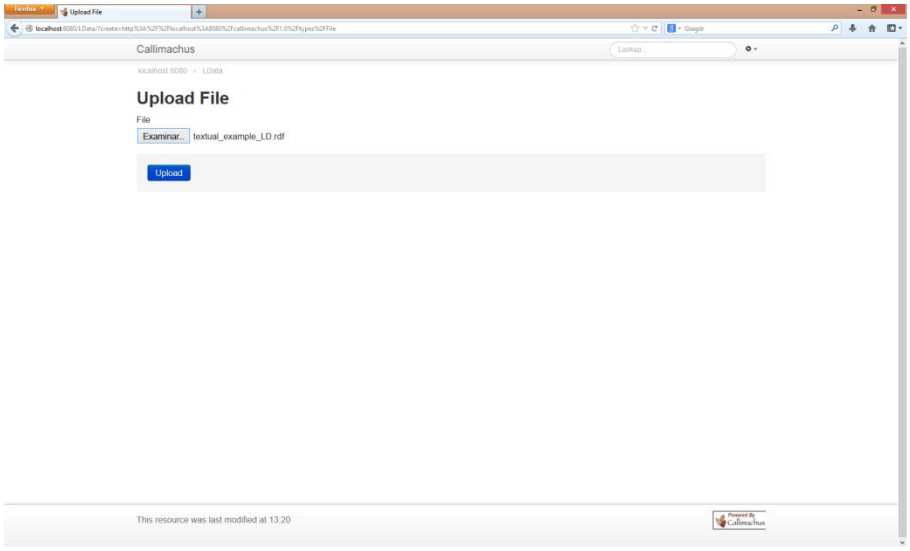


Fig 82 - Managing linked data into Callimachus (V)

Now, our RDF data is on http://localhost:8080/LData/textual_example_ld.rdf, as you can see in Fig 83.

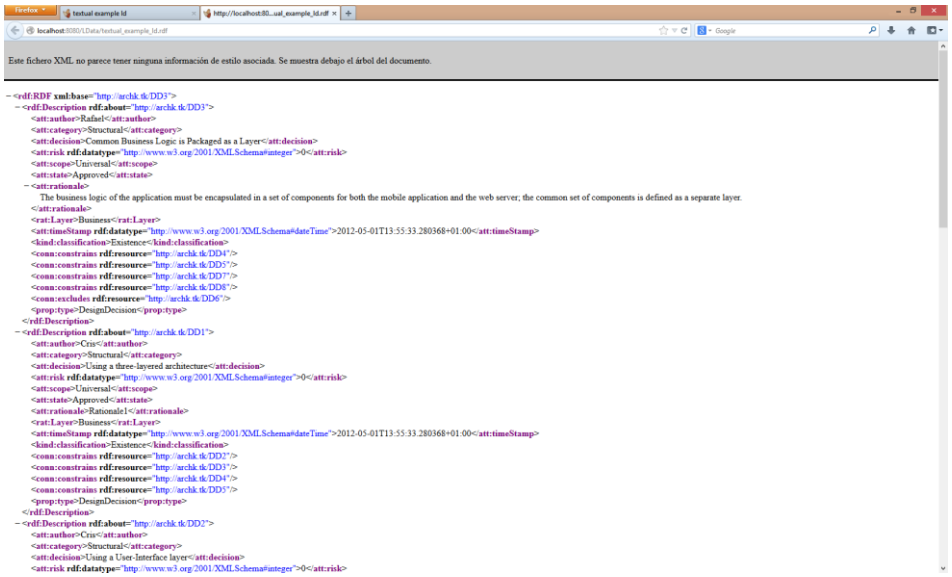


Fig 83 - Managing linked data into Callimachus (VI)

9. Finally, we can create a named query, i.e. a SPARQL query that you can manage within Callimachus (edit, delete, show it results…). Go to our LData folder and click on the "Query" option.

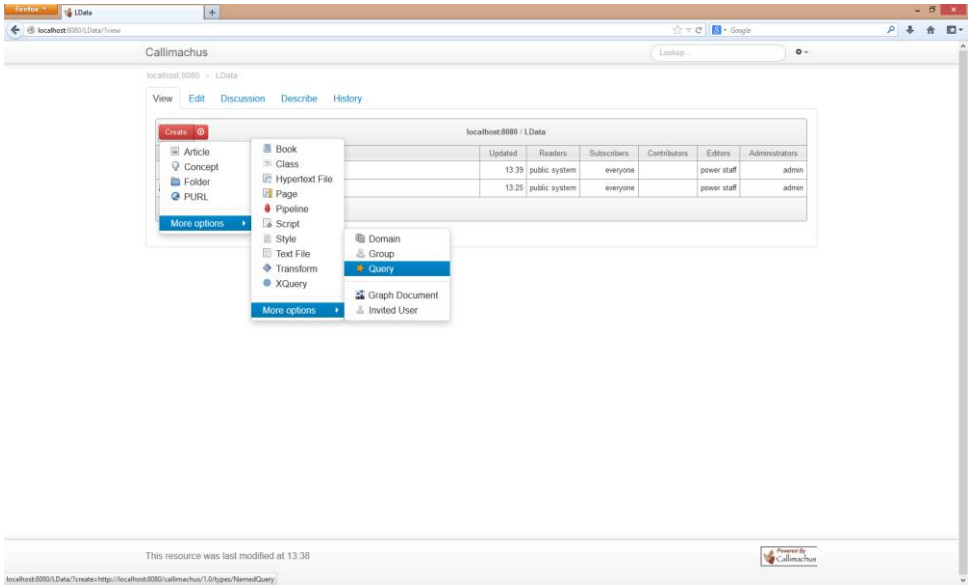


Fig 84 – Creating a SPARQL query with Callimachus (I)

Then, we can write our SPARQL query and save it within Callimachus, pressing the "Create" button at the bottom of the screen.

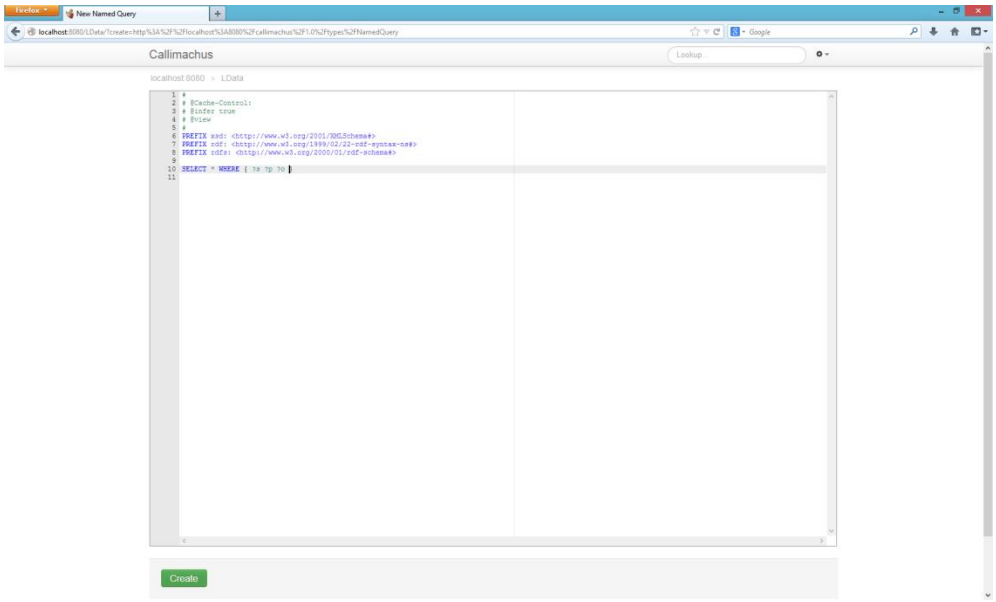


Fig 85 - Creating a SPARQL query with Callimachus (II)

Then, give a name to the concrete query and click on the “Save” button.

