# MASHQL, A STEP TOWARDS SEMANTIC WEB PIPES

Constantinos Savvides

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

May 2010

# ABSTRACT

This thesis is motivated by the massively increasing structured data on the Web, and the need for novel methods to exploit these data to their full potential. Building on the remarkable success of Web 2.0 mash-ups, and especially Yahoo! Pipes, we represent a design and implementation of a server-side interactive query formulation language, MashQL [17], which allows people to navigate, query, and mash-up a data source(s) without any prior knowledge about its schema, vocabulary, or technical details. Assuming the Internet data is represented in RDF, we can query by using SPARQL which is the recent recommendation by the W3C. Furthermore, Oracle 11g semantic technology supports an efficient way for RDF storage and query in a SPARQL-like style. We also present the technologies that we have use in our implementation and how they are related to it.

In addition we present the design issues that we have to consider, so that we could achieve a nice user-friendly environment for our users. The user interface is inspired from the interface that Yahoo Pipes has, because of its simplicity to formulate pipes in a visual way. We also refer to the system and software architecture that we have follow, in order to develop a successful web application.

We also evaluate the usability of the editor and obtain the first impressions and comments from our participants, about using this unfamiliar technology, by completing some scenarios. By evaluating these scenarios, we will understand the potentials of our editor and also know where we can modify and evolve our editor in a future version. Also we evaluate the time-cost of formulating a MashQL query and time-cost of loading RDF sources into oracle's semantic technology, and present the limits of our implementation.

# MASHQL, A STEP TOWARDS SEMANTIC WEB PIPES

Constantinos Savvides

A Thesis

Submitted in Partial Fulfilment of the

Requirements for the Degree of

Master of Science

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

May, 2010

# APPROVAL PAGE

Master of Science Thesis

**MASHQL, A STEP TOWARDS SEMANTIC WEB PIPES**

Presented by

Constantinos Savvides

Research Supervisor

________________________________________

Marios D. Dikaiakos

Committee Member

________________________________________

Panagiotis Zacharias

Committee Member

________________________________________

Demetris Zeinalipour

University of Cyprus

May, 2010

# ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Marios Dikaiakos for his significant help, commentary and guidance throughout the whole duration of writing this thesis, including helping me with the evaluation part of it.

Also I would like to thank my co-supervisor Dr Mustafa Jarrar for his help and guidance in the implementation part of my thesis, and also the ideas he provide about evaluating the application.

In addition I would like to thank Dr. Panayiotis Zaharias for his contribution, to write the usability test report, and also the 20 participants that accepted to have the usability test of our implementation.

Last but not least, I would like to thank my family for their support and tolerance.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# Chapter 1

# Introduction

## 1.1 Motivation and contribution

We are currently running on the radiance of glory of Web 2.0 World Wide Web technology, by leaving Web 1.0 far away. Every day we use Web 2.0 and we realize that it is established in our life. But, who wonders, how we came here? Who is extending the Web 1.0 to have today the Web 2.0? The answer for all this is very simple; we are the creators of the Web 2.0. In Web 2.0 era, it is the users who add value to services. We can regard Web 2.0 as a read/write platform that everybody can share everything such as photos, videos, music, bookmarks, articles, viruses' threads etc. Additionally we can see Web 2.0's pioneer companies to moving in "long tail" philosophy for their target market rather than focus on center target market. In addition companies are taking advantage of their Web services using the edges of the internet rather than the center.

The advent of mashups has made possible the combination of data into a new service. One no longer has to navigate across the internet to find the different data he/she wants to have. He/She can be informed about the recent news through a mashup of his own. He can get the sports news along with the politics news in one application using technologies like RSS, Atom and JSON. One can access available data on the Web through xml files that can be imported to his application and can also filter or sort them in any kind of way one likes.

Semantic Web allows two different applications to exchange information meaningfully, thus, increase the use of information to their full potential. The Resource Description Framework (RDF) is an attempt to provide the means to represent the metadata of the Web applications in well-formed manner.

Semantic Web pipes, is a powerful paradigm for building RDF-based mashups. It works by querying, operating, and producing an output which is accessible via a stable URL, by using the RDF resources. RDFa enables the developers to make an HTML page have double duty. That is, the page works as a presentation page and as a machine-readable source which is structured in RDF data.

To expose the massive amount of public content and allow people to build mash-ups in an easy way, several mash-up editors have been launched, like Google Mashup, Yahoo! Pipes and others. However, Yahoo! Pipes received the most attention thanks to its simplicity and user-friendly environment. In addition Yahoo! Pipes are considered to be a milestone in the history of the internet [26]. On the other hand, Yahoo! Pipes are limited since the mash-ups are focused only on Web feeds (only capable to presenting news items and not capable representing data retrieved from Deep Web and encoded in RDF and XML).

We present an implementation of an interactive query formulation language, MashQL, which proposed by Dr. Mustafa Jarrar and Dr. Marios Dikaiakos [17], to regard mash-ups as data queries and view the Internet as a database, where each source is seen as a table and a mash-up as a query. Assuming the Internet data is represented in RDF (plays the role of a semantically enabled metadata model), we can query by using SPARQL which is the recent recommendation by the W3C. Furthermore, Oracle 11g semantic technology supports RDF storage and query in a SPARQL-like style. This implementation is scalable as shown by Oracle [3].

Summary of Contributions:

1. **Design** (chapter 4). We present the design issues that we have to consider, so that we could achieve a nice user-friendly environment for our users. The user interface is inspired from the interface that Yahoo Pipes has, because of its simplicity to formulate pipes. Also we have to consider where each of the functionality will be executed in order to not have time-cost delays and not overload the application server site. We have decided to follow the three-tier architecture, because this architecture is simple and easily scalable. We have also followed the Model-View-Controller pattern because it allows the separation of Data and Presentation, which make things easier to implement especially when we have to deal with a Web application.
2. **Implementation** (chapter 4). Considering the design that we have implement, we present an implementation of MashQL, proposed by Dr. Mustafa Jarrar and Dr. Marios Dikaiakos [17], as a server-side mashup editor, and why it is a promised solution to query RDF resources. We choose to implement the editor by using JSP because is a cross-platform language, AJAX for the asynchronous calls, which we will deal with a lot of database interaction, JavaScript for the dynamic content of the visual modules, and XML for data transferring (section 2.3). We will also conclude if actually the Oracle's Semantic Technology (a new technology available in the latest Oracle database version)

can be adapted, provide an efficient way to query RDFs, and also verify that we can achieve our assumptions for a successful mash-up formulation in an easy way.

3. **Evaluation** (chapter 5). Evaluate the usability of the editor and get the first impressions and comments about using this unfamiliar technology. We prepared a list of 20 participants, with age range 18 – 44, and separated them into groups of two or individuals. We chose to have small groups so that we could handle the participants better, observing their interaction and the difficulties they faced during the completion of the six scenarios that we have provide them. Before presenting the scenarios, we formulated a 15 minute training session for the participants, to demonstrate the editor and present some query formulation examples. Also evaluate the response-time of MashQL on a dataset, by using the Oracle's semantic technology. We have use RDF sources with content length 1MB, 2MB, 5MB, and 10MB respectively. By evaluating these scenarios, we will know the potentials of our editor and also know where we can modify and evolve our editor in a future version.

## 1.2 Challenges

A mash-up editor allows people to combine different data sources into mash-ups, in a graphical and user-friendly way without having to write code and have technical experience to get started with. A challenge is to allow to our users, to navigate, query, and mash up a data source(s) without any prior knowledge about its schema, vocabulary, or technical details, in a nice graphical way.

A challenge to consider in the implementation is that a query might involve multiple sources. In this point of view, we have to allow users to query structured data flexibly when having many sources. We have to invent a mechanism that will allow the involvement of multiple resources in an easy and efficient way for the users.

Building data mush-ups is a challenge, since there is not yet an approach to easy access and expose structured data on the Web. Also this task is usually limited to high skilled programmers [18]. In the case of using RDF and SPARQL, this challenge is more complicated ever for the IT people [16], because the RDF sources are typically large and bulky and SPARQL is not familiar to most Internet users. So our implementation has to focus on these challenges and develop an editor that will consider these aspects.

Allowing people to easily query data in an open environment, it should consider four assumptions [17] that we can fulfill in our implementation:

1. The user does not have to know the schema.
2. The data might be schema-free.
3. A query may involve multiple data sources.
4. The query method is sufficiently expressive (not merely a single-purpose user interface).

We will consider these assumptions in chapter 2 in related work section.

## 1.3 Thesis Structure

The structure of the thesis is as follows: In chapter 2 we present the related work done so far in this sector, including mash-up editors review, that MashQL is inspired from, especially the Yahoo! Pipes. Also we present the related technologies that are used to implement the MashQL editor. In chapter 3 we define the data model, the syntax, and the semantics of MashQL, and also the query formulation algorithm that helps the user to formulate the query modules. Next in chapter 4, we present the design and implementation issues of MashQL in detail, including the functionality and mechanism that have been implemented, and how the editor interacts with the oracle's semantic technology. In Chapter 5 we present the experiments concerning the usability and time-cost evaluations of the editor. Finally in chapter 6, we sum up with the conclusions and future work.

# Chapter 2

## Related Work & Technologies

### 2.1 Query Formulation Approaches

The query formulation refers to the ability of allowing people querying data sources (e.g. RDF, XML, RSS etc) into an easy and simple way. In this section we refer to the main approaches to query formulation and how these are related to our MashQL editor. They are many approaches included in [17].

**Query-by-Form** approach is the simplest one defined. However, because of this simplicity, this approach is not flexible or expressive and also fails with the assumptions 2-4 stated in section 1.2. Query by form allows a person to enter some criteria into the query by using a form rather than having to understand the whole machinations of the queries themselves [27]. To understand this approach let say that if a person wants to view a list of all the people attending in a particular course. The user might type the course code into a field on a form, and then the form would feed the criterion into the query directly. The result of this query will be displayed on the very same form.

**Query-by-Example** approach is a known language for querying (like SQL) relational databases [19]. However, it differs from SQL and from most other database query languages, in having a GUI (graphical user interface) that allows users to formulate queries by creating example tables. To get started, the user needs minimal information, and the whole language contains few concepts. The query by example is mostly used for queries not too complex and expressed in the terms of few tables. Although it is simple, it requires the data to be schemed and the users to be aware of the database schema. For this reason this approach fails with assumptions 1, 2, and 4 stated in section 1.2.

**Conceptual Queries** approach. The information is expressed in its most fundamental form; by using concepts and language familiar to the users (e.g. Employee drives Car). It ignores implementation and external presentation aspects [11]. Many databases are modeled in the conceptual level by using EER (Entity Relationship) or ORM (Object Role Modeling) diagrams. Someone can start querying these databases, starting from these diagrams, selecting part of a given diagram, and their selection is translated into SQL [1, 4, 5, 13, 20]. However, the data must have a schema, and the user must have a good knowledge of that schema. So this approach fails with assumptions 1, 2, 3 and partly with 4 (section 1.2).

**Natural Language Queries** is an approach that allows users to write queries in a natural language as a sentence and then translate the sentence into a formal language like SQL (a database computer language designed for the retrieval and management of data in relational databases) or XQuery (query and functional programming language that is designed to query collections of XML data). Despite that this approach does not require any knowledge of the schema and the language is close to the natural, the problem is that this approach is basically bounded with the language ambiguity and the “free mapping” between the sentences and the data schemes. This makes the approach to fails with assumptions 2, 3, and relatively with 4 (section 1.2).

**Visualize queries** allows a user to quickly identify important features of queries that would take much longer if one was working from the underlying SQL text. Such a query system is the QueryScope [15]. The scope for this approach is to communicate the essence of a query though a visual semantics, to visualize queries in the context of a physical schema, to facilitate searches for similar queries, and make the tuning process productive and readable. In addition this approach requires less programming skills to formulate a query, by visualizing the triples of SPARQL as ellipses connected with arrows. However the approach is not intuitive and also fails with assumptions 1 and 4 (section 1.2).

**Mashup editors and Visual Scripting** approach allows people to write query scripts inside visual modules by the usage of mashup editors like Yahoo! Pipes, Google Mashup editor, sMash, Popfly and others. Those modules are expressed as visual boxes connected though wires (lines) presented with their input and output terminals. There are two approached that are inspired from the visual scripting; an implementation over a DBin 2.0 system [29] and SPARQLMotion of TopQuadrant [28]. Those systems allow the users to write SPARQL queries in text inside boxes and link them to another box in order to have a pipeline of queries. However, they do not provide any guidance on query formulation. MashQL purpose in not only to visualize those boxes and links, but also to help the user to formulate what is inside those boxes. Consequently any query formulated by using MashQL cannot be formulated by using Yahoo! Pipes. MashQL however, is inspired from Yahoo! Pipes, in the way that the Yahoo! Pipes has the interface implemented.

**Interactive Queries** approach which applies for querying schema-free XML sources, without the assumption that the user should know the schema. There are two systems that follow this approach that are related to MashQL. These two systems are the Lore DBMS (Lightweight Object Repository) [7], and the NaLIX [21], a generic interactive natural language query interface to an XML database, a DBMS for semi-structured data under development at Stanford University. However the Lore does not support multiples sources and fails assumption 3 and its expressivity is very basic, failing assumption 4. One the other hand, NaLIX supports a highly user interactive search box, where the use can write a keyword and the system suggests auto-complete words that much this keyword. This mechanism is very intuitive and supported in MashQL, because is a very simple for the user to understand and also does not assume any prior knowledge of the dataset used. The problem with NaLIX is that it cannot play the role of a query language; failing for assumptions 2, 3, and 4.

## 2.2 Mashup Editors

A mashup is a Web 2.0 application that combines data from two or more sources into a completely new service. It could also be considered as a remix and process of data from different resources. The term mashup actually came up from the music industry where in the recent years DJs tend to mashup two or more songs into a one remix.

Mashups can be categorized into three types based on the way they extract data: consumer mashups, data mashups and business mashups [30]. An example of a consumer mashup is a Google maps application, where the mashup combines data elements from multiple sources, hiding the process under a graphical user interface. Data mashups mixes data of similar types from different sources like Yahoo pipes combining RSS feeds from multiple sites into one feed with a graphical front end. An enterprise mashup integrates internal and external sources. For example, a real estate company could mashup data about all houses sold last month in the market with the houses sold within the company. A business mashup is a combination of the above using visualization and aggregation of data, making the application available for business use.