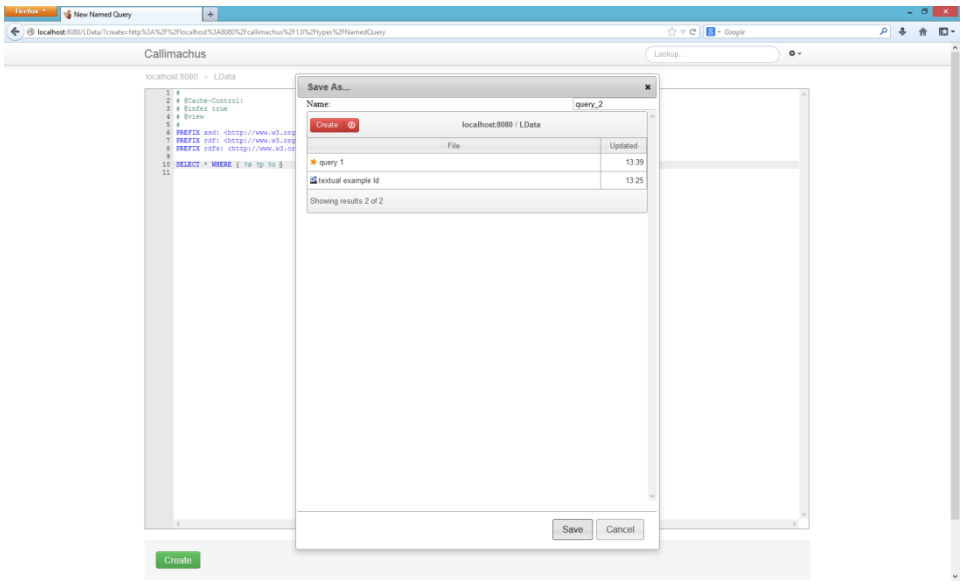


Fig 86 - Creating a SPARQL query with Callimachus (III)

Finally, we may see the query results.

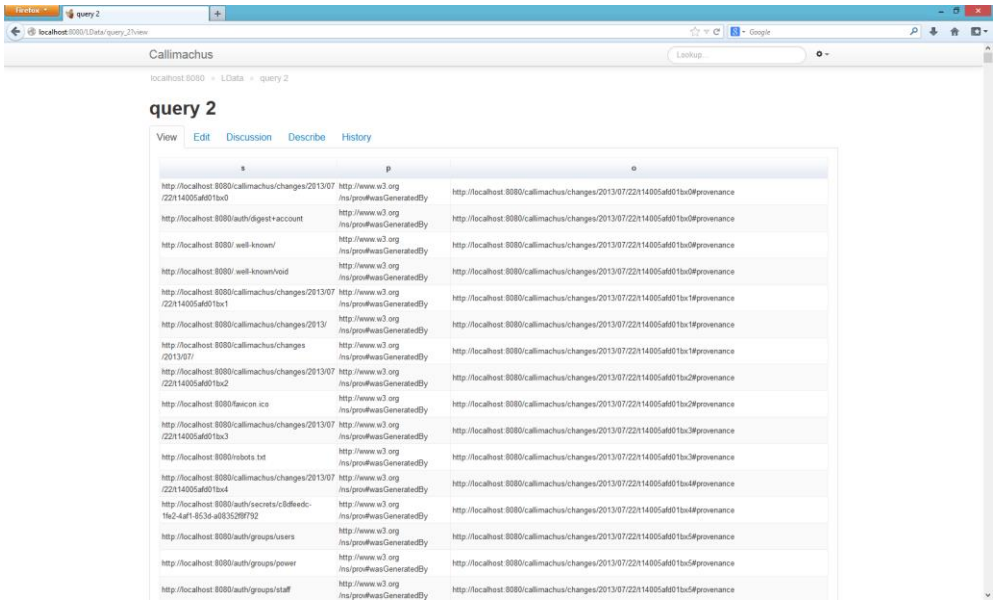


Fig 87 – SPARQL query results

4.9 Other tools

This section provides some information about several Linked Data tools that we couldn't test properly due to the fact of problems with installation, start-up or data management. Therefore, these applications are not going to take part in the feature analysis presented in section 5.

4.9.1 Longwell

Longwell [20] is a web-based RDF-powered highly-configurable faceted browser. This application is written as an open source Java web application designed to let you visualize, browse and search an RDF complex dataset, especially large ones.

Longwell mixes the flexibility of the RDF data model with the effectiveness of the faceted browsing UI paradigm, allowing you to build a user-friendly web site out of your data within minutes and without requiring any code at all.

This tool has been developed by *Massachusetts Institute of Technology* (MIT) through the Simile project.

4.9.1.1 Architectural Knowledge as Linked Data: step by step with Longwell

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *Longwell*.

1. First of all, we need some specific requirements in order to run Longwell:
   - A Java 1.5 or later compatible virtual machine for our operating system
   - An installation of Apache Maven 2.0 or later (http://maven.apache.org/download.cgi)
   - (Optional) A servlet container (such as Jetty or Apache Tomcat) or any J2EE compatible application server (such as Apache Geronimo, JBoss, WebSphere, Orion or Weblogic)
2. Go to http://simile.mit.edu/dist/longwell/ and download the latest release of Longwell. At the time of writing, the latest version is 2.5.5. In this case, we are going to download the

ZIP file instead of the TAR.GZ one. Then, unzip this file in order to start to use Longwell in our machine.

3. Then, open the terminal DOS Prompt on Windows. Go to the base directory of the Longwell distribution that we have already downloaded and type *mvn package* to download the dependencies and build the code, as you see in Fig 88.

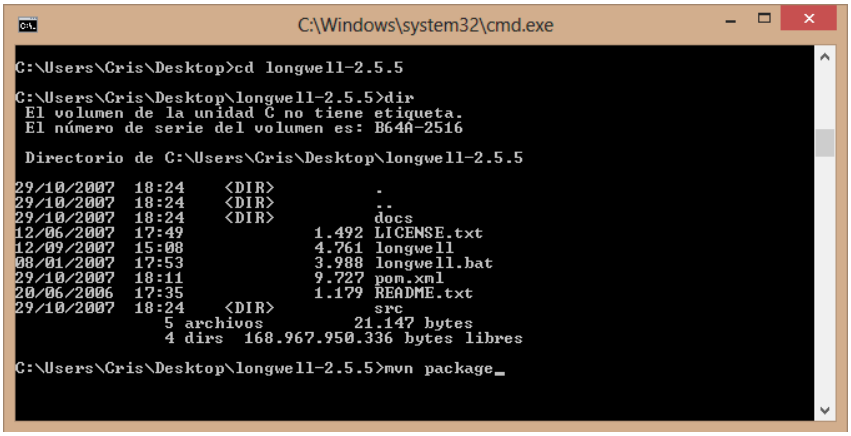


Fig 88 – Starting with *Longwell*

4. Type *./longwell -r path/to/RDF/data/* on Unix or MacOSX, or *longwell /r path\to\rdf\data* on Windows, where "path to RDF data" is the a location on your disk where the RDF data you want to load is currently stored.

NOTE: Longwell is capable of loading files with the extension RDF, RDFS, OWL, N3 and RSS. If the path is a directory, Longwell will recursively scan it for all files that look like RDF. If the file is GZIP compressed (e.g., ends in .rdf.gz), Longwell will automatically decompress it before reading it.

5. Once Longwell has stopped initializing, loading and processing the data (which might take from a few seconds to hours, depending on the size of your dataset), we can point our browser at http://127.0.0.1:8080/ and enjoy it.

Notice that we don't include this tool in our feature analysis given that it doesn't work properly, as you can see in Fig 89 when we execute step 3, so navigating to http://127.0.0.1:8080/ doesn't show anything. In addition, Longwell is a retired project, as we may see in http://simile.mit.edu/mail.html, so it doesn't be supported anymore.