Examples of mashups are: ChicagoCrime.com where we have mapping of the crime incidences throughout the Chicago city by the Police Department. Flickr.com is a mashup using mostly video and photo sharing across the internet. Digg.com mashes up news from various Web sites.

Mashup editors can be classified from the development point of view as:

1. User friendly mashups, like Yahoo pipes where you can drag and drop elements on a form and connect them with pipes. Serena can also be considered as a user friendly tool using GUI interface.
2. Hard coding mashups, where in the editor you are actually building an HTML file with certain tags at the beginning and at the end of the file. For Example in Google mashup editor the starting tag is <gm:page title  = “Mashup”> and the ending is </gm:page>.

Both categories use the RSS files to bring their data into mashups.

### 2.2.1 Yahoo Pipes

Yahoo Pipes is a web application tool/service made from Yahoo [34]. It is called “Pipes” because you can fetch data from any source that supports RSS, Atom, or other XML feeds, extract the data you want, combine them, apply filters and have an output. Even if this tool is still beta, Yahoo Pipes is a powerful tool that has the ability to use and combine together simple commands that you insert, and create an output that meets your needs:

- Aggregate Web feeds with web pages and other services
- Manipulate and mashup content from many sources around the web
- Create Web applications from various sources, and makes them available to public.
- Grab the output of any pipe as RSS, JSON and other formats.
- Combine any feeds into one, sort, filter them and also translate them as you like.
- Geo feeds and browse them on an interactive map like Google and Yahoo maps.

All these could be made by the using the excellent user-friendly interface that Yahoo provides through its site. It’s a very easy-to-learn and easy-to-understand tool and from our experience while exploring the tool, no more than few minutes are needed to create a mashup, even if it contains data that must be displayed in a map, like Yahoo or Google maps. Also, from exploring the documentation of Yahoo pipes that is available, we saw that you can find a well documented, step-by-step tutorial of how to create a simple mashup or even how to use data from a site and combine them to an interactive map. In addition, we found that many pipes already made by other users, which could be cloned, or even used inside other pipes. The site also provides a forum that you can discuss your problems or anything you want with the other yahoo pipes users.

The editor of Yahoo allows one to create and edit pipes in an intuitive visual interface. The editor consists of three panes: the Library with the available modules, the Canvas for assembling the pipes and the Debugger which lets you inspect the Pipe output and any state of the Pipe. MashQL editor is inspired from this type of interface. Figure 1 shows the layout of the tool.

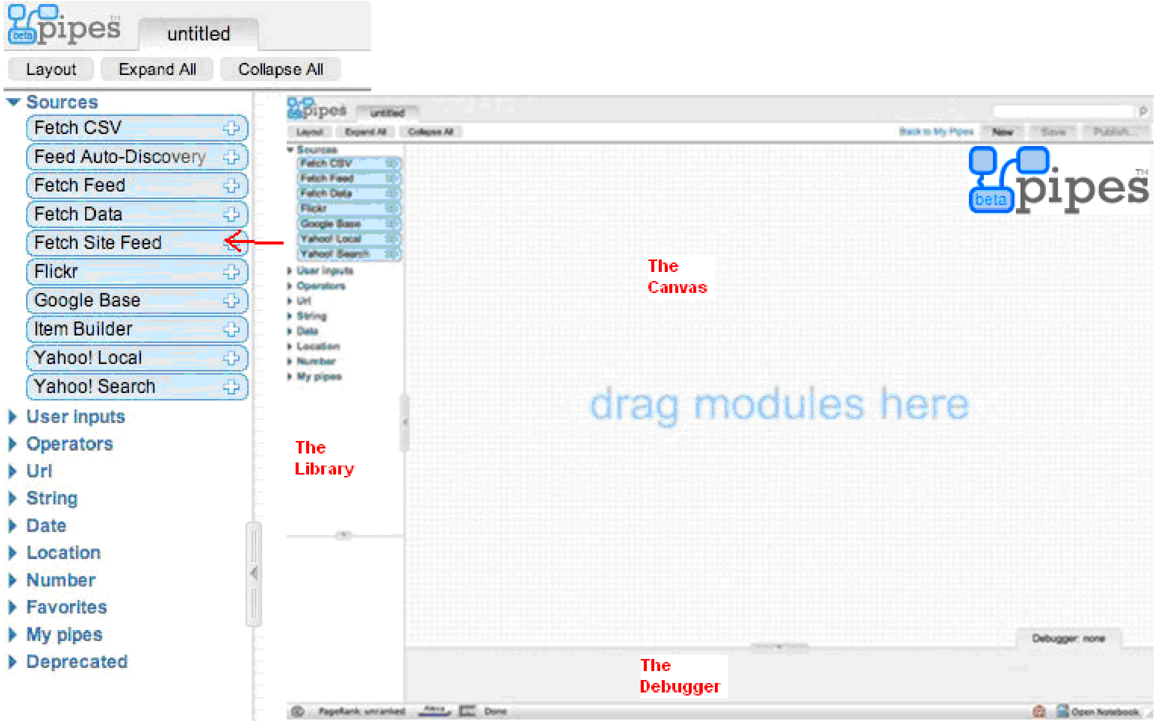


**Figure 1 - Yahoo Pipes Environment**

When you create your pipe, you have the ability to save it to your account by giving the pipe a name. You can also have the option to publish the pipe so that other people can see it and use it. This is a way to distribute your work. So if everyone publishes their work, everyone can gain and learn more, or learn a better ways to use the tool. Once you save the pipe, you can run it and see the results.

To get some ideas of how it works, we have experience and make an example. The figure 2 shows this simple example of Yahoo Pipes usage. In this example you drag Fetch Feed from the Sources bar of the editor. Then you enter the full URL, which supports RSS, to the RSS feed you would like to customize. Then you can filter the articles that you don’t want to include in the output, by dragging the Filter module from the Operations menu into the main workspace. Then you link the Fetch Feed module with the Filer module (Note that you can see the output of the feed by clicking on the module you would like to see the output). Then use the Sort module from the Operations menu so that we sort our results by publication date in descending order, and connect the filter module with the sort module, and the sort module with the Pipe Output module. Then you save the module, run it and that’s it.

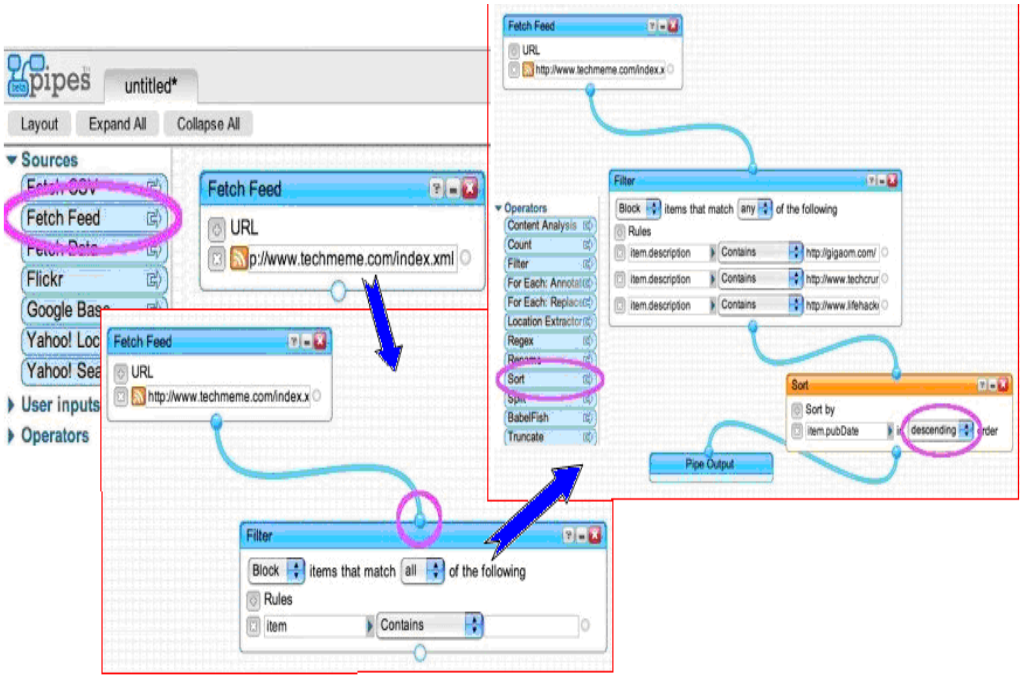


**Figure 2 - Example of creating Yahoo Pipe**

“Yahoo Pipes” is a great tool, easy to use, build, and understand pipes though a user friendly and drag-and-drop way. However, because of the lot of drag-and-drop, the user is not free at all to make what he/she wants. In other words, a user doesn’t have the flexibility that wanted from a tool. It sticks with the available libraries that the tool offers, since you can't write your own code to create pipes as other tools can do. So this tool is used only in the Yahoo environment and you can not use it in your own web site. To conclude, the tool is a great for learning the concept of pipes because it has no coding. But that's it! If you want something more advance you must go to another tool like Google or Serena.

Also a problem is that it doesn't support RDF format, and is only focuses on web feeds. In addition it depended on RSS that limits its potentials, and it only outputs RSS and Atoms, which are only able to present news items and not data. Finally, to deploy a Yahoo Pipe, you have to send the data to the Yahoo servers to get processed, which is not a preferred solution for a company's private data.

On the other hand, many consider Yahoo pipes as a milestone to the history of the internet, for the reason that it provides a service to give the web users the flavor and idea of the mashup technology, by providing an easy drag-and-drop editor. Finally we have to say that the concept and interface of our implementation of MashQL is inspired from Yahoo! Pipes.

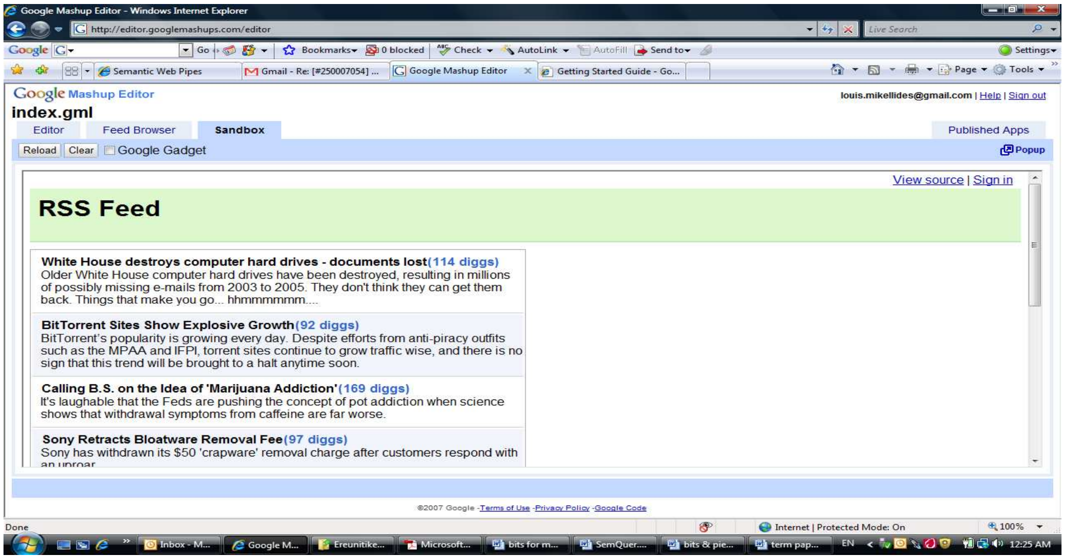
### 2.2.2 Google Mashup Editor

Google Mashups use the Google Mashup Editor (GME) as a tool to edit, compile, test and manage the application. The basic difference of the Google mashups with the others is that everything has to be implemented with code. The code is included between the tags `<gm:page>` and `</gm:page>`. One can add HTML or CSS or JavaScript code within the

<body> and </body> tags. The compilation of the application results that all GME tags are transformed into JavaScript. A tool for debugging JavaScript applications is Firebug which may shows you JavaScript errors, debug the code with breakpoints, inspect the DOM, and view variables [8]. An example of Google code is the following:

```
<gm:page title="Sample - RSS Feed" authenticate="false">
<!-- The RSS Feed application demonstrates displaying a external RSS feed
and a few of the feeds custom elements @author: GME Team  -->
  <div class="gm-app-header">
    <h1>RSS Feed</h1>
  </div>
  <gm:list id="myList" template="myListTemplate"
      data="http://www.digg.com/rss/index.xml" pagesize="10"/>
  <gm:template id="myListTemplate">
    <table class="blue-theme" style="width:50%">
        <tr repeat="true">
          <td style="padding-bottom:10px">
            <b><gm:text ref="atom:title"/>
            <span style="color:#3366cc">
              (<gm:number ref="digg:diggCount"/>&nbsp;diggs)
            </span></b>
            <br/>
            <gm:html ref="atom:summary"/>
          </td>
        </tr>
    </table>
  </gm:template>
</gm:page> [8]
```

The RSS feed example takes the RSS feeds of the digg.com and shows them in one site.



RSS Feed

**White House destroys computer hard drives - documents lost**(114 diggs)
Older White House computer hard drives have been destroyed, resulting in millions of possibly missing e-mails from 2003 to 2005. They don't think they can get them back. Things that make you go... hhmmmmmm....