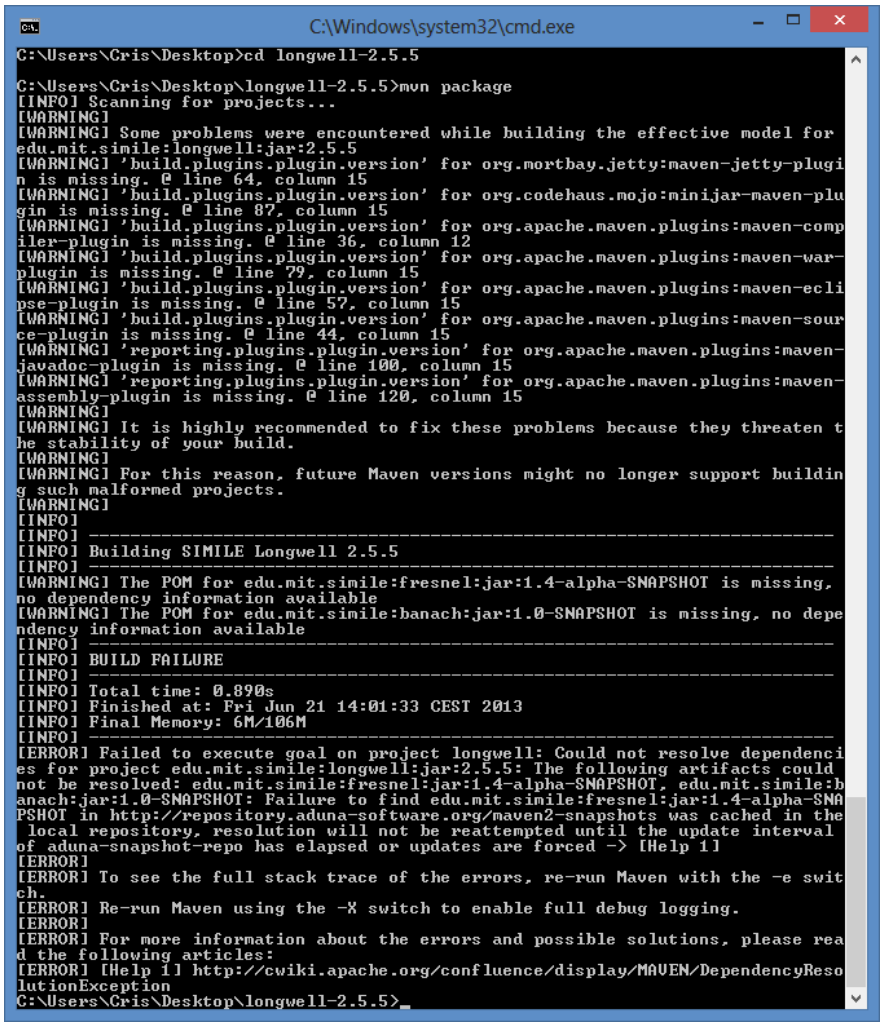


Fig 89 – *Longwell* error

4.9.2 Pubby

Pubby [21] can be used to add Linked Data interfaces to SPARQL endpoints. Therefore, Pubby makes it easy to turn a SPARQL endpoint into a Linked Data server. It is implemented as a Java web application.

Many triple stores and other SPARQL endpoints can be accessed only by SPARQL client applications that use the SPARQL protocol. It cannot be accessed by the growing variety of Linked Data clients. Pubby is designed to provide a Linked Data interface to those RDF data sources. Fig 90 shows its architecture.

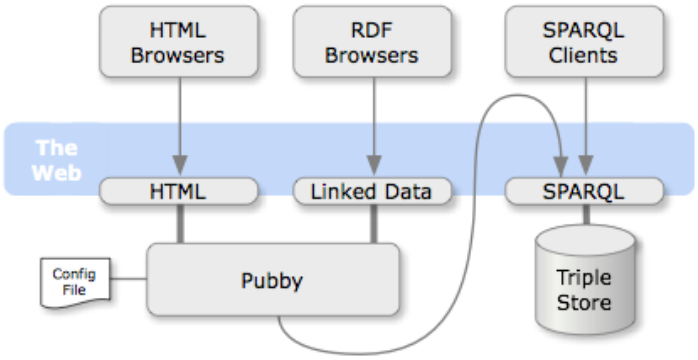


Fig 90 – Pubby architecture

When setting up a Pubby server for a SPARQL endpoint, you will configure a mapping that translates those URIs to dereferenceable URIs handled by Pubby. It will handle requests to the mapped URIs by connecting to the SPARQL endpoint, asking it for information about the original URI, and passing back the results to the client. It also handles various details of the HTTP interaction, such as the content negotiation between HTML, RDF/XML and Turtle descriptions of the same resource.

So, to sum up, its main features are:

- Linked Data interface to local or remote SPARQL protocol servers
- Dereferenceable URIs by rewriting URIs found in the SPARQL-exposed dataset into the Pubby server's namespace
- A simple HTML interface showing the data available about each resource
- Compatible with Tomcat and Jetty servlet containers
- A metadata extension to add metadata to the provided data

Pubby has some limitations too, such as:

- Only works for SPARQL endpoint that can answer DESCRIBE queries
- Multiple dataset support may not work as expected: If a requested URI is matched by the *conf:datasetURIPattern* of more than one dataset (or one doesn't have a *conf:datasetURIPattern*), then only one of the possible endpoints will be queried at a time. Pubby will never try to query multiple endpoints in order to create a single response. In most cases, it is recommended to simply set up a separate Pubby instance for each dataset.
- Hash URIs on the web side are not supported

4.9.2.1 Architectural Knowledge as Linked Data: step by step with Pubby

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *Pubby*.

1. First of all, go to http://wifo5-03.informatik.uni-mannheim.de/pubby/download/ and download the latest version of Pubby which is 0.3.3 at the time of writing.
2. If you haven't already, download and install a servlet container, like Tomcat or Jetty (see 4.5.1 to use Jetty).

3. Unzip the Pubby distribution and copy the *webapp* directory into the servlet container's *webapps* folder. If Pubby is the only web application you want to run in the container, then rename the *webapp* directory to *root*.
4. Before starting Pubby, we have to modify the configuration file to suit our needs. This file is located within Pubby's *webapp* directory, at */WEB-INF/config.ttl*. In our case, this is the final look of the configuration:

```
    # Example configuration which loads a static RDF file
    # and re-publishes it (textual_example_LD.rdf).
    # This configuration allows using Pubby as an RDF server for publishing
static RDF files.
    # Assumes Pubby is running at http://localhost:8080/

    @prefix conf: <http://richard.cyganiak.de/2007/pubby/config.rdf#> .

    <> a conf:Configuration;
        conf:projectName "Example";
        conf:webBase <http://localhost:8080/>;
        conf:projectHomepage <http://archk.tk/>;
        # When the homepage of the server is accessed, this resource will
        # be shown.
        conf:indexResource <http://archk.tk/DD1>;
        conf:dataset [
          conf:datasetBase <http://archk.tk/>;
          conf:addSameAsStatements "true";
          conf:loadRDF <textual_example_LD.rdf>;
        ];
        .
```

Notice that this configuration loads a static RDF file and re-publishes it, so it allows using Pubby as an RDF server for publishing RDF files. These files have to be at the same directory as our configuration file. In addition to this, if we change the name of the configuration file, we have to indicate it on the *web.xml* file, which is located at the same folder.

5. Now we may start Pubby, going to our Jetty directory and writing the following line in the console: *java –jar start.jar*. Then, if we go to http://localhost:8080 through our browser, we can see the initial page:

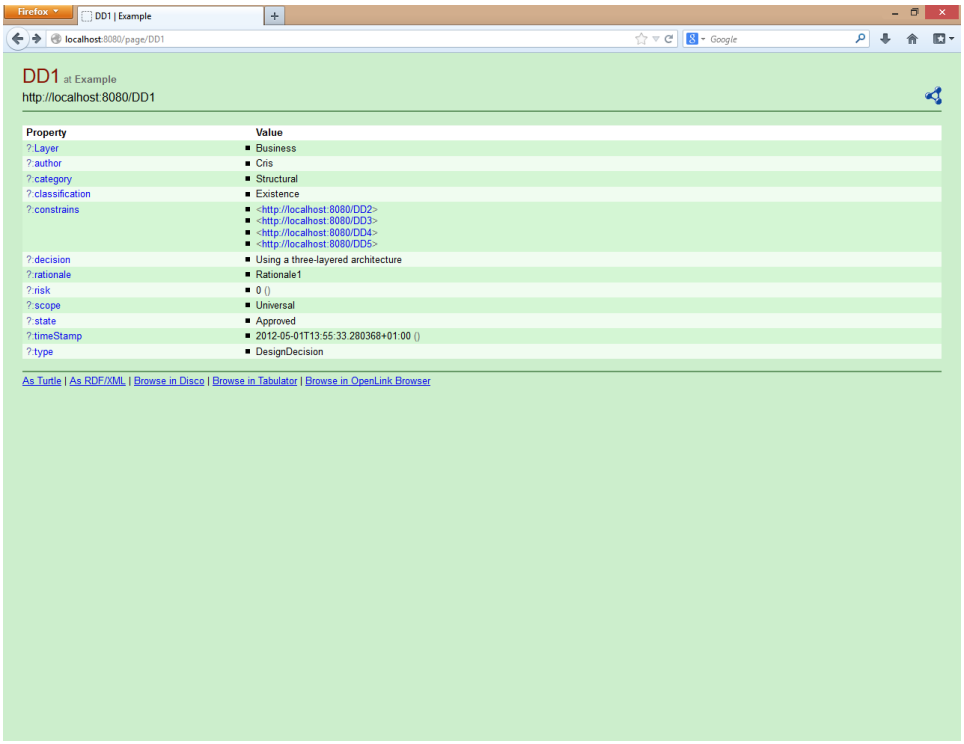


Fig 91 – Initial page in *Pubby*

Through this page, we can navigate between the different Design Decisions of our RDF data, given that they are written as links, for example by clicking on http://localhost:8080/DD3, information about design decision DD3 is displayed:

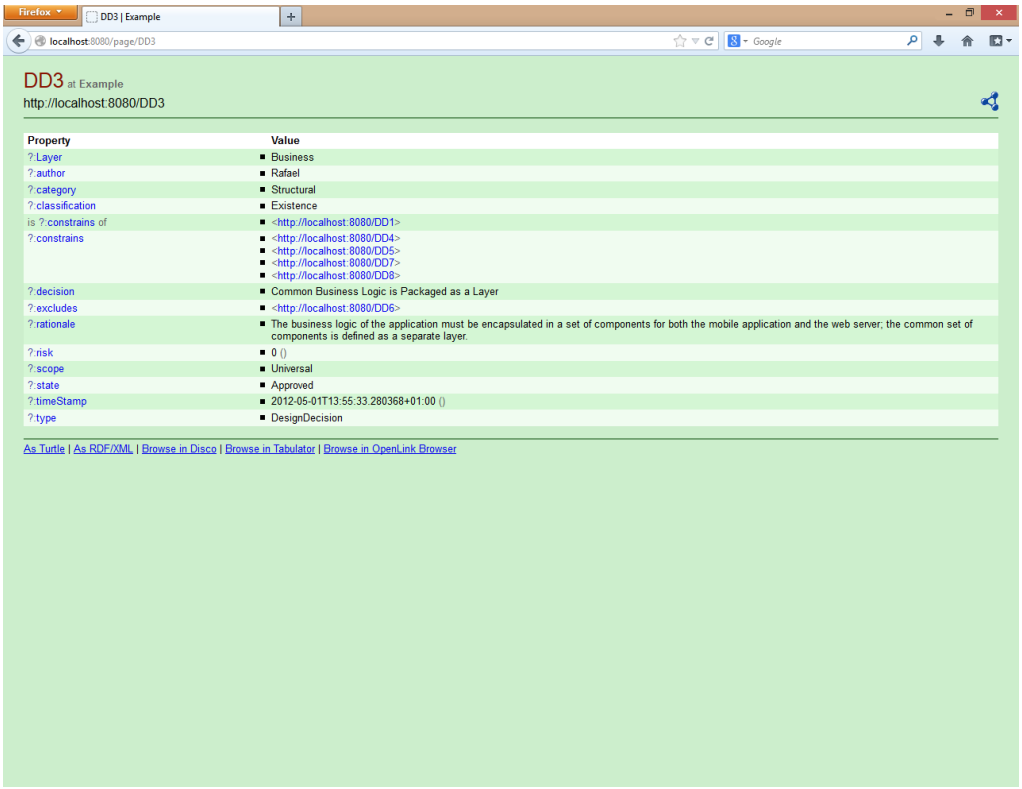


Fig 92 – Navigating through *Pubby*

Notice that we don't include this tool in our feature analysis due to the fact it doesn't give access and query to data. We can use query agents as SemWeb Client Library or SWIC to carry out SPARQL queries and access our data, thus Pubby itself doesn't satisfy our initial requirements.

4.9.3 AK-RDF

AK-RDF is a query system developed for managing and querying Architectural Knowledge networks. In this sense, this application is designed for documenting all design decisions of a particular Software Architecture, simulating the network created among them. Thus, we can access and execute SPARQL queries for recover information from this network in an easy, fast and efficient way. Notice that the Architectural Knowledge will be represented using RDF.

In addition, AK-RDF provides functionality to register user queries in order to access them at any time. As well it supports five possible types of servers in order to perform store and query tasks, but the server installation process is responsibility of the user, namely:

- Virtuoso (more information in section 4.1)
- AllegroGraph (not supported on Windows systems)
- Apache Jena (more information in section 4.3)
- 4Store (not supported on Windows systems)
- Sesame (more information in section 4.5)

So, the user has to install one (or more) of these servers, either locally or remotely (i.e. accessed only through its URI), in order to use the application. AK-RDF can be used in Windows and Linux environments.

This application provides the following functionality:

- Architectural Knowledge management
    - Create a new element: Requirement, Component or Design Decision
    - Edit an element
    - Delete an element
- Database query service
    - Execute customized query
    - Execute predetermined query
- Dataset management
    - Register a new dataset
    - Open a dataset
    - Delete a dataset
- Query management
    - Register a new query
    - Edit a query
    - Delete a query
- System settings

4.9.3.1 Architectural Knowledge as Linked Data: step by step with AK-RDF

In this section, we are going to describe in detail the steps you have to follow in order to manage Architectural Knowledge as Linked Data, using *AK-RDF*. Notice that we are working with Windows.

1. AK-RDF has been developed with MonoDevelop/Xamarin Studio, therefore we have to download *Mono for Windows* in order to execute it properly. Go to http://www.go-mono.com/mono-downloads/download.html and select the latest stable version available. At the time of writing, the latest release is 2.10.9.
2. Then, run the setup file in order to install Mono for Windows on your computer. Once you have done it, execute the Mono Command Prompt, locate the path on your AK-RDF-Windows directory and write *mono RDFinterface.exe*, as you see in Fig 93.

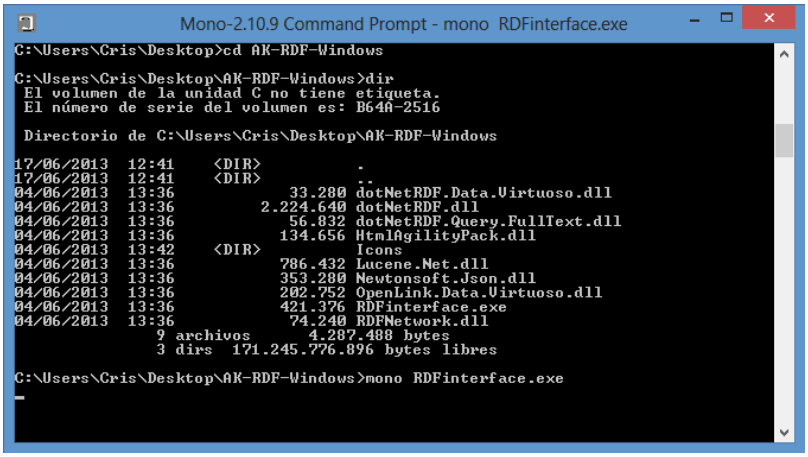


Fig 93 – Mono Command Prompt

After that, AK-RDF main interface is running, as you see in Fig 94.

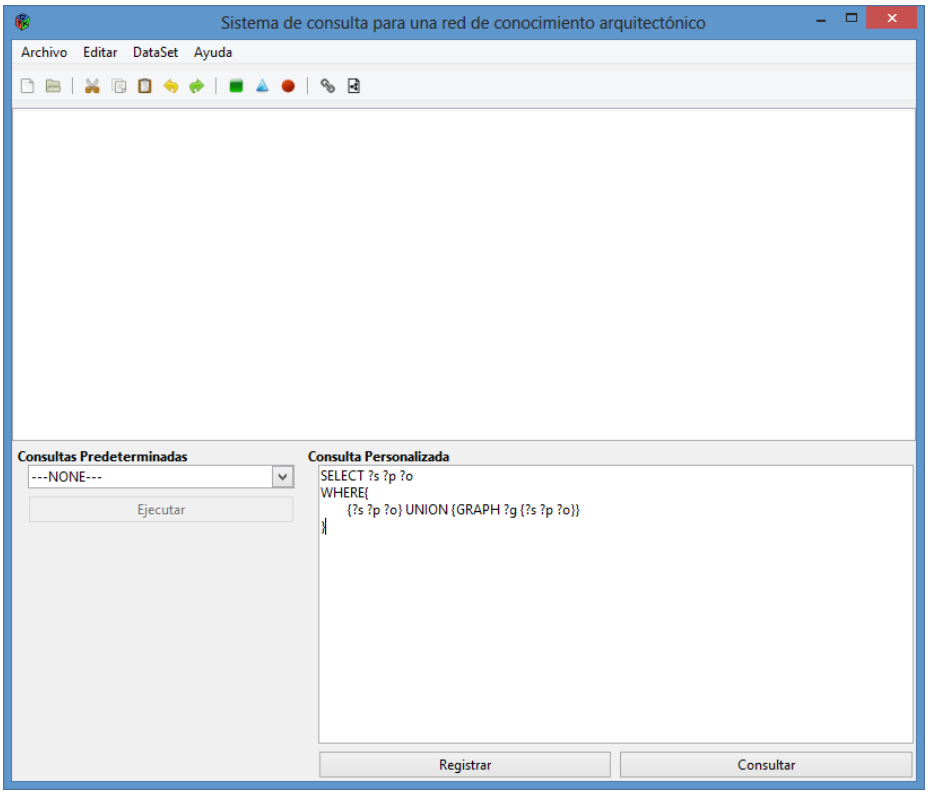


Fig 94 – *AK-RDF* main interface

3. Next, we have to register a new dataset in order to use a specific server, in this case we are going to choose Fuseki server. Go to Archivo-Nuevo DataSet, as you see in Fig 95.