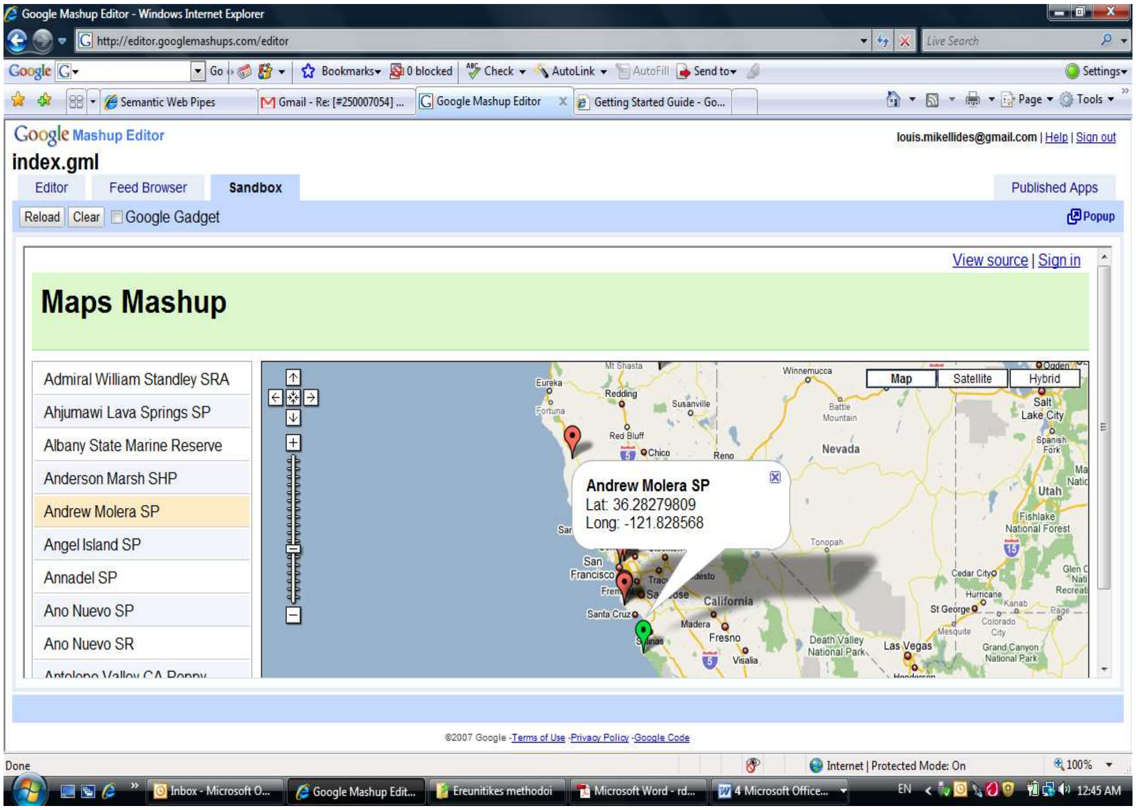
**BitTorrent Sites Show Explosive Growth**(92 diggs)
BitTorrent's popularity is growing every day. Despite efforts from anti-piracy outfits such as the MPAA and IFPI, torrent sites continue to grow traffic wise, and there is no sign that this trend will be brought to a halt anytime soon.

**Calling B.S. on the Idea of 'Marijuana Addiction'**(169 diggs)
It's laughable that the Feds are pushing the concept of pot addiction when science shows that withdrawal symptoms from caffeine are far worse.

**Sony Retracts Bloatware Removal Fee**(97 diggs)
Sony has withdrawn its $50 'crapware' removal charge after customers respond with an uproar

**Figure 3 - The output of the RSS feed code [8]**

### Second example: Map mashup



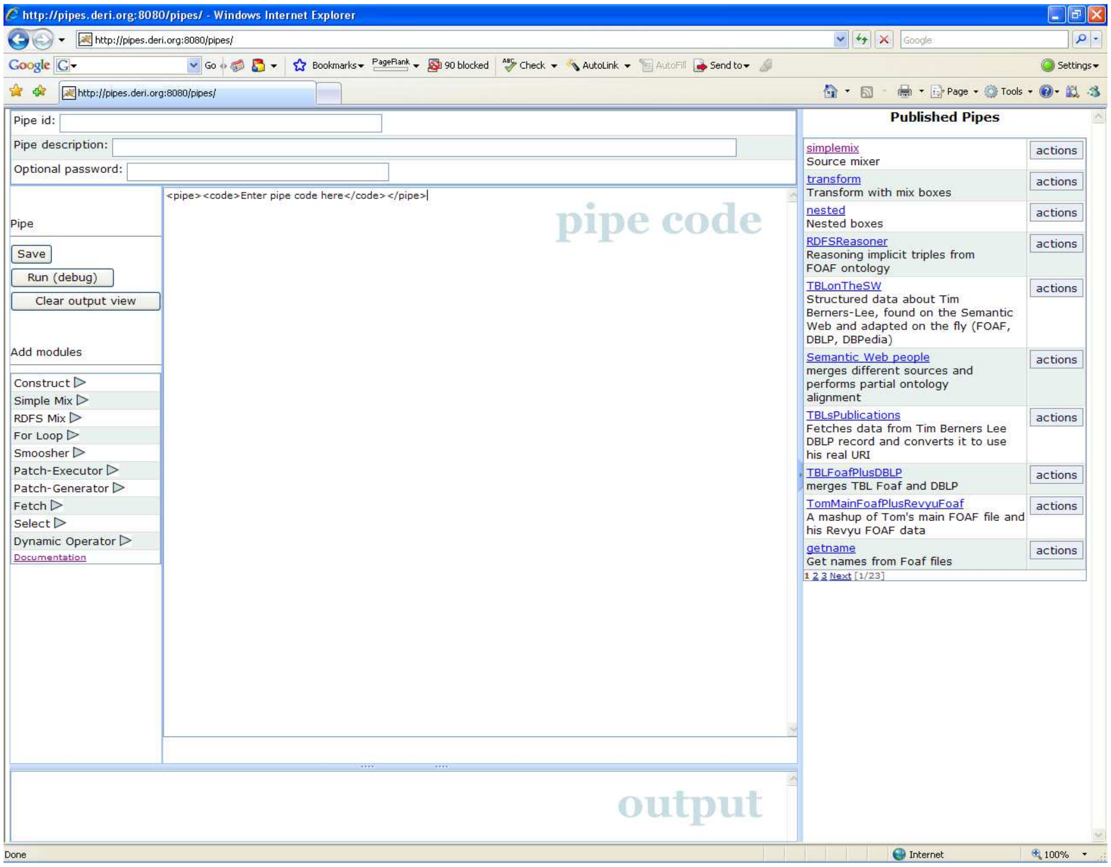
Andrew Molera SP
Lat: 36.28279809
Long: -121.828568

**Figure 4 - The result of the Map Mashup [8]**

The Maps mashup actually takes the Google map of California and the feeds of the site `http://www.mapnut.com/calstatepark.xml` that gives the position of every city on the map. [8]

### 2.2.3 Deri pipes

This pipes mode has a simple construction model, which consists of liked operators [23]. In the next figure you can see how the interface looks like.



```
<pipe><code>Enter pipe code here</code></pipe>
```

simplemix
Source mixer

transform
Transform with mix boxes

nested
Nested boxes

RDFSReasoner
Reasoning implicit triples from FOAF ontology

TBLonTheSW
Structured data about Tim Berners-Lee, found on the Semantic Web and adapted on the fly (FOAF, DBLP, DBPedia)

Semantic Web people
merges different sources and performs partial ontology alignment

TBLsPublications
Fetches data from Tim Berners Lee DBLP record and converts it to use his real URI

TBLFoafPlusDBLP
merges TBL Foaf and DBLP

TomMainFoafPlusRevyuFoaf
A mashup of Tom's main FOAF file and his Revyu FOAF data

getname
Get names from Foaf files

**Figure 5 - Deri pipes interface**

Each of the operators allows a set of unordered inputs inside the different formats and a list of ordered optional inputs. The usage of those operators is to help mashing up information from the semantic web. In addition these operators could be use for general extension in the future [23], since Deri pipes implementation is still in the development face. The interface consists of the pipes menu where you can add a ready made code to the pipe code interface, the pipe code where you can customize by writing code, the published pipes menu and the output of your code where you can see the calculated output of the pipe live when the pipe URL is fetched.

The operators included in Deri pipes have, are the following [23]:

1. The ⊎-Operator, which takes a list of RDF graphs and produce an RDF graph which is the merged of the RDF inputs.
2. The R-Operator: to express revocations of statements inside RDF. Take an RDF graph as input and applies its revocations.
3. The C- and S-Operators: which is used to CONSTRUCT and SELECT
4. The RDFS-Operator: which makes RDFS Materialization
5. The XSLT-Operator

Figure 6 shows the operators and the inputs and output by using them.

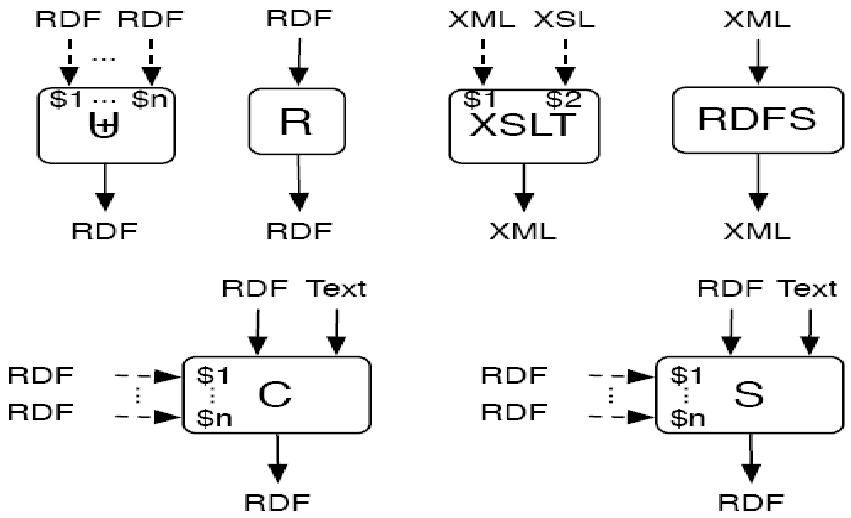


**Figure 6 - Base Operators**

The implementation of the Deri pipes covers two main aspects, which includes first the online execution engine and AJAX based pipe editor available now at http://pipes.deri.org., and secondly the DBin 2.0 client for publishing and editing the RDF data.

By using this engine, the user can create pipes so that to fetch sources from many sites, mix them by using the operators, process the RDF files and finally publish them on the web as an HTTP-retrievable RDF model or XML file.

The pipes are written in a simple XML style language. As an example of mashing up two RDF sources, the code that you should write in the pipes editor is the following [23]:

```
<merge>
        <source url="http://www.w3.org/People/Berners-Lee/card"/>
        <source url="http://g1o.net/foaf.rdf"/>
</merge>
```

The AJAX pipe editor is a simple editor, which helps the user by providing inline documentation of the operators (presented before) when inserting a component. You can also find a list of available pipes which gives the opportunity to the user to reuse those pipes. Concerning the debugger of the tool, is a simple one, highlighting any execution errors. In addition another feature of the editor is the password protection of the pipes and off course the editor is an open source one.

The DBin 2.0 cooperates with the web script which is a PHP file, to publish the RDF files on the web. The user during the publish process has to enter the URL where the script can be executed and the name of the online where the created RDF data will be stored.

We can see from this editor that operators have been used to accomplish the mashup. Also we saw that the actual construction of the pipes has to be implemented with code, just like the Google editor. However, because we want all people to be able to use the mashup editor, even and non IT people, we need an implementation that everything has to be done with drag-and-drop and not need any coding at all. Under these circumstances, we will implement our editor with an interface similar to Yahoo! Pipes, and implementation that uses the RDF framework that Google and Deri pipes uses.

Next we are going to present the technologies that our editor will be implemented with.

## 2.3 Related Technologies

### 2.3.1 RDF

In the past, the Internet was consisted nothing more than web sites, which were build in a very simple way and were basically HTML based. As time went on, businesses wanted something more structured and meaningful, so the design got harder and websites required more versatile programming specifications. Also the problem of many devices, and wireless revolution brought the problem of adaptation of web sites on any platform. To solve those challenges, languages and specifications were created, that can be adapted by any platform. That includes XML, which is a language that defined data without telling any web browser how to display them. The success of XML came from the fact that the XML file was a text file readable and understood from any browser and any system. However, computer-to-computer peer services are increasing. Therefore the need for developing a new model that will bring structures to descriptions of the web content is increasing too. Here comes the RDF (Resource Description Framework) that was written in XML that gives the platform dependability, but is more powerful than XML to give a solution to the new problems of the web, since in XML the handling of metadata in different frameworks is incompatible due to the different sets of rules and properties of representing data.

RDF is an XML-based language that can be used to describe information contained in the web resources like web pages. The RDF model consists of schemas, statements, components, containers, XML namespaces and statements about RDF statements. The next code [22] shows an example of an RDF model for a document.

```
<?xml version = "1.0"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:my="http://mymetadata.org/schema/">
        <rdf:Description about="http://www.asu.edu/namespace/">
                <my:Title>NamespaceFAQ</my:Title>
                <my:Description>
                        This is the page of FAQ for ASU namespace.
                </my:Description>
                <my:Date>2001-06-14T09:46</my:Date>
        </rdf:Description>
</rdf:RDF>
```

In this example we first specify the XML version (Line 1) that the document matches. Root element *rdf:RDF* is defined at line 2 till line 4 with two namespaces prefixes *rdf*, and *my*. Those namespaces are applicable at lines 5 till 11; with URI which are associated with the namespace declarations reference the corresponding schemas. Line 5 uses *description* to describe the resource with properties of *title* and *description* at line 6 and 7 respectively. Those are the metadata to provide more information about the resource.

RDF is the ultimate meaningful mashup language, which provides a simple way for expressing information from many sources likes spreadsheets, web pages, databases and XML files. The RDF looks like the Meta tags that are used for HTML page description [22]. It starts with XML namespace to define the page and then defines a number of elements depending of the page, to describe the page. Also, RDF lets developers to make an HTML page have double duty [1]. That is, the presentation page (the usual HTML) and the machine-readable source of structured data in RDF. It has a simple to master syntax and is easy to learn. RDF is necessary for all web designers to learn, as it has been officially adopted by the W3C as a programming standard. In addition the number of online sources that provide RDF is now estimated in the order of tens on millions, and is rapidly increasing, as more and more large databases in nowadays give RDF representations, because it provides developers with solid basis for metadata description.

It has influences from many different resources. Influences came from the web standardization community in the form of HTML metadata, XML, object-oriented programming and modeling languages and also databases.

RDF is originally designed as a metadata model, but now is used as a general method to model information by using various syntax formats. RDF model is based on the triples terminology, which the form of three expressions:

1. Subject: The subject is a resource, frequently named as the URL.
2. Predicate: denotes a resource, representing a relationship.
3. Object expressions: denotes a relationship between subject and object. The object is a resource or a string literal.

For example, if we have the sentence “My car has the color grey”, then the subject is “My car”, the predicate “has the color”, and the object “grey”. In more detailed, if we have the statement “Nicosia has a postal abbreviation NI” then in RDF format we have:

```
<urn:states:Nicosia> <http://purl.org/dc/terms/alternative> "NI" .
```

The *urn:states:Nicosia* is the URI of the resource, *http://purl.org/dc/terms/alternative* is the URI for the predicate, and *NI* is the string literal. The triple above could also be written in the standard RDF format as follows [1]:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:terms="http://purl.org/dc/terms/">
      <rdf:Description rdf:about="urn:x-states:Nicosia ">
      <terms:alternative>NI</terms:alternative>
      </rdf:Description>
</rdf:RDF>
```

If an RDF is embedded in HTML pages, it is called RDFa. The next example shows how the RDFa looks like and also shows its abilities. To start with, a typical web site with embedded RDFa data looks like the following example:

```
<p><font size="2"><div name="rlhv" id="rlhv">
  <link rel="rdf:type" href="[MusicStores: NicosiaMusicStore]"/>
    <meta property="rdfs:label"> Nicosia Store</meta> <br/>
    <a href="../../images/nicosiastore.jpg">Nicosia Store</a>
    <meta property="geo:address">Diogenous 5, 2412, Nicosia<br/></meta>
    <a href="../../images/stores.jpg">The store </a></div>
</font></p>
```

In this scenario we have a "StoreInNicosia" web page. In this sample code you can see the tags <div>, <link>, and <Meta> which include some simple facts about the web page. You also see the "rlhv" thing that has a "NicosiaMusicStore" type with the label "Nicosia Store" and address "Diogenous 5, 2412, Nicosia". Those mark-ups do not make any difference to the site viewers in the way that the page is shown.

Before we continue to the explanation of RDF, we have to say what OWL is. The Web Ontology Language (OWL) is a language that is build on top of RDF to give more semantics; thus, we can describe how we want the information to be combined together from multiple sources. In the previous example the "NicosiaMusicStore" RDF type that we have, is called a class in OWL. So, if we have information from different sources, we will have different types which will belong to different classes. For example "StoreInNicosia" page describes members of "NicosiaMusicStore". We can also define new classes for anything that we want to implement like other stores in Nicosia. To understand what we have said so far, see the next example [1]:

```
<owl:Class rdf:about="#MyStores"/>
<owl:Class rdf:about="#MusicStore Entertainment">
      <rdfs:subClassOf rdf:resource="#MyStores"/>
</owl:Class>
<owl:Class rdf:about="#GamesStore">
```

```
      <rdfs:subClassOf rdf:resource="#MyStores"/>
</owl:Class>
<owl:Class rdf:about="#BookStores">
      <rdfs:subClassOf rdf:resource="#MyStores"/>
</owl:Class>
<owl:Class rdf:ID="Adventures">
      <rdfs:subClassOf rdf:resource="#GamesStore"/>
</owl:Class>
<owl:Class rdf:ID="Sports">
      <rdfs:subClassOf rdf:resource="#GamesStore"/>
</owl:Class>
<owl:Class rdf:ID="RPG">
      <rdfs:subClassOf rdf:resource="#GamesStore"/>
</owl:Class>
<owl:Class rdf:ID="Pop">
      <rdfs:subClassOf rdf:resource="#MusicStore"/>
</owl:Class>
```

Here we use OWL to combine all those things together. As you can see in the previous example, all we wanted to do is to combine the stores of an imaginary company, call "MyStores". So the "MyStores" is a new class. This class has some ranges of stores like the "MusicStore", "GamesStore", and "BookStore", which are subclasses of "MyStores" class. Finally we have "Adventures", "Sports", and "RPG" which are subclasses of "GamesStore", and "Pop" which is a subclass of "MusicStore". To sum up, as [1] says, this structure looks like a concept-oriented style programming that helps you to organize in an efficient way your data. So by using RDF in an OWL language can help you to display your mashup at any level of the tree.

RDF provides a rich data model where the entities and relationships can be described in details. The relationships in the RDF framework are class objects, which mean that the relationship between the objects may be created arbitrarily and stored separately from the objects. This makes RDF suitable for dynamically changing, shared and distributed nature of web. Also RDF's simple data model and the ability of modeling data which differ from resource to resource, led to its increasing usage in applications unrelated to semantic web activity. It is easy to write, flexible, and without constrains. However RDF has been criticized that the XML syntax for RDF is verbose and the triplet (subject, predicate, object expressions) notation is not expressive enough.

Our implementation of MashQL uses RDF resources as the input sources to query from.

### 2.3.2 SPARQL

SPARQL is the formal language to make queries using the RDF data from the web pages that support RDF. SPARQL stands for Simple Protocol and RDF Query Language. The language was standardized on 15th January 2008 by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium. This language could work for any of the data sources that could be mapped into RDF with result to query across various data sources stored or view as RDF via middleware. Currently SPARQL defines only read-only queries, so you can use it only for read and compare operation for graphs. Also note that the appearance of SPARQL brought to the light the RDF concept which was not so well-known so far, because now you have a way to query RDF resources. This language is more likely SQL language so an example will help us to understand better the query syntax of how to retrieve variables by using SPARQL:

```
PREFIX dc: <http :// purl.org/dc/elements /1.1/ >
PREFIX ns: <http :// example.org/ns#>
SELECT ?title ?price
WHERE { ?x ns:price ?price .
         FILTER (?price < 30) .
         ?x dc:title ?title }
ORDER BY ?title
LIMIT 5
OFFSET 5
```

By executing this query, we want to retrieve the price and title of the items that have price less than 30. As you can see, this statement consist of the PREFIX which allows to define namespace prefixes, SELECT, WHERE clause and FILTER conditions. Filtering includes logical operation like <, <=, >, >=, = and others. Also some other optionally statement could be use like ORDER BY for sorting, LIMIT to limit the number of returned variables, and OFFSET for pagination. In addition all variables as you can notice start with a question mark "?" or "$".

SPARQL supports four kinds of query results that can give any solution to the queries:

1. SELECT: This statement returns the variables and variables' binding directly as a result. For those familiar with SQL, SPARQL's SELECT * returns all the variables. SELECT statement consists of three parts and an option set of prefixes like the next example:

   ```
   PREFIX (list of the optional prefixes)
   SELECT (the selection of the wanted variables)
   FROM (specify the dataset to retrieve the variables)
   WHERE (any conditions)
   ```
2. CONSTRUCT: returns a single RDF graph.
3. ASK: return a simple Boolean to test whether a query matches or not (has solution or not.
4. DESCRIBE: return a single result which contains data about resources.

**The Importance of SPARQL:**

Think about a scenario that a large book store receives thousands of publishers, with each one having the details of the book and personal details in an RDF format. What will happen next? Well, RDF resources could be seen as a table like the database [25] which has its tables, due to the semantic format of the RDF. So the most appropriate way to retrieve those data with an easy way is to use the SPARQL language. SPARQL offers to the query writers semantic interoperability, better search and retrieval than typical web searching, and data integration. Also think about an extension of this scenario. If we have publication from different sites like Google Scholar, CiteSeer, etc., and those are in an RDF source, then by using SPARQL we can mash up the contents into a single site easily [25].

**The problems of SPARQL:**

Even if SPARQL brings to the light new different and useful scenarios concerning the usage of RDF, like semantic interoperability and data integration, it still has some problems, since the very little time that has standardized by W3C.

First of all, as [25] says, learning such language requires some basic knowledge in RDF and its structure. So it’s not an easy task for the IT people especially that the language came to the light recently. However, from our little experience while studying these technologies we found them very interesting, challenging and seem easy-to-learn. This is because we already have experience in using databases, SQL and XML; things that most IT community is not so familiar with this.

Secondly, [25] continues, that the formulation of a query requires from the writer of the query to understand first of all, the structure of the RDF that will execute the query and get the results. However, this is done by seeing the source of the particular RDF structure and then makes the parsing. Taking into account that a typical RDF resource could be many lines of code, this parsing is not practical for the query writer at all. Also SPARQL queries, which are the only way to filter and aggregate data from the sources, are not adequate for taking subjective revocations, if we don’t know in an early stage which subjective statements a graph may revoke.

In our implementation of MashQL, we have used a SPARQL-like query language (Oracle's SPARQL) to query from the RDF recourse.

### 2.3.3 JSP

The presentation of the MashQL editor is done by using JSPs. The JavaServer Pages (JSP) is an extension of the Java Servlets Technology that is permitting the combination of HTML/XML with Java scriptlets to produce pages with dynamic content. The current version is 2.0. The specification can be found at: http://www.jcp.org/en/jsr/detail?id=152. In addition, [12] says about JavaServer Pages that, *JSPs are, in essence, HTML files with the extension JSP, which also contain special JSP tags. To deliver a JSP container reads the JSP and uses it as a template to create and compile a servlet. The JSP container then invokes the temporary servlet, which creates the Web page that is sent back to the user. As with Servlets, JSPs have both a request and a response, and similarly have access to all of the request and context/Web application data.*

Java was developed by Sun Microsystems. The language is controlled by them. For more details, the URL for JSP technology can be found at: http://java.sun.com/products/jsp/. JavaServer Pages (JSP) according to [14]:

- *Use HTML and XML tags to design the page and JSP scriptlet tags to call Java programs that generate dynamic content.*
- *Use JavaBeans, JDBC objects, Enterprise Java Beans (EJB) and Remote Method Invocation (RMI) objects which are reusable components that are invoked by scriptlets.*

### Why Java Server Pages?

The task to write server-side Java code and designing great Web pages are two separated disciplines that are handled by two different people or groups of people. Now, the challenge is to find a way to separate the presentation of the Web site, that is, the visual design, from the programming logic, that determines how dynamic information arrives on the web page. The solution of JavaServer Pages (JSP) is Sun's answer to that challenge.

The purpose of JSPs in the MashQL implementation is to retrieve the entire editor's requests, interact with the database, get the response back, and send the results back to the editor for further process.

### Why Java

Using Java language is better than any other language. There are four main reasons why to choose Java:

- **Java is a productive language**. Although it owes much to C++, it is by comparison a more forgiving language and simpler. The absence of pointers (and attendant pointer match) helps to make code more readable. Also the automatic garbage collector frees the developer from memory de-allocation worries.
- **Java has application programming (APIs)** that makes the language very useful for Web development. Not the least of these are the Servlet and JavaServer Pages API, but other aspects of the language make developing a complete Web application that much easier. You can use Java Database Connectivity (JDBC), to connect to a back-end database and/or use the graphics library to create on-the-fly images for your web site.

- **Java has wide industry support.** There is a lot that have been made about bickering over the future of Java standards. And the lack of standards on the client side has clearly hurt the acceptance of Java applets. Meanwhile, there has been massive industry investment in Java technology. Many large corporations like Oracle and IBM, have invested in and supported many Java projects, which many of them server related.
- **Java is scalable and lightweight.** To create a massive successful Web site, means a lot of traffic. Although Java does carry a slight performance and resource penalty compared to languages like C++, it is also design to thread as lightly as possible, mainly for the reason that Java does not have to load new code to execute each request. The server simply starts another thread, which makes use of code that is already loaded.

In addition [12] says that, *despite the many solutions available, many developers are taking up Servlets and JSP because of Java's unique combination of productivity, portability, power and capability.*

In MashQL we have use Java for the parser mechanism (section 4.4.3) which translates the MashQL query design into SPARQL query for execution or debug purpose.

**Why JSP rather than other scripting languages**

JavaServer Pages has many features that have contributed to its acceptance as an attractive alternative to other scripting languages. As [6] refers, JSP is:

- ***Platform dependence.*** *The use of JSP adds versatility to a Web application by enabling its execution on any computer.*

- ***Enhanced performance.*** *The compilation process in JSP produces faster results or output.*
- ***Separation of logic from display.*** *The use of JSP permits the HTML-specific static content and a mixture of HTML, Java, and JSP-specific dynamic content to be placed in separate files.*
- ***Ease of development.*** *The use of JSP eliminates the need for high-level technical expertise, thereby helping Web developers, designers, content creators, and content managers to work together and develop Java-based applications in less time and with less effort.*
- ***Ease to use.*** *All JSP applications run major servers and operation systems, including Microsoft IIS, Netscape Enterprise Server, iPlanet Web Server, and Apache Web Server.*

### Comparison of JSP and ASP, ColdFusion and PHP

**Portability:** JSP is a portable application, is flexible because it is machine, operating system and server independent. As a result, such applications can be shared between developers. The “Write Once, Run Anywhere” phenomenon offer JSP application a unique, platform-independent reusability. For that reason, JSP can be deployed on several browsers, servers, and tools. On the contrary, ASP is a Microsoft product that can be deployed mainly on Windows NT, and is therefore restricted in its scope.

[6] adds about JSP, *considering its support for the Apache Web Server, which hosts a major percentage of the world’s Web sites, it is merely a matter of time before JSP is also to use the strong UNIX platform.* In addition [10] says that *JSP is portable to other operating*

*systems and Web servers; you aren't locked into Windows and IIS. Even if ASP.NET succeeds in addressing the problem of developing server-side code with VBScript, you cannot expect to use ASP on multiple servers and operating systems.*

**Performance:** The compilation or translation phase of a JSP page is such that once generated, the servlet class for a particular JSP page remains in the server memory. As a result, a subsequent request for the same JSP page does not require reloading and recompiling. The response time of a JSP page is therefore very short. On the other hand, a first-time or subsequent request for an ASP page requires continuous recompilation, which increases response time. Also [10] adds that *ASP is a competing technology from Microsoft. The advantages of JSP are twofold. First, the dynamic part is written in Java, not VBScript or another ASP-specific language, so JSP is more powerful and better suited to complex applications that require reusable components.*

**Deployment:** JSP components (JavaBeans and custom tags), help separate the pages design from the programming logic of a Web page. This kind of approach to application development, gives more power to the individual, skill-based profile of Web designers and developers. If you visualize the extensibility and flexibility that can be added to such applications, the designers concentrate on static content generation, and developers program for generation of the dynamic content. Even if ASP is depended on Microsoft's COM/DCOM components, developing this kind of applications is quite a complex process as compared to the easy development and deployment of components (JavaBeans) using Java.

The same arguments stand and when comparing JSP to the current version of ColdFusion. With JSP you can use Java for the "real code" and are not tied to a particular server product.

Likewise PHP is a free, open-source HTML-embedded scripting language, which is somewhat similar to both JSP and ASP. One advantage of JSP is that the dynamic part is written in Java, which already has an extensive API for networking, distributed objects and database access, whereas PHP requires learning an entirely new, less widely used language. A second advantage is that JSP is much more widely supported by tool and server vendors than PHP is.

### 2.3.4 AJAX

AJAX is an acronym that stands for Asynchronous Javascript and XML. The interesting thing about AJAX is that the XML component is actually unnecessary, or rather optional. The important component is asynchronous javascript - this is the meat of how web2.0 applications work and the XML component is just one possible format for sending and receiving the additional data requests to the server. However, since this processing happens in the background and is invisible to the end user, you can actually build an application using any format for the data requests that you wish.

The key to implementing an AJAX application is in the XMLHTTP Object. The XMLHTTP objects exists in many forms, both server-side and client-side, and the purpose of it, is to allow retrieval and processing of external web pages from within the coding application. Since we are trying to build a client-side application, and since Javascript is the most widely available scripting application for web browsers, AJAX is the ideal implementation, and a good cross-browser Javascript code for instantiating the XMLHTTP Object [33].

MashQL editor make use of the AJAX objects in many mechanisms that will be described in chapter 4, where the design and implementation of the editor is analytically explained. Just to give you some information about, the AJAX has been applied when requesting data from the database server, when executing or debugging a pipe or query component, and when loading a pipe. The reason is that everything has to be transparent and not force the user to wait for the results to come, since everything is done asynchronously.

### 2.3.5 JavaScript

JavaScript is a scripting language most often used for client-side web development. It is a small, lightweight, object-oriented, cross-platform scripting language. A lot of confusion is made when referring to it. The basic syntax is intentionally similar to both Java and C++ to reduce the number of new concepts required to learn the language. Language constructs, such as if statements, for and while loops, and switch and try ... catch blocks function are nearly the same as in these languages. Objects are created programmatically in JavaScript, by attaching methods and properties to otherwise empty objects at run time, as opposed to the syntactic class definitions common in compiled languages like C++ and Java. Once an object has been constructed it can be used as a blueprint for creating similar objects.

Someone who hasn’t heard of JavaScript the first that comes in mind is that is has to do with Java language. However JavaScript is essentially unrelated to the Java programming language. An example of Java script that determines the client browser type is shown below.