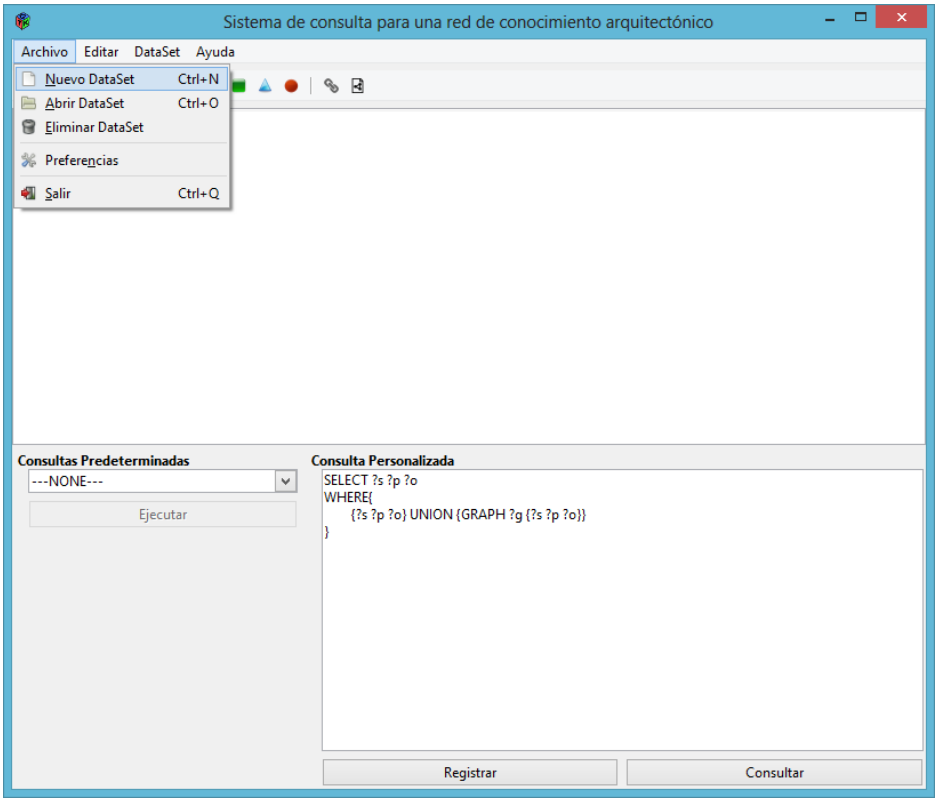


Fig 95 – Registering a new server with *AK-RDF* (1)

Then, a new window will appear in order to register the new server. We select Fuseki option.

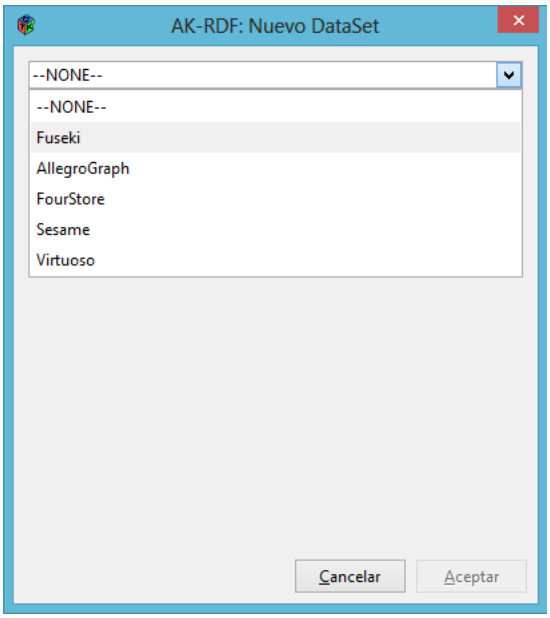


Fig 96 - Registering a new server with *AK-RDF* (2)

And fill in the required information about, in this case, Fuseki server.

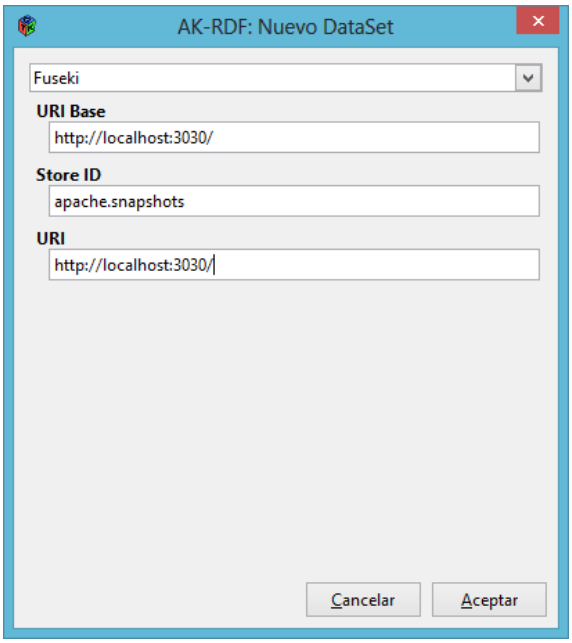


Fig 97 - Registering a new server with *AK-RDF* (3)

4. Then, we have to select this dataset in order to work with it. Go to Archivo-Abrir DataSet.

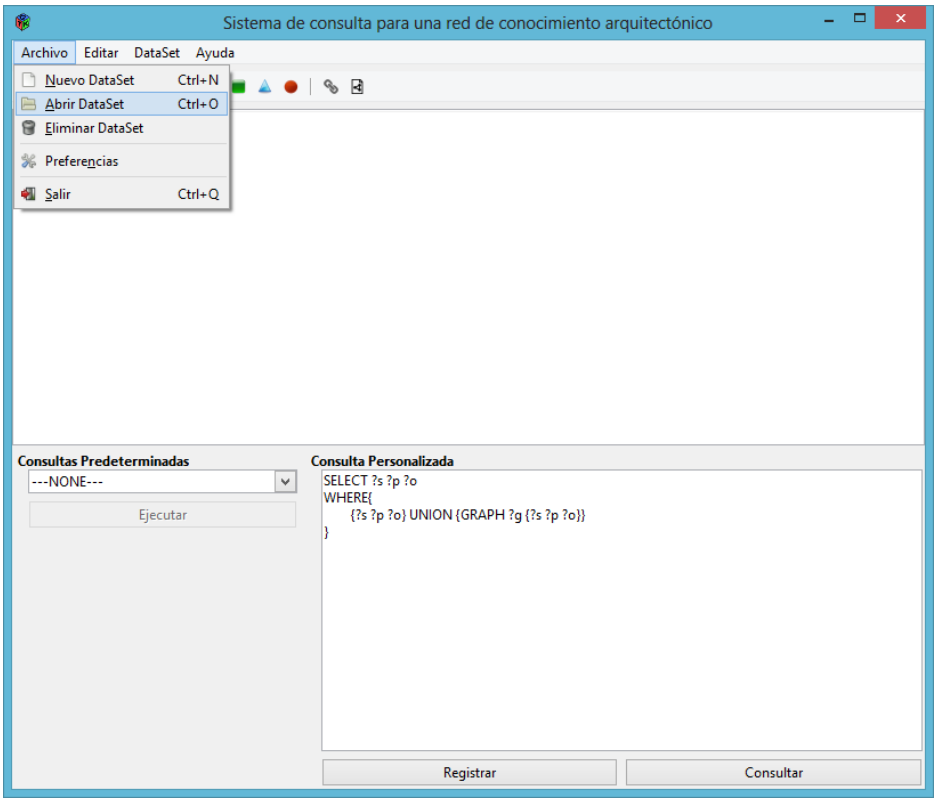


Fig 98 – Opening server with *AK-RDF* (1)

Then, select the Fuseki server which we have already created and click *Aceptar* button.

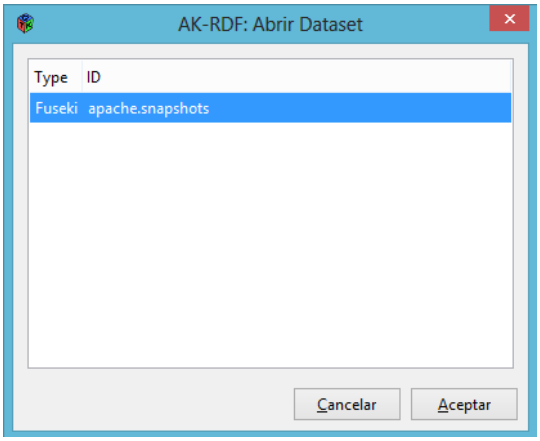


Fig 99 - Opening server with *AK-RDF* (2)

5. The next step is to create our own design decision network with AK-RDF in order to manage it later. Go to DataSet-Nuevo Elemento…-Decisión de Diseño, as you can see in Fig 100, to create a new design decision.

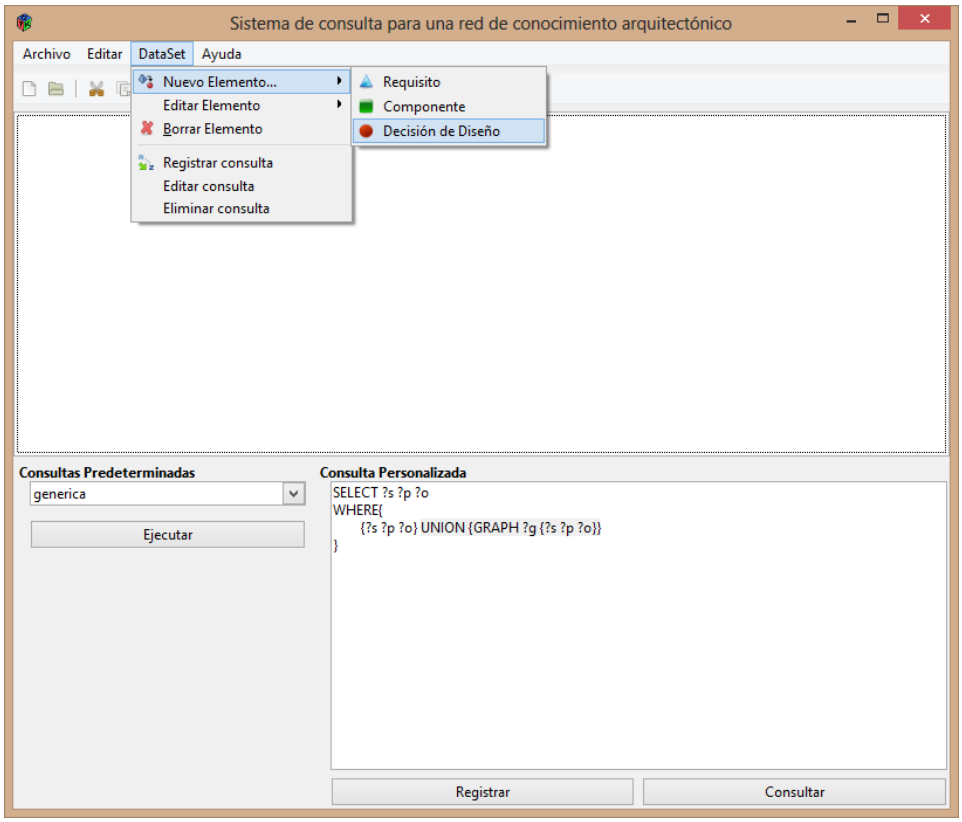


Fig 100 – Creating a design decision network with *AK-RDF* (1)

Thus, Fig 101 illustrates an example of the creation of design decision DD1, including all its properties indicated at the beginning of section 4 in RDF/XML format, namely:

```
<rdf:Description rdf:about="http://archk.tk/DD1">
        <att:author>Cris</att:author>
        <att:category>Structural</att:category>
        <att:decision>Using a three-layered architecture</att:decision>
        <att:risk rdf:datatype="&xsd;integer">0</att:risk>
        <att:scope>Universal</att:scope>
        <att:state>Approved</att:state>
        <att:rationale>Rationale1</att:rationale>
        <rat:Layer>Business</rat:Layer>
        <att:timeStamp rdf:datatype="&xsd;dateTime">2012-05-01T13:55:33.280368+01:00</att:timeStamp>
        <kind:classification>Existence</kind:classification>
        <conn:constrains rdf:resource="http://archk.tk/DD2" />
        <conn:constrains rdf:resource="http://archk.tk/DD3" />
        <conn:constrains rdf:resource="http://archk.tk/DD4" />
        <conn:constrains rdf:resource="http://archk.tk/DD5" />
        <prop:type>DesignDecision</prop:type>
</rdf:Description>
```

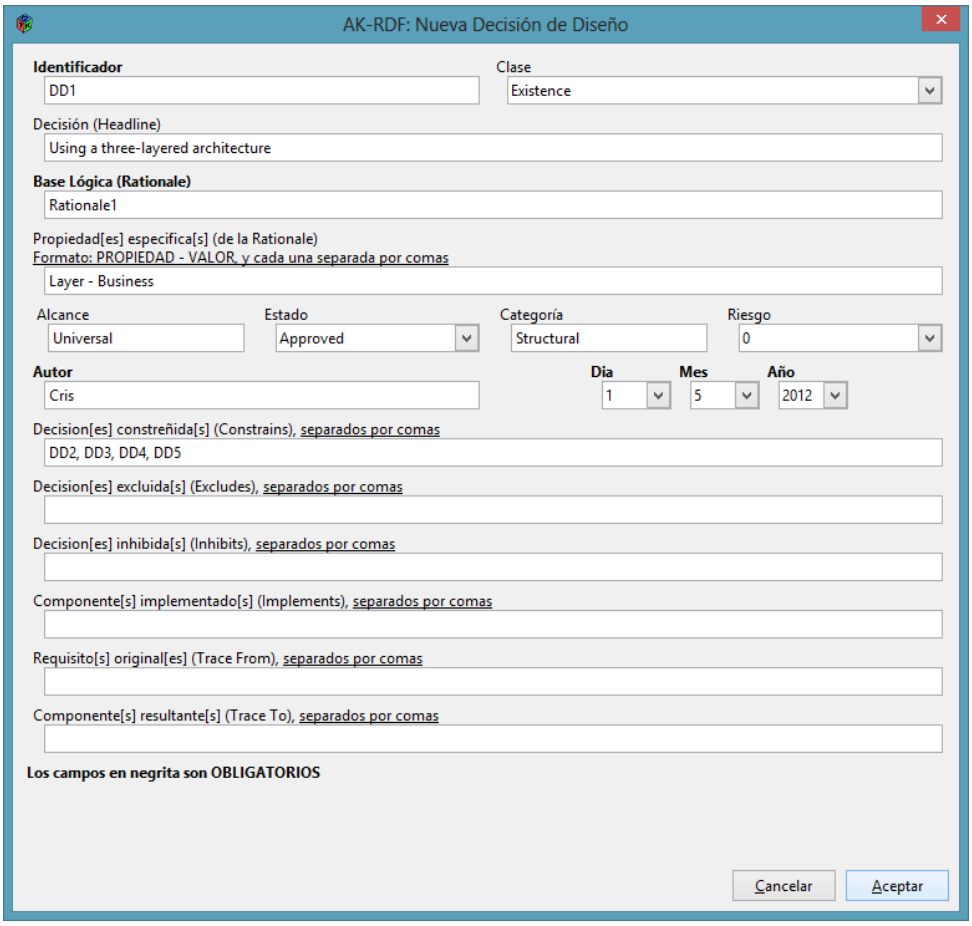


Fig 101 - Creating a design decision network with *AK-RDF* (2)

Finally, we click on the *Aceptar* button and our design decision DD1 will be created.

This final step returns an error, so DD1 cannot be created. Apart from that, we don't include this tool in our feature analysis because it is based on Apache Jena and Fuseki (section 4.3) so that we are really using and analysing the same technology.

5 Feature analysis

In this section, we carry out an analysis between the Linked Data tools presented in section 4 in order to compare their features and determine what is/are the best one/s. Table 1, Table 2 and Table 3 show their own features. These tables detail some features that we have considered highly relevant in order to manage AK data, namely:

- *Type of tool* specifies the type of the analysed Linked Data tool.
- *Interaction UI* (User Interface) indicates if the Linked Data tool has a friendly user interface or not.
- *Data persistence* indicates if the Linked Data tool provides persistence for its stored data.
- *Data storage* specifies where the data is stored.
- *Query languages* points out the different query languages that we can use to manipulate our linked data.
- *Supported schemas/vocabularies* within the Linked Data tool, like XML or RDF.

- *Federated queries* indicates if the Linked Data tool supports data searching across multiple databases.
- *Input data formats* supported by the Linked Data tool in order to store and manage it.
- *Query output formats* provided by the Linked Data tool when you execute a query.
- *License* informs of the type of license that the Linked Data tool has.
- *Security* indicates how the Linked Data tool guarantees and ensures that the information is always safely stored.
- *RDF serialization formats* supported by the Linked Data tool, like RDF/XML, Turtle, etc.
- *SDK*[4] *support* indicates if the Linked Data tool allows you to create applications for a certain software framework.
- *Complexity* of the Linked Data tool, i.e. how easy is it to install, to manage data on it or to use. Namely, complexity of *Installation*, *Start-up* and *Data management*.