```
<script type=text/javascript>
if (navigator.appName.toUpperCase().match(/MICROSOFT INTERNET EXPLORER/) != null)
  document.write("You are using IE.");
```

```
if (navigator.appName.toUpperCase().match(/NETSCAPE/) != null)
  document.write("You are using NS.");
</script>
```

JavaScript's dynamic capabilities include runtime object construction, variable parameter lists, function variables, dynamic script creation and other functions. For that reason JavaScript is used in the MashQL editor. The functions that implemented by the power that JavaScript offers includes the following

- Dynamic module creation, the resizable panels
- Call to database though AJAX
- Drag-and-drop of modules
- Movement of the modules
- Validation of the editor components (text fields, combo boxes etc.)
- Alert messages

### 2.3.6 XML

The Extensible Markup Language (XML) is a set of rules that encodes documents electronically. It is produced by W3C and is an open standard. The goal of XML is to emphasize the simplicity, usability, and generality though its design over the internet. The format is actually a text with support for all the languages of the world via Unicode. The advantage of the XML is that it can be accessed by a variety of programming interfaces, and also several schema systems have been design to support in the definition of languages based on XML. As from 2009, hundreds of XML-based languages have been developed, [32] including RSS, Atom, SOAP, and XHTML. XML has become the default file format for most office-productivity tools, including Microsoft Office, OpenOffice.org, and Apple's iWork.

An XML document consists of markup, content, tags, elements, attribute, and XML declaration at the beginning of the document. The markup are all the strings which begin with the character “<” and end with the “>”, or begin with “&” and end with “;”. Everything that does not included to this is content. The markup could be named as a tag. There are three types of tags: The start-tags e.g. <section>, end-tags e.g. </section>, and empty-tags e.g. <section/>. An element is a logical component in the document which begins with start-tag and ends with a matching end-tag, or consists only of an empty-element tag. The characters between start/end-tag are called the element’s content e.g. <section>1</section>. An attribute is a name/value pair that exists within a start-tag or empty-tag. For example in <section number=”4”>4.1</section>, the attribute is “number” and the value is “3”. Note that the attribute must be unique inside one start-tag or empty-tag. The declaration is some information about the XML document which always must begin with. For example <?xml version="1.0" encoding="UTF-8" ?> is the declaration of an XML document. An example of a complete XML document is the following:

```
<?xml version="1.0" encoding='UTF-8'?>
<painting>
  <img src="madonna.jpg" alt='Foligno Madonna, by Raphael'/>
  <caption>This is Raphael's "Foligno" Madonna, painted in
  <date>1511</date>-<date>1512</date>.</caption>
</painting>
```

In this example we have the declaration at the beginning and five elements following: painting, img, caption, and two dates. The date elements are children of caption, which is a child of the root element painting. The img tag has two attributes, the src and alt.

The most common usage of XML is to interchange/send data over the Internet. For this reason there are some rules included in RFC 2023 that defines the types “application/xml” and “text/xml” [32].

In our editor we have use both types. The “application/xml” type is used in the case that we want to call the parser java module to do the transformation of the MashQL query to a SPARQL query. The “text/xml” type is used when loading the user’s pipes since we want to retrieve the whole xml document back and then parse it and get the wanted information of the pipes to display to the user. These mechanisms will be described in chapter 4.

### 2.3.7 Oracle 11g.

We chose Oracle 11g for the implementation of MashQL, as a back-end, because of its support for native RDF queries and storage. In addition Oracle will be more suitable to our implementation because it allows for future performance tuning and maintenance of the queries. On the other hand if we use JENA SPARQL query libraries the queries will be executed inside the browser using the machine resources which are limited. So nothing much can be done for better performance and quicker queries. To sum up Oracle Semantic Technology leave as a window open for better performance in our next versions of MashQL.

Oracle Database enables to store semantic data and ontologies, to query semantic data and to perform ontology-assisted query of enterprise relational data, and to use supplied or user-defined inferencing to expand the power of querying on semantic data [24]. To load semantic data, bulk loading is the most efficient approach that Oracle refers to. Semantic data and ontologies can be queried, and you can also perform ontology-assisted queries of semantic and traditional relational data to find semantic relationships. In addition to its formal semantics, semantic data has a simple data structure that is effectively modeled using a directed graph. The metadata statements are represented as triples: nodes are used to represent two parts of the triple, and the third part is represented by a directed link that describes the relationship between the nodes. The triples are stored in a semantic data network. Statements are expressed in triples: {subject or resource, predicate or property, object or value} (see section 2.3.1).

In our implementation to query semantic data, we use the SEM_MATCH table function. This function has the following attributes:

```
SEM_MATCH(
query VARCHAR2,
models SEM_MODELS,
rulebases SEM_RULEBASES,
aliases SEM_ALIASES,
filter VARCHAR2,
index_status VARCHAR2,
options VARCHAR2
) RETURN ANYDATASET;
```

The query attribute is required. The other attributes are optional (that is, each can have a null value).

The query attribute is a string literal (or concatenation of string literals) with one or more triple patterns, usually containing variables. A triple pattern is a triple of atoms enclosed in parentheses. Each atom can be a variable (e.g. ?x), a qualified name (e.g. rdf:type) that is expanded based on the default namespaces and the value of the aliases attribute, or a full URI (e.g. <http://www.example.org/family/Male>). Moreover, the third atom can be a numeric literal (e.g. 3.14), or a plain literal (e.g. "Herman"), or a language-tagged plain literal (e.g. "Herman"@en), or a typed literal (e.g. "123"^^xsd:int).

For example, the following query attribute specifies three triple patterns to find grandfathers (that is, grandparents who are also male) and the height of each of their grandchildren: '(?x :grandParentOf ?y) (?x rdf:type :Male) (?y :height ?h)'. Also note that the oracle version used is 1.0.0.6 which accepts patterns like the previous example. The next version will have the new patch 1.0.0.7 which accepts patterns as a graph like: '{?x :grandParentOf ?y} {x rdf:type :Male}{?y :height ?h}'. The difference is that in the newer version the triples may not be in parenthesis but in braces, indicating that the patterns denote a graph.

The models attribute identifies the model or models to use. Its data type is SEM_MODELS, which has the following definition: TABLE OF VARCHAR2(25). Every RDF resource has its own model which can be as a model parameter in the SEM_MODELS attribute. Also note that you can add as many models as you want as a parameter if and only if each model refers only ones in there.

The other attributes in SEM_MATCH are not necessary for our implementation. So an example of a query from MashQL to Oracle semantic technology might be the following:

```
SELECT P1 FROM TABLE(SEM_MATCH('(?S ?P1 ?O1)', SEM_Models('GELJMLUZ'), null,
                                null, null))
 where P1 like '%%'
 AND P1 IN (
  SELECT P FROM TABLE(SEM_MATCH('(?S ?P ?O)', SEM_Models('GELJMLUZ'), null,
                      null, null))
  WHERE P = 'http://data.semanticweb.org/ns/swc/ontology#hasLocation'
  AND O = 'http://data.semanticweb.org/conference/eswc/2007/location-igls'
 )
 and rownum <= 100
 group by P1  order by P1 ASC
```

In this example the query in natural language is the following: Select all predicated from GELJMLUZ model (RFD resource) having a subject S and object O1, where the predicate P1 is like anything and P1 exists in all P-predicated having GELJMLUZ model as source and that P1 have a subject S and an object O, and also have the conditions that P is equal to http://data.semanticweb.org/ns/swc/ontology#hasLocation and also that O is equal to http://data.semanticweb.org/conference/eswc/2007/location-igls. Also the query will be made for the first 100 rows. The results will be grouped by P1 and ordered by ascending order of P1. As we can observe the syntax is very similar to SQL, except from the SEM_MATCH function with its attributes. The relative SQL syntax for this query is:

```
SELECT {something}
FROM {table}
WHERE {conditions}
GROUP BY
ORDER BY
```

whereas the semantic query is