[4] Software Development Kit

		Virtuoso	LMF	Apache Jena + Fuseki
Type of tool		Server	Server application	Java framework + SPARQL server
Interaction UI		OpenLink Data Spaces (ODS) - Briefcase	LMF	Fuseki
Data persistence		Yes	Yes	Yes (*--loc=DIR* option when starting Fuseki)
Data storage		Virtuoso server	LMF server	Fuseki server
Query languages		SQL, SPARQL, XQuery, XPath 1.0, XSLT 1.0	SPARQL	SPARQL, RDQL
Supported schemas/vocabularies		RDF, XML, FOAF, OWL	RDF, DC, SKOS, *RDF Path Program* (to adapt the search to our specific domain)	RDF, OWL, RDFS, DC, RDFa, OWL 2, RSS, VCARD, DB (for Database properties)
Federated queries		Yes (SPARQL 1.1)	Yes (SPARQL 1.1)	Yes (SPARQL 1.1)
Input data formats		XML, SQL, RDF, free text	CSV, Excel, XML, RDF	RDF, XML
Query output formats		HTML, XML, JSON, Javascript, NTriples, RDF/XML, spreadsheet	SPARQL-XML, JSON, HTML, chart (Sgvizler), XML	JSON, XML, Text, CSV, TSV, SSE[5] (only in Apache Jena ARQ, *ResultSetFormatter* class)
License		GPL[6] v2 (Open source: OpenLink Virtuoso)	Apache license (Free software)	Apache license (Free software)
Security		User permissions on folders and login to ODS	Authentication and authorization mechanism (pre-defined profiles)	Authentication and control of the number of concurrent requests can be added using an Apache server and either blocking the Fuseki port to outside traffic or by listening only the *localhost* network interface. Data can be updated without access control if the Fuseki server is started with the *--update* argument. If started without that argument, data is read-only.
RDF serialization formats		N-Triples, Turtle, N3, RDF/XML, RDFa	RDF/XML, N3, Turtle, N-Triples, JSON-LD (JSON for Linked Data)	Turtle, RDF/XML, N-triples, RDF/JSON, TriG, N-Quads, RDFa, N3
SDK support		No	No	Yes
Complexity	Installation	Low	Low	High (Jena Java libraries + Eclipse + Fuseki)
	Start-up	Low	Low	Medium
	Data management	Low	Medium	Medium

Table 1 - Features of Linked Data tools (I)

[5] SPARQL Syntax Expressions - http://jena.apache.org/documentation/notes/sse.html
[6] GNU *General Public License*

		TopBraid Suite	Sesame	Mulgara
Type of tool		Suite of Semantic products	Java framework	RDF database
Interaction UI		TopBraid Composer/Live	OpenRDF Workbench	Mulgara Viewer
Data persistence		Yes (TBL)	Yes (*In Memory Store* repository - *Persist* parameter)	Yes (*Resolver Database Class* configuration)
Data storage		TBL Personal/Enterprise server	Sesame server	Mulgara server
Query languages		SPARQL	SPARQL, SeRQL	TQL, SPARQL
Supported schemas/vocabularies		RDF, RDFS, OWL, RDFa, XML	RDF, RDFS, OWL, DC, FOAF, XML, DOAP, EARL, XPath, Sesame, Sesame QName, SKOS	RDF, OWL, DAML, OIL
Federated queries		Yes (SPARQL 1.1)	Yes (SPARQL 1.1)	No
Input data formats		RSS/Atom, RDBMS, CMS, XML, JSON, Excel, HTML/Calais text, CSV, e-mail, UML, RDF	RDF	RDF, XML, RDBMS, RSS
Query output formats		XML, HTML, Text/CSV, Text/TSV	BINARY, SPARQL/XML, SPARQL/JSON, SPARQL/TSV, SPARQL/CSV	HTML
License		Commercial, Pay Licensed Closed Source[7]	BSD licenses[8] (Free software)	Open Software License 3.0
Security		Version control and governance[9]	User and password (server)	The commercial code that Mulgara was originally based on, contained both transport layer security (SSL/TLS) and store level authentication/authorization. The open source releases of Mulgara contain no security infrastructure.
RDF serialization formats		RDF/XML, N3, N-Triples, Turtle	TriG, BinaryRDF, TriX, N-Triples, N-Quads, N3, RDF/XML, RDF/JSON, Turtle	RDF/XML, N3, N-Triples
SDK support		No	Yes	No
Complexity	Installation	Low	Medium	Low
	Start-up	Low	Medium	Low
	Data management	Medium	Low	Low

Table 2 - Features of Linked Data tools (II)

[7] *Pay Licensed Closed Source* is a category of commercial software licenses distinguished by the fact that you must pay to use a software and cannot view the source code.

[8] A *BSD-style license* is a particular free software license that grants wide permissions on usage and redistribution of a program.

[9] TBC can be used with Subversion (SVN). *Data governance* is a system that describe who can take what actions with what information, and when, under what circumstances, using what methods.

		RedStore	Callimachus
Type of tool		RDF triplestore	Linked Data management system
Interaction UI		RedStore	Callimachus
Data persistence		Yes (*-t <options=hashes, mysql, postgresql, sqlite, tstore or virtuoso>*)	Yes
Data storage		HTTP server	Callimachus server
Query languages		SPARQL, LAQRS, RDQL	SPARQL
Supported schemas/vocabularies		RDF	RDF, RDFS, OWL, RDFa, SKOS, DC, FOAF, Freebase, GoodRelations, GeoNames, Open Graph Protocol, SIOC, VCARD
Federated queries		Yes (SPARQL 1.1)	Yes (SPARQL 1.1)
Input data formats		RDF	RDF
Query output formats		XML, JSON, Table, CSV, TSV, HTML, Turtle, RDF/XML, N-Triples, RDF/XML (XMP Profile), RDF/XML (Abbreviated), RSS 1.0, Atom 1.0, GraphViz DOT format, RDF/JSON Triples, RDF/JSON Resource-Centric, N-Quads	HTML
License		GPL (Free software)	Apache License 2.0 (Free software)
Security		User permissions for the database server (MySQL, PostgreSQL, Tstore and Virtuoso)	By default, it allows public read-only access and requires authenticated users for editing and deleting resources. Authorization is setup using user accounts and groups.
RDF serialization formats		RDF/XML, N-Triples, Turtle, N-Triples-plus, N3, TriG, RSS Tag Soup, guess, RDFa, N-Quads	RDF/XML, Turtle, RDFa
SDK support		No	No
Complexity	Installation	High	Medium
	Start-up	Medium	Low
	Data management	Low	Medium

Table 3 - Features of Linked Data tools (III)

Notice that almost all Linked Data tools are able to provide federated queries as they support SPARQL 1.1. SPARQL [22] can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. Namely, *SPARQL 1.1 Federated Query* extension has been created for executing queries distributed over different SPARQL endpoints.

Now, in order to get to know which the best/worst Linked Data tools are regarding to their features, we are going to establish our preferences for every feature and how we are going to score them.

ANALIZED FEATURES		PARAMETER	PREFERENCES	HOW TO SCORE
Type of tool		-	(Without preferences)	(Not scoring, only informative)
Interaction UI		IUI	A User Interface that allows users to interact with the Linked Data tool.	**1** If the tool has a UI **0** Otherwise
Data persistence		DP	A platform that provides data persistence over the time.	**1** If the tool provides data persistence **0** Otherwise
Data storage		-	(Without preferences)	(Not scoring, only informative)
Query languages*		QL1	A tool that supports, at least SPARQL.	**1** If the tool supports SPARQL **0** Otherwise
		QL2	The more query languages supported the more preferable.	**(Number of query languages supported)/5**
Supported schemas/vocabularies*		SS1	A tool that supports, at least RDF and OWL as schemas.	**1** If the tool supports RDF and OWL as schemas **0** Otherwise
		SS2	The more schemas supported the more preferable.	**(Number of supported schemas)/13**
Federated queries		FQ	A tool that has the option of carry out federated queries.	**1** If the tool provides federated queries **0** Otherwise
Input data formats*		IDF1	A tool that supports, at least RDF as an input data format.	**1** If the tool supports RDF as an input data format **0** Otherwise
		IDF2	The more input data formats supported the more preferable.	**(Number of input data formats)/11**
Query output formats		QOF	The more query output formats supported the more preferable.	**(Number of query output formats)/17**
License		Li	A tool with an Open Software license.	**1** If the tool has an Open Software license **0** Otherwise
Security		Sec	A tool with security services.	**1** If the tool has security services **0** Otherwise
RDF serialization formats*		RSF1	A tool that supports, at least RDF/XML as a RDF file format.	**1** If the tool supports RDF/XML as a RDF file format **0** Otherwise
		RSF2	The more RDF file formats supported the more preferable.	**(Number of RDF file formats)/10**
SDK support		SDKS	A tool that provides SDK support.	**1** If the tool has SDK support **0** Otherwise
Complexity	Installation	CI	It is desirable a *Low* complexity in all cases.	**0.2** If CI is *Low* **0.1** If CI is *Medium* **0** If CI is *High*
	Start-up	CSU		**0.3** If CSU is *Low* **0.15** If CSU is *Medium* **0** If CSU is *High*
	Data management	CDM		**0.5** If CDM is *Low* **0.25** If CDM is *Medium* **0** If CDM is *High*

Table 4 – Feature preferences

(*) At least one specific value, such as SPARQL or RDF. These features are compulsory.