```
SELECT {something}
FROM TABLE (SEM_MATCH(…..),SEM_MODEL(….),null,null,null)
WHERE {conditions}
GROUP BY
ORDER BY
```

# Chapter 3

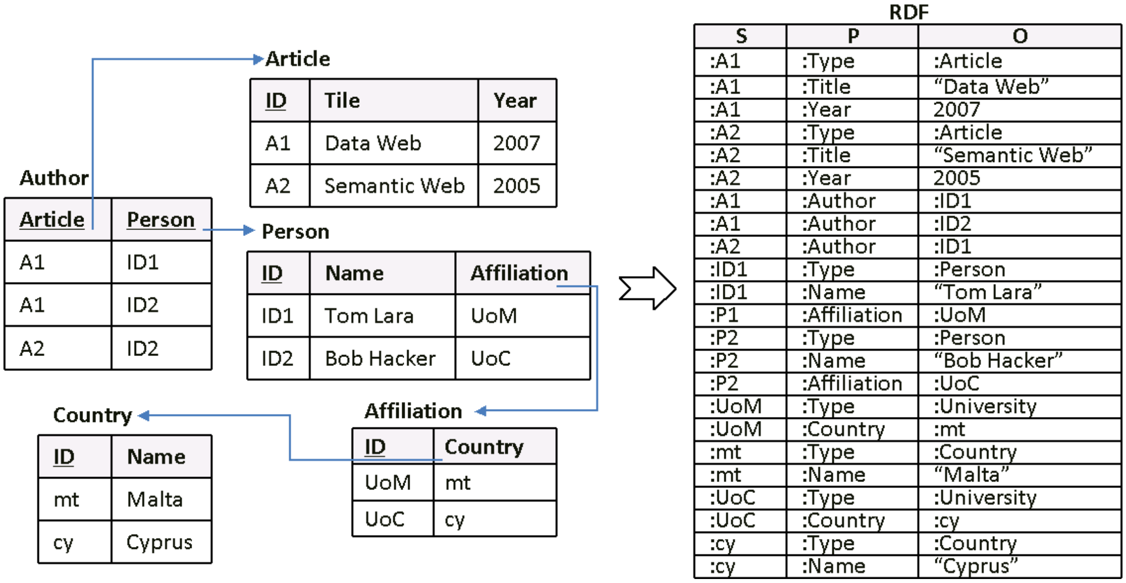
## The Definition of MashQL

In this section we will concentrate on the definition of the data model which is similar to the RDF model, syntax and semantics, following the one of SPARQL and their rules. Also we will focus on query formulation of MashQL explain it step-by-step, to show that the complexity and responsibility of understanding a data source is to the query editor site and not to the user.

### 3.1 The Data Model - RDF

The data model of MashQL follows the model of RDF with few changes. First we shall state that in MashQL we assume the queried dataset is structured as a direct labeled graph, and that it is a set of triples including <Subject, Predicate, Object> (review in section 2.3.1). That is, each triple can be expressed as (?S ?P ?O), where the S and P can only be an identifier (URL or a key), and O can be a unique identifier or a literal. The changes that this syntax has compared to the RDF model, is that the identifier is allowed to have also the form of a key and not only a URL. In other words the identifier with a key in MashQL is weaker than the identifier with URI. The reason why this change is allowed in MashQL is that this will simplify the use of MashQL for querying databases and also a relational database can be easily mapped to this primitive data model [17].

In Oracle database an RDF dataset could be stored into a table by using the semantic technology that oracle provides. In figure 7 we illustrate an example of how the RDF table is created inside Oracle database [17].



Article

| ID | Tile | Year |
|---|---|---|
| A1 | Data Web | 2007 |
| A2 | Semantic Web | 2005 |

Author

| Article | Person |
|---|---|
| A1 | ID1 |
| A1 | ID2 |
| A2 | ID2 |

Person

| ID | Name | Affiliation |
|---|---|---|
| ID1 | Tom Lara | UoM |
| ID2 | Bob Hacker | UoC |

Country

| ID | Name |
|---|---|
| mt | Malta |
| cy | Cyprus |

Affiliation

| ID | Country |
|---|---|
| UoM | mt |
| UoC | cy |

RDF

| S | P | O |
|---|---|---|
| :A1 | :Type | :Article |
| :A1 | :Title | “Data Web” |
| :A1 | :Year | 2007 |
| :A2 | :Type | :Article |
| :A2 | :Title | “Semantic Web” |
| :A2 | :Year | 2005 |
| :A1 | :Author | :ID1 |
| :A1 | :Author | :ID2 |
| :A2 | :Author | :ID1 |
| :ID1 | :Type | :Person |
| :ID1 | :Name | “Tom Lara” |
| :P1 | :Affiliation | :UoM |
| :P2 | :Type | :Person |
| :P2 | :Name | “Bob Hacker” |
| :P2 | :Affiliation | :UoC |
| :UoM | :Type | :University |
| :UoM | :Country | :mt |
| :mt | :Type | :Country |
| :mt | :Name | “Malta” |
| :UoC | :Type | :University |
| :UoC | :Country | :cy |
| :cy | :Type | :Country |
| :cy | :Name | “Cyprus” |

**Figure 7 - Mapping a database to RDF**

For this figure the primary keys e.g. A1, A2, mt, cy, ID1 etc, that belongs to a relational database, are presented as S (subjects) in an RDF table. The column label in relational database is presented as P (predicate) in RDF, and data-entry as the O (object) in the RDF table. So from this figure we can observe that not any schema has to be known from the user to query an RDF table opposed to the one on the relational database.

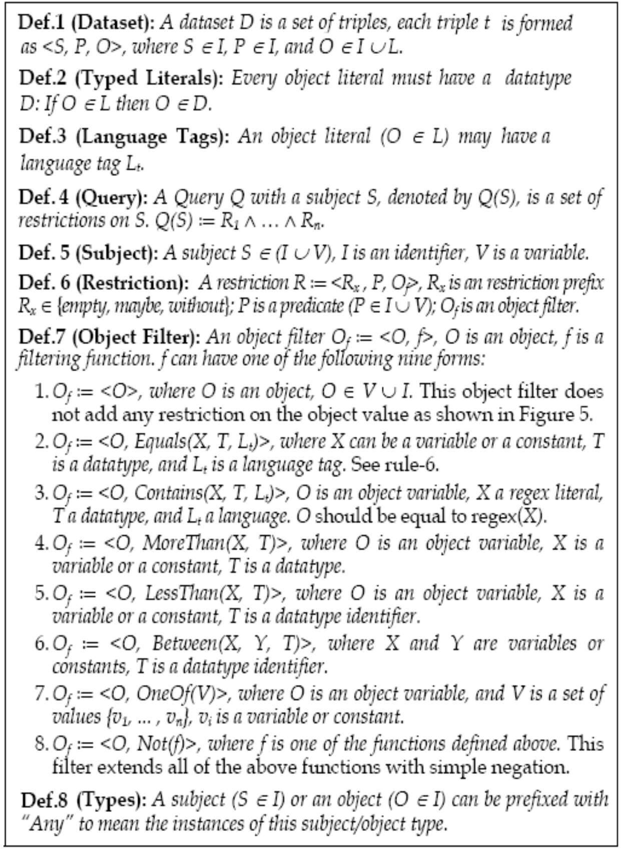
The datatypes of each object are handled automatically by the oracle. In this state we have to mention that MashQL does not support language tags in this version such as “Person”@En, “Ατομο”@Gr.

## 3.2 Syntax and Semantics

The MashQL query follows the semantics of SPARQL. This is because a MashQL query cannot be executed by itself, so it has to be translated to SPARQL, and the MashQL query's SPARQL can then be executed. Note that the SPARQL to be executed in oracle's semantic technology, is has to be translated into oracle's SPARQL which has a slightly difference from pure SPARQL. However, this is done automatically by the editor, so for now we assume that everything is translated and executed in pure SPARQL.

In table 1 as [18] refers we present the formal definition of the MashQL constructs and in table 2 we present their SPARQL interpretation.

**Table 1 - The formal definition of MashQL**



**Def.1 (Dataset):** *A dataset D is a set of triples, each triple t is formed as <S, P, O>, where* $S \in I$, $P \in I$, *and* $O \in I \cup L$.

**Def.2 (Typed Literals):** *Every object literal must have a datatype D: If* $O \in L$ *then* $O \in D$.

**Def.3 (Language Tags):** *An object literal* ($O \in L$) *may have a language tag* $L_t$.

**Def. 4 (Query):** *A Query Q with a subject S, denoted by Q(S), is a set of restrictions on S.* $Q(S) \coloneqq R_1 \wedge \ldots \wedge R_n$.

**Def. 5 (Subject):** *A subject* $S \in (I \cup V)$, *I is an identifier, V is a variable.*

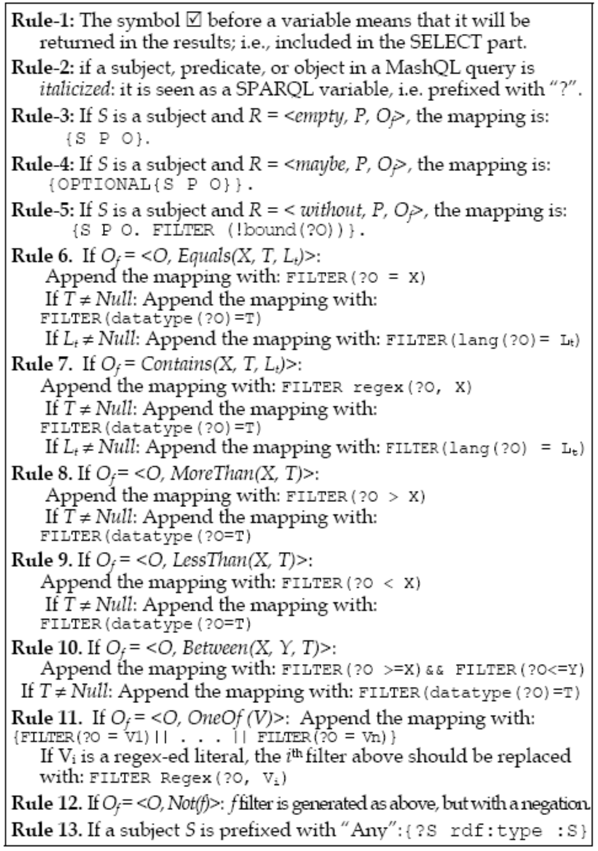
**Def. 6 (Restriction):** *A restriction* $R \coloneqq <R_x, P, O_f>$, $R_x$ *is an restriction prefix* $R_x \in$ *{empty, maybe, without}; P is a predicate* ($P \in I \cup V$); $O_f$ *is an object filter.*

**Def.7 (Object Filter):** *An object filter* $O_f \coloneqq <O, f>$, *O is an object, f is a filtering function. f can have one of the following nine forms:*

1. $O_f \coloneqq <O>$, *where O is an object,* $O \in V \cup I$. This object filter does not add any restriction on the object value as shown in Figure 5.
2. $O_f \coloneqq <O, Equals(X, T, L_t)>$, *where X can be a variable or a constant, T is a datatype, and* $L_t$ *is a language tag.* See rule-6.
3. $O_f \coloneqq <O, Contains(X, T, L_t)>$, *O is an object variable, X a regex literal, T a datatype, and* $L_t$ *a language. O* should be equal to regex(*X*).
4. $O_f \coloneqq <O, MoreThan(X, T)>$, *where O is an object variable, X is a variable or a constant, T is a datatype.*
5. $O_f \coloneqq <O, LessThan(X, T)>$, *where O is an object variable, X is a variable or a constant, T is a datatype identifier.*
6. $O_f \coloneqq <O, Between(X, Y, T)>$, *where X and Y are variables or constants, T is a datatype identifier.*
7. $O_f \coloneqq <O, OneOf(V)>$, *where O is an object variable, and V is a set of values* $\{v_1, \ldots, v_n\}$, $v_i$ *is a variable or constant.*
8. $O_f \coloneqq <O, Not(f)>$, *where f is one of the functions defined above.* This filter extends all of the above functions with simple negation.

**Def.8 (Types):** *A subject* ($S \in I$) *or an object* ($O \in I$) *can be prefixed with "Any" to mean the instances of this subject/object type.*

**Table 2 - MashQL to SPARQL mapping rules**



**Rule-1:** The symbol ☑ before a variable means that it will be returned in the results; i.e., included in the SELECT part.

**Rule-2:** if a subject, predicate, or object in a MashQL query is *italicized*: it is seen as a SPARQL variable, i.e. prefixed with "?".

**Rule-3:** If *S* is a subject and *R* = <*empty, P, O*$_f$>, the mapping is:
`{S P O}.`

**Rule-4:** If *S* is a subject and *R* = <*maybe, P, O*$_f$>, the mapping is:
`{OPTIONAL{S P O}}.`

**Rule-5:** If *S* is a subject and *R* = < *without, P, O*$_f$>, the mapping is:
`{S P O. FILTER (!bound(?O))}.`

**Rule 6.** If $O_f = <O, Equals(X, T, L_t)>$:
Append the mapping with: `FILTER(?O = X)`
If $T \neq Null$: Append the mapping with:
`FILTER(datatype(?O)=T)`
If $L_t \neq Null$: Append the mapping with: `FILTER(lang(?O)= Lt)`

**Rule 7.** If $O_f = Contains(X, T, L_t)>$:
Append the mapping with: `FILTER regex(?O, X)`
If $T \neq Null$: Append the mapping with:
`FILTER(datatype(?O)=T)`
If $L_t \neq Null$: Append the mapping with: `FILTER(lang(?O) = Lt)`

**Rule 8.** If $O_f = <O, MoreThan(X, T)>$:
Append the mapping with: `FILTER(?O > X)`
If $T \neq Null$: Append the mapping with:
`FILTER(datatype(?O=T)`

**Rule 9.** If $O_f = <O, LessThan(X, T)>$:
Append the mapping with: `FILTER(?O < X)`
If $T \neq Null$: Append the mapping with:
`FILTER(datatype(?O=T)`

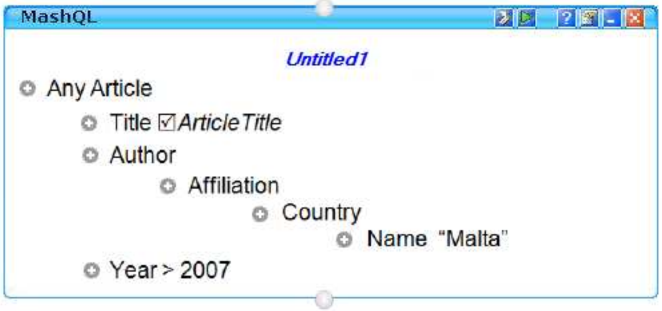
**Rule 10.** If $O_f = <O, Between(X, Y, T)>$:
Append the mapping with: `FILTER(?O >=X)&& FILTER(?O<=Y)`
If $T \neq Null$: Append the mapping with: `FILTER(datatype(?O)=T)`

**Rule 11.** If $O_f = <O, OneOf(V)>$: Append the mapping with:
`{FILTER(?O = V1)|| . . . || FILTER(?O = Vn)}`
If $V_i$ is a regex-ed literal, the *i*th filter above should be replaced with: `FILTER Regex(?O, Vi)`

**Rule 12.** If $O_f = <O, Not(f)>$: *f* filter is generated as above, but with a negation.

**Rule 13.** If a subject *S* is prefixed with "Any":`{?S rdf:type :S}`

First of all in each MashQL query Q can be view as tree. The root of is the query subject, the Q(S), which is the one that we are looking for in a query (Def 4 in Table 1). To understand this better we give an example in figure 8, with the dataset which is available in figure's 7 RDF table.



MashQL

*Untitled1*

- Any Article
    - Title ☑*ArticleTitle*
    - Author
        - Affiliation
            - Country
                - Name "Malta"
    - Year > 2007

```
SELECT ?ArticleTitle
FORM < http://localhost.example2#>
WHERE { ?X :Type :Article.
          ?X :Title ?ArticleTitle.
          ?X :Author ?X1.
          ?X1 :Address ?X2.
          ?X2 :Country ?X3.
          ?X3 :Name ?X4.
          ?x :Year ?X5.
          FILTER regex(?X4, "Malta"))
          FILTER (?X5 > 2007)}
```

**Figure 8 - Query paths and sub-trees in MashQL**

The Q(S) in this example is the "Article". The S can be an identifier (URL or a key) I or a variable (user's entry) V (Def 5 in Table 1). In the case the S or P or O is a variable V, then it is prefixed with "?" in the SPARQL query (Rule 2 in Table 2). Further on, each branch in tree with root S is called a restriction R, on a particular property P of that subject. Each of the branches can be further expanded to allow sub-trees, called query paths. (Note that there is no limitation on the level of expansion, however deeper the level is more time-costly for retrieving atom values in their lists). In the case that we have a sub-tree, the value of property is seen as the subject of its sub tree, so that a user can navigate though the underlying dataset and built complex queries [17]. In the case that any of the atom in a restriction has the check-box checked, this means that this will be a return value in the query (Rule 1 in Table 2).

Each of the restrictions can be prefixed. If a restriction has no prefix (Def 6 in Table 1) then the truth-evaluation of this restriction is true if the S, P, and the object-filter (will descript later on) are matched. The pattern of it is (S P O) (Rule 3 in Table 2). If the restriction is prefixed with "maybe", the true-evaluation of this restriction is always true. The pattern is now translated to SPARQL as {OPTIONAL {S P O}} (Rule 4 in Table 2). Finally if the prefix is “without”, then the true-evaluation is true if the subject S and predicate P does not appear in a triple. This is emulated in SPARQL (without does not exist is SPARQL syntax) by using optional and object O not bound (Rule 5 in Table 2).

Also we shall refer that in each restriction in the MashQL query there are supported 10 forms of object filters as we can see in figure 9. All of those functions are translated to SPARQL (Def 7 in Table 1, Rules 6-12 in Table 2). Note that not all these function can be directly transformed to SPARQL so some of then are emulated into it (like OneOf, NotOnOf).

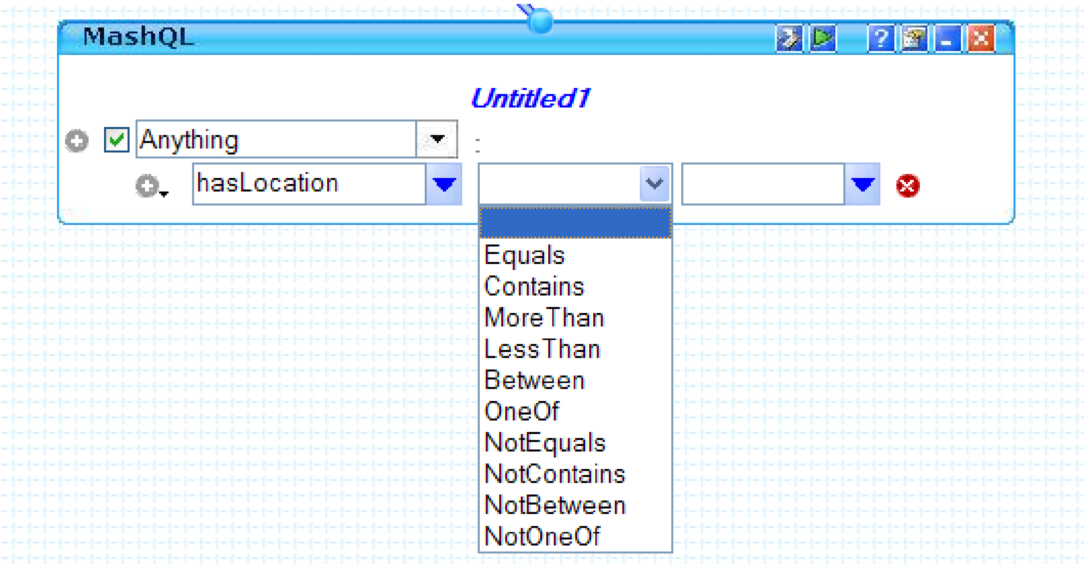


**Figure 9 - Query object filter function in MashQL**

To allow users formulating queries at the “type” level the construct “Any” is used before a subject, to retrieve the instances of this subject instead of the subject itself as shown in the query in figure 8 (Def 8 in Table 1, Rule 13 in Table 2).

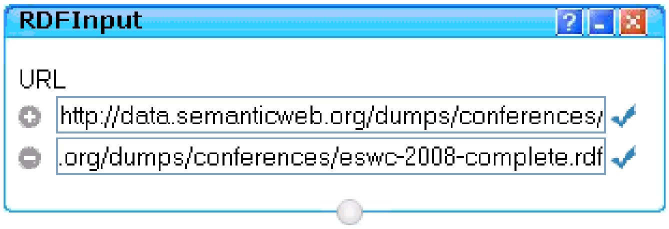
Finally we should state that in the case that we have in figure 8, the query is the following: Select any article that has a title and an author x1, x1 has an affiliation x2, x2 has a country x3, and x3 has a name Malta. Also the articles have year greater than 2007. Because the query is represented as a tree, these variables (x1, x2, and x3) are implicit from the end user. These variables are used in the background to formulate and execute the query. The user sees nothing of these variables.

### 3.3 Query Formulation Algorithm

In this section we will present a step-by-step query formulation. The complexity and the responsibility of understanding the data source, which is schema-free, are not user's responsibility, but the editor's. Figure 11 shows an example of a query formulation.

A simple query formulation consists of 4 steps. The user first has to specify the source, and then the connection of the source is done to the query. The next steps refer to subject selection, then the property, and finally the object or object filter selection. Final step refers to the return variables in the query. After formulating the query, it can be connected though the output terminal to the pipe output module for further processing.

In the algorithm we will refer to the dataset specification as D. The user can specify the dataset, by writing the source in the RDFInput module of the editor as illustrated in figure 10.



URL
http://data.semanticweb.org/dumps/conferences/
.org/dumps/conferences/eswc-2008-complete.rdf

**Figure 10 - Specification of dataset sources**

After we specify the data sources we can connect the RDFInput module though a wire to a query module and start the query formulation. In the query formulation algorithm we will refer to the data source specification as step 0, because this is done outside of the query module. However this step is required so it has to be included in the algorithm.

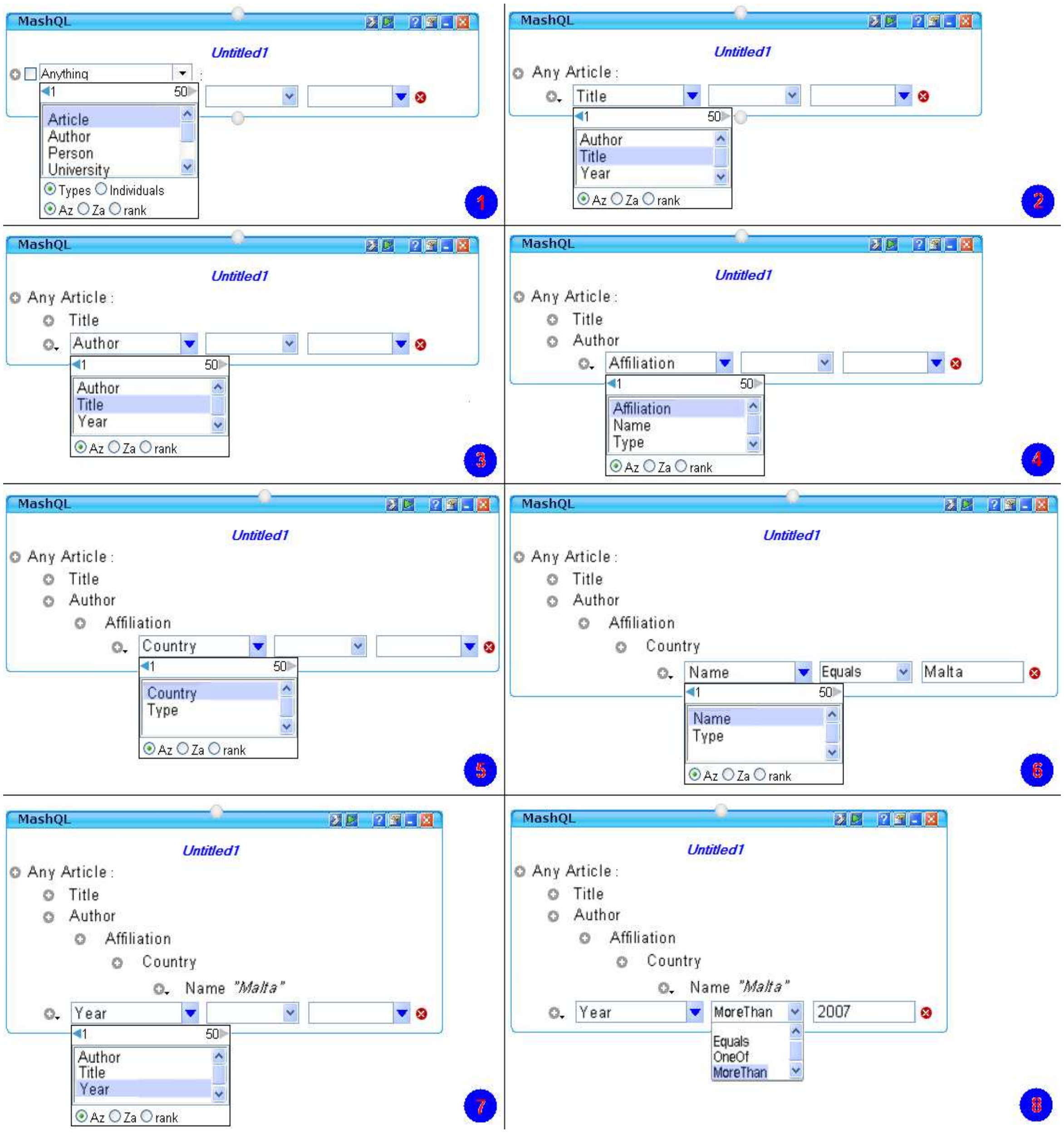


**Figure 11 - Query Formulation example**

The query formulation algorithm is the following:

**Begin**

**Step 0:** Specify the dataset D in the input module. D can be a merge of multiple data sources.

**Step 1:** Select a subject S, where $S \in S_T \cup S_I \cup V$. The user can select S from a drop down list that contains: $S_T$: the set of all subject-types, $S_I$: the union of all subjects objects identifiers in the dataset (URI and key), V: not select from a list and introduce own label for S. In the last case the subject will be in italic style, which means anything. The default value for S is “Anything”. These three options as [18] refer, are presented in relational algebra and SPARQL as follows:

(1) $S \in S_T : \pi_O(\sigma_{P=':Type'}(D))$

(1′) `O1:{(?S1 <rdf:Type> ?O1)}`

(2) $S \in S_I : \pi_S(D) \cup \pi_O(\sigma_{O=URI}(D))$

(2′) `S1:{(?S1 ?P1 ?O1)} UNION O1:{(?S1 ?P1 ?O1).Filter isURI(?O1)}`

(3) $S \in V$

**Repeat Step 2-3 until the user stops**

**Step 2:** Select a predicate/property P. This is depended on the S selected in step 1. In other words the list of properties is dependent on the subject selected before. Here we have four possibilities. The first one is that $S \in S_T$ such the “Article” in the example. The list will display all the properties that this S has as properties like “Title”, “Author”, and “Year”. The second case is that $S \in S_I$. The list will show all properties that this S has. The third case is that the S is a variable and the list will show all the properties of all S exist in the dataset. The last case is that the P is a variable (value introduced from the user). In [18] the formalization of these four cases is the following:

(4) $(S \in S_T) \rightarrow P \in \pi_{P2}(\sigma_{P1=':Type' \wedge O1=Subject}(D) \bowtie_{S1=S2} \sigma(D))$

(4′) `P2:{(?S1 <rdf:Type> <S>)(?S2 ?P2 ?O2)}`

(5) $(S \in S_I) \rightarrow P \in \pi_P(\sigma_{S=Subject}(D))$

(5′) `P1:{(<S> ?P1 ?O1)}`

(6) $(S \in V) \rightarrow P \in \pi_P(\sigma(D))$

(6′) `P1:{(?S1 ?P1 ?O1)}`

(7) $P \in V$

**Step 3:** Select an object filter. This selection stands for filtering the P property selected in step 2. We have three types of filtering: a filtering function, an object identifier, and a query path. In the filtering function case, the user can select from a list of function (e.g. equals, contains etc) a function to restrict the P and add the restriction value afterwards. In the object identifier case, the user selects a value from a possible list of objects [18].

**(8)** $(S \in S_I) \wedge (P \in V) \rightarrow O \in \pi_{O1}(\sigma_{S1=S \wedge O1=URI}(D))$

**(8′)** `O1:{(<S> ?P1 ?O1) Filter isURI(?O1)}`

**(9)** $(S \in S_I) \wedge (P \notin V) \rightarrow O \in \pi_{O1}(\sigma_{S1=S \wedge P1=P \wedge O1=URI}(D))$

**(9′)** `O1:{(<S> <P> ?O1) Filter isURI(?O1)}`

**(10)** $(S \in S_T) \wedge (P \in V) \rightarrow O \in \pi_{O2}(\sigma_{P1=':Type' \wedge O1=S}(D) \bowtie_{S1=S2} \sigma(D))$

**(10′)** `O1:{(?S1 <rdf:Type> <S>)(?S1 ?P2 ?O2)}`

**(11)** $(S \in S_T) \wedge (P \notin V) \rightarrow O \in \pi_{O2}(\sigma_{P1='rdf:Type' \wedge O1=S}(D) \bowtie_{S1=S2} \sigma_{P2=P}(D))$

**(11′)** `O:{(?S <rdf:Type> <S>)(?S <P> ?O)}`

**(12)** $(S \in V) \wedge (P \in V) \rightarrow O \in \pi_O(\sigma(D))$

**(12′)** `O1:{(?S1 ?P1 ?O1)}`

**(13)** $(S \in V) \wedge (P \notin V) \rightarrow O \in \pi_O(\sigma_{P_1=P}(D))$

**(13′)** `O1:{(?S1 <P> ?O1)}`

In the query path case, the user can expand the property of P and declare a path on it. In this case the property of the restriction will be the subject for the branch of it. The general cases of for an n-level property are:

**General Cases**

The n-level paths properties and objects, in case $(S \in S_T)$

**(14)** $P \in \pi_{Pn}(\sigma_{P1=':Type' \wedge O1=S}(D) \bowtie_{S1=S2}(\sigma_{C2}(D) \bowtie_{O2=S3}(\sigma_{C3}(D) \ldots \bowtie_{On-1=Sn}(\sigma_{Cn}(D)))))$

**(14′)** `Pn:{(?S1 <Type> O)(?S1 P2 O2)(O2 P3 O3) … (On`$_{-1}$` ?Pn ?On)}`

**(15)** $O \in \pi_{On}(\sigma_{P1=':Type' \wedge O1=S}(D) \bowtie_{S1=S2}(\sigma_{C2}(D) \bowtie_{O2=S3}(\sigma_{C3}(D) \ldots \bowtie_{On-1=Sn}(\sigma_{Cn}(D)))))$

**(15′)** `On:{(?S1 <Type> O)(?S1 P2 O2)(O2 P3 O3)…(On`$_{-1}$` Pn ?On)}`

The n-level paths properties and objects, in case $(S \in S_I)$