Therefore, the formula used to mark each tool will be the following one:

$$\textit{Total score} = QL1 \cdot SS1 \cdot IDF1 \cdot RSF1 \cdot (IUI + DP + QL2 + SS2 + FQ + IDF2 + QOF + Li + Sec + RSF2 + SDKS + CI + CSU + CDM)$$

These parameters will be calculated using the rules presented in Table 4, namely in *How to score* column. Parameters QL1, SS1, IDF1 and RSF1 represent the compulsory features, therefore their values will be 1 or 0, depending on their support.

In this manner, the tool that obtains the highest score will be the best Linked Data tool analysed and thus, the lowest score will belong to the worst Linked Data tool. Notice that every feature is considered as important as the others, therefore they are equally scored, namely *1 point* per each feature at most –so the score varies between 0 and 1 and the maximum score is 12.

Table 5 shows the marks for each Linked Data tool. Notice that we have omitted the two features that are not scoring, i.e. *Type of tool* and *Data storage*.

		Virtuoso	LMF	Apache Jena & Fuseki	TopBraid Suite	Sesame	Mulgara	RedStore	Callimachus
Interaction UI	IUI	1	1	1	1	1	1	1	1
Data persistence	DP	1	1	1	1	1	1	1	1
Query languages	QL1	1	1	1	1	1	1	1	1
	QL2	5/5=1	1/5=0.2	2/5=0.4	1/5=0.2	2/5=0.4	2/5=0.4	3/5=0.6	1/5=0.2
Supported schemas/vocabularies	SS1	1	**0**	1	1	1	1	**0**	1
	SS2	4/13=0.31	4/13=0.31	9/13=0.69	5/13=0.39	12/13=0.92	4/13=0.31	1/13=0.08	13/13=1
Federated queries	FQ	1	1	1	1	1	0	1	1
Input data formats	IDF1	1	1	1	1	1	1	1	1
	IDF2	4/11=0.36	4/11=0.36	2/11=0.18	11/11=1	1/11=0.09	4/11=0.36	1/11=0.09	1/11=0.09
Query output formats	QOF	7/17=0.41	5/17=0.29	6/17=0.35	4/17=0.24	5/17=0.29	1/17=0.06	17/17=1	1/17=0.06
License	Li	1	1	1	0	1	1	1	1
Security	Sec	1	1	1	1	1	0	1	1
RDF serialization formats	RSF1	1	1	1	1	1	1	1	1
	RSF2	5/10=0.5	5/10=0.5	8/10=0.8	4/10=0.4	9/10=0.9	3/10=0.3	10/10=1	3/10=0.3
SDK support	SDKS	0	0	1	0	1	0	0	0
Complexity	Installation	CI	0.2	0.2	0	0.2	0.1	0.2	0	0.1
	Start-up	CSU	0.3	0.3	0.15	0.3	0.15	0.3	0.15	0.3
	Data management	CDM	0.5	0.25	0.25	0.25	0.5	0.5	0.5	0.25
TOTAL SCORE			**8.58**	**0**	**8.82**	**6.98**	**9.35**	**5.43**	**0**	**7.3**

Table 5 - Linked Data tools scores

As we can see in Table 6, the first position is for *Sesame* (9.35), second *Apache Jena and Fuseki* (8.82), then *Virtuoso* (8.58), *Callimachus* (7.3), *TopBraid Suite* (6.98), *Mulgara* (5.43), and the last ones, *RedStore* (0) and *Linked Media Framework* (0) that have no score because they do not support OWL as an schema (parameter SS1).

Position	Linked Data tool	Score
1	Sesame	9.35
2	Apache Jena & Fuseki	8.82
3	Virtuoso	8.58
4	Callimachus	7.3
5	TopBraid Suite	6.98
6	Mulgara	5.43
7	RedStore	0
8	LMF	0

Table 6 - Linked Data tools ranking

6 References

[1] Open Knowledge Foundation, “Open Data - An Introduction,” 2012. .

[2] C. Bizer, T. Heath, and T. Berners-Lee, “Linked Data - The Story So Far,” *Int. J. Semant. Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, Jan. 2009.

[3] T. Berners-Lee, “Linked Data - Design Issues,” 2009. [Online]. Available: http://www.w3.org/DesignIssues/LinkedData.html. [Accessed: 18-Apr-2013].

[4] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, vol. 1, no. 1. Morgan & Claypool, 2011, pp. 1–136.

[5] OpenLink Software, “Virtuoso Universal Server,” 2013. [Online]. Available: http://virtuoso.openlinksw.com/. [Accessed: 05-Jun-2013].

[6] OpenLink Software, “OpenLink Data Spaces Framework & Application Suite,” 2009. [Online]. Available: http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/OdsApplicationSuite. [Accessed: 05-Jun-2013].

[7] OpenLink Software, “OpenLink Data Spaces (ODS),” 2013. [Online]. Available: http://my.openlinksw.com/ods/. [Accessed: 25-Sep-2013].

[8] OpenLink Software, “OpenLink Virtuoso SPARQL Query Editor,” 2013. [Online]. Available: http://my.openlinksw.com/sparql/. [Accessed: 25-Sep-2013].

[9] OpenLink Software, “OpenLink iSPARQL,” 2009. [Online]. Available: http://my.openlinksw.com/isparql/. [Accessed: 25-Sep-2013].

[10] Semantic Web, “Linked Media Framework,” 2012. [Online]. Available: http://semanticweb.org/wiki/Linked_Media_Framework.

[11] The Apache Software Foundation, “Apache Jena,” 2013. [Online]. Available: http://jena.apache.org/. [Accessed: 10-Jun-2013].

[12] TopQuadrant, “TopBraid Suite,” 2013. [Online]. Available: http://www.topquadrant.com/products/TB_Suite.html. [Accessed: 17-Jun-2013].

[13] TopQuadrant, “TopBraid Composer,” 2013. [Online]. Available: http://www.topquadrant.com/products/TB_Composer.html. [Accessed: 11-Jun-2013].

[14] TopQuadrant, “TopBraid Live,” 2013. [Online]. Available: http://www.topquadrant.com/products/TB_Live.html. [Accessed: 26-Jun-2013].

[15] OpenRDF, “Sesame 2.7,” 2013. [Online]. Available: http://openrdf.callimachus.net/sesame/2.7/docs/users.docbook?view. [Accessed: 11-Jun-2013].

[16] Mulgara Project, “Mulgara,” 2012. [Online]. Available: http://www.mulgara.org/. [Accessed: 24-Jun-2013].

[17] T. Adams, “Mulgara Semantic Store,” 2007. [Online]. Available: http://docs.mulgara.org/overview/faq.html. [Accessed: 24-Jun-2013].

[18] N. J. Humfrey, “RedStore,” 2011. [Online]. Available: http://www.aelius.com/njh/redstore/. [Accessed: 25-Jun-2013].

[19] “Callimachus,” 2013. [Online]. Available: http://callimachusproject.org/. [Accessed: 19-Jul-2013].

[20] Simile, “Longwell,” 2008. [Online]. Available: http://simile.mit.edu/wiki/Longwell. [Accessed: 20-Jun-2013].

[21] C. Cyganiak, Richard;Bizer, “Pubby – A Linked Data Frontend for SPARQL Endpoints,” 2011. [Online]. Available: http://wifo5-03.informatik.uni-mannheim.de/pubby/. [Accessed: 25-Jun-2013].

[22] W3C, “SPARQL 1.1 Federated Query,” 2013. [Online]. Available: http://www.w3.org/TR/sparql11-federated-query/. [Accessed: 05-Jul-2013].

[23] A. Hogan, J. Umbrich, A. Harth, R. Cyganiak, A. Polleres, and S. Decker, “An empirical survey of Linked Data conformance,” *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 14, pp. 14–44, Jul. 2012.

[24] C. Bizer, R. Cyganiak, and T. Heath, “How to publish Linked Data on the Web,” 2008. [Online]. Available: http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/. [Accessed: 26-Jul-2013].

[25] A. Tang, “A Rationale-based Model for Architecture Design Reasoning,” Swinburne University of Technology, 2007.