**(16)** $P \in \pi_{Pn}(\sigma_{C1}(D) \bowtie_{O1=S2}(\sigma_{C2}(D) \bowtie_{O2=S3}(\sigma_{C3}(D) \ldots \bowtie_{On-1=Sn}(\sigma_{Cn}(D)))))$

**(16′)** `Pn:{(S1 P1 O1)(O1 P2 O2 … (On`$_{-1}$` ?Pn ?On)} \\Subject ∈ SI`

**(17)** $O \in \pi_{On}(\sigma_{C1}(D) \bowtie_{O1=S2}(\sigma_{C2}(D) \bowtie_{O2=S3}(\sigma_{C3}(D) \ldots \bowtie_{On-1=Sn}(\sigma_{Cn}(D)))))$

**(17′)** `On:{(S1 P1 O1)(O1 P2 O2)(O2 P3 O3)…(On`$_{-1}$` Pn ?On)}`

**Step 4:** Indicate the return values (projections) of the query. The ☑ symbol before each variable of S, P, and O indicates that this value will be returned in the results. At least one return value must be indicated by the user, so that the query could be executed correctly.

**End**

We have presented the algorithm of formulating a query in MashQL editor. This algorithm, except from the user interaction with the editor, it illustrates how it uses background queries to fill the atoms' lists that the user can select the values for each of S, P, and O. The design of the algorithm allows the user to navigate and query a data graph, without any prior IT knowledge, or schema knowledge. The user can learn the content as he/she navigates though it.

In the next chapter we will discuss the details behind all the techniques and mechanisms that are used to enhance the query formulation process, and also all the functionality that the editor offers to the users.

# Chapter 4

## Editor Design & Implementation

### 4.1 System Architecture

The MashQL Editor is implemented as an online server-side query and mashup editor and it has an interface like the one illustrated in figure 12.

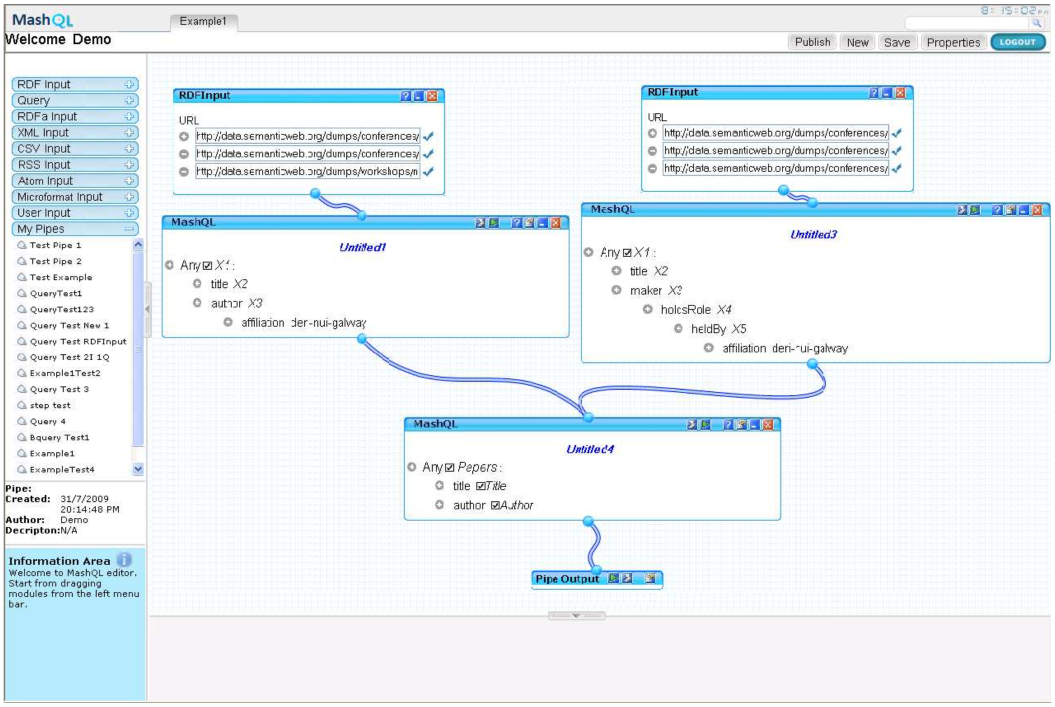


**Figure 12 - Screenshot of the online MashQL Editor**

The editor is implemented to have the following main functionality:

1. MashQL language Components (section 4.3.1).
2. The user-interface (section 4.2.1).
3. A state-machine dispatching the “background queries” to support the interactive exploration of RDF datasets (section 4.4.2).
4. Parser module in XML format that translates the formulated MashQL query into SPARQL and Oracle SPARQL for execution or debugging (section 4.4.3).
5. Results Renderer module that retrieves, merges, and presents the results of the submitted SPARQL query (section 4.4.1).
6. Pipes generator module that saves the generated pipe in an XML format, and load a pipe by parsing the XML pipe file into MashQL diagram (section 4.4.4).

The general idea of how the editor works is that when the user specifies an RDF source/sources as input in the RDFInput component of the editor, the source(s) is bulk-loaded into an Oracle 11g database. When the loading finishes, the resource is ready to be used for query formulation.  Later when the user formulates the queries, the editor uses AJAX to dispatch background queries and use JavaScript to translate and formulate MashQL queries to SPARQL and Oracle Sparql to execute them by using Oracle 11g database. The figure 13 shows the system architecture of the editor.

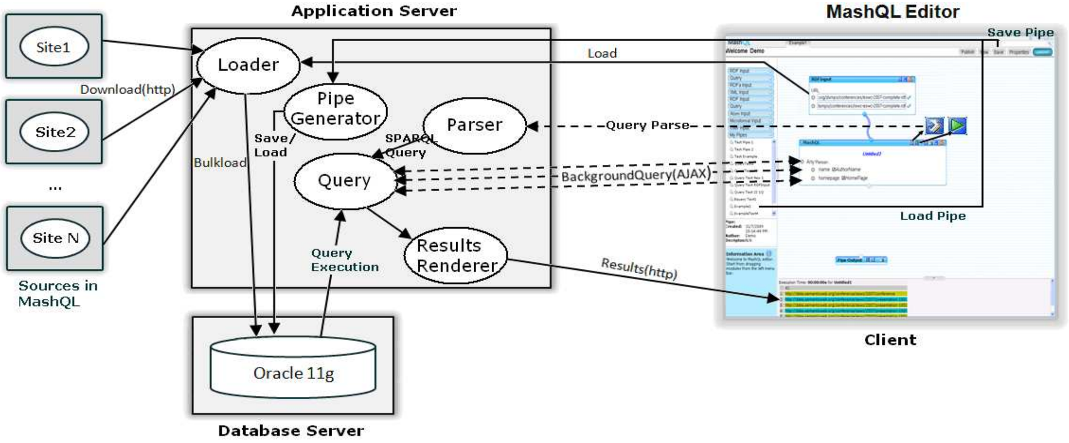


**Figure 13 - System model**

The editor is implemented by using the 3-tier architecture. The client from a browser can interact with the editor which resides in the application server site. The interaction between the editor and the database is made by using JDBC connections from the application server site to the database server site where the Oracle 11g database exists. The system model in figure 13 presents the interaction that the client has with the editor and the editor with the database.

One of the first actions that the user will make is to drag an RDFInput component in the editor's drawing area so to enter an RDF source to query from. The mechanism which is responsible to allow the user to drag and drop a component in the drawing area and construct the component's design, menus, fields, and functionality inside, is the component creator (section 4.3.1). The component construction is executed in the client's browser by using JavaScript so that to avoid overload the application server, because a lot of components will be constructed at the same time from various users.

When the user initialize a new source in the RDFInput component, the loader mechanism (section 4.4.5) of the editor, executed in the application server site, is responsible to send the request to the database and check if the source already exists in the database. If not, then the loader is responsible to signal a source downloading to the database and check every 6 seconds if the content is downloaded to inform the user that the source is ready to use. The loader is implemented on the application server site, since it has to interact with the database frequently.

As soon as the resources are ready, the user can create a Query component so that he/she can query those RDF sources. In the Query component the application is helping the user to formulate a query by suggesting candidate values that exist in the resource queried. The lists of these values are filled by the background queries mechanism (section 4.4.2). This mechanism is implemented in the application serve site, and is triggered by using AJAX when the user clicks on the query's component drop down lists. When the mechanism receives a request it formulates the appropriate query and send it to the database for execution. The results are fetched back from the database, and passed though the normalizer mechanism (section 4.3.2) which is executed in the client site, and is responsible to formulate the results in a readable and ordered way. The mechanism is then responsible to respond back to the editor the results, so that the user can choose a value from the lists. These lists are generated locally on the client side (section 4.3.4). The Query component also has a verbalizer mechanism (section 4.3.3) which is executed in the client site and its purpose is to further improve the elegancy and readability of the restrictions in a query.

When the query formulation is finished, the user can execute or debug the query by pressing the execution or debug button respectively, available at the Query module menu. In both cases of execution and debug, the editor calls the parser mechanism (section 4.4.3) which is responsible to parse the MashQL query diagram into Oracle SPARQL query and send it to the Query mechanism which interacts with the database and sends the results to the results renderer (section 4.4.1) for visualization of them in the debug area of the editor.

When the user wants to save or load the created pipe, he/she has to click the save button or load a pipe from the list of pipes that already saved. When one of these actions is taken, then the pipe generator mechanism (section 4.4.4) existing in the application server site is called. Pipe generator is responsible to parse the MashQL pipe's diagrams to an XML file and save it to the database in the case of save. In case of loading an existing pipe, this mechanism is responsible to retrieve the XML of the pipe and parse it into a MashQL pipe diagram.

In this section we present the cycle and functionality of the whole application that the user can do. In the next sections we are going to see these functionalities in detail.

## 4.2 Software Architecture

The implementation of MashQL editor is separated into the interface parts and the logic of the application. The interface parts are only display and visualize the editor interface to the user's browser. The different functions and mechanisms, and the interactions to the database are made in the libraries called from the interface transparently from the user. The logic is separated into JavaScript functions, JSP functions, and Java classes. Each function is included in the appropriate library. In other word, a JavaScript library includes the JavaScript functions that are executed in the client site, a JSP library include the JSP pages that mostly interact with the database and executed in the application server side, and the Java classes are included in a package at the application server site. Figure 14 shows the software architecture of the editor.

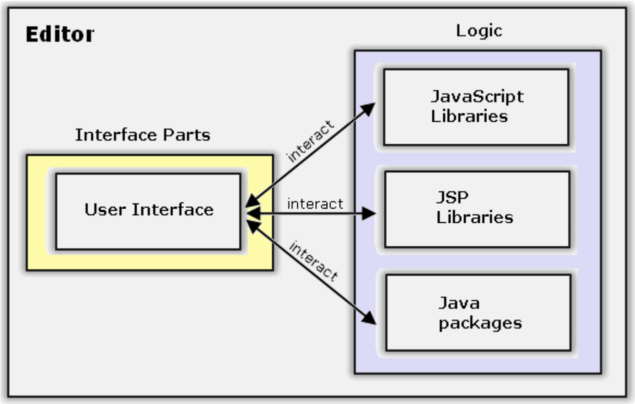


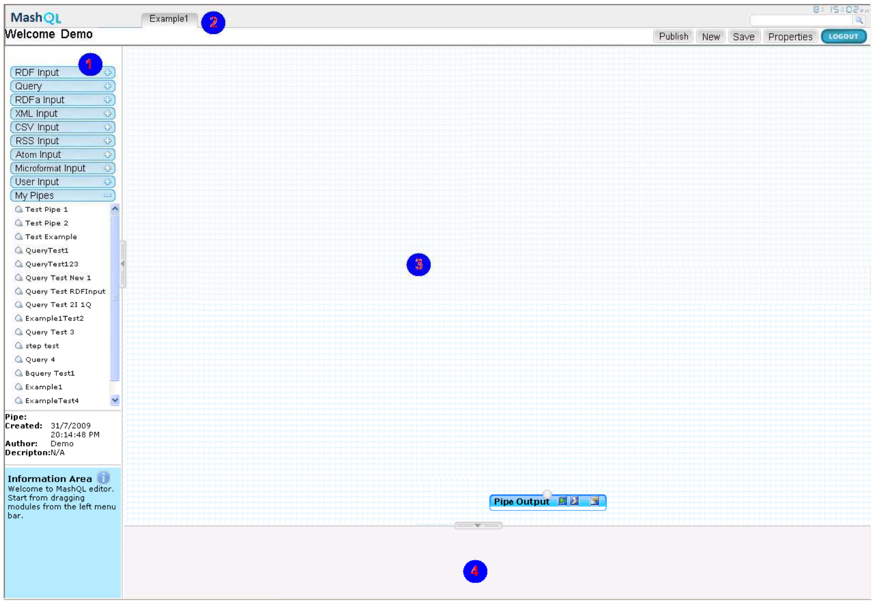
**Figure 14 - Software Architecture**

### 4.2.1 Graphical User Interface

In this section we are going to present the interface of the editor, and what functionality it supports. All the functionality of the User Interface (implemented with HTML) is only executed in the client site and does not affect the application server site.

First of all we shall clarify that this first version of MashQL editor runs under Internet Explorer 6+. The reason why is implemented only for Internet Explorer is that the editor has a lot of JavaScript and Ajax calls, where the syntax, methods, and objects of them differ between the browsers.

When the user registers and logs-in the editor, the interface that he/she can see in like the one appears in figure 15.



**Pipe:**
**Created:** 31/7/2009 20:14:48 PM
**Author:** Demo
**Decripton:**N/A

**Information Area**
Welcome to MashQL editor. Start from dragging modules from the left menu bar.

**Figure 15 - MashQL Editor GUI**

As one may notice, the MashQL's graphical interface follows the style of Yahoo! Pipes that can visualize feed mashups, and also the style that most development tools have, so that the user will be familiar of what he/she expected to do. In addition, the choice of following the style that Yahoo! Pipes has is to illustrate that MashQL can be used to *query and mashup the web data as simple as filtering web feeds.*

Also we have to refer that the user does not have to write complex scripts, or even any script at all, since everything is done transparently to the user. The user only has to work with the language components and formulate the queries by using the interface.

The editor consists of four main parts:

1. **Control Panel** (Figure 15 point 1): Consists of these three modules:
    a. **Language components:** Including all the components that the user can drag to the editor. It calls the libraries of the editor to create each of these components (section 4.3.1).
    b. **Pipe Information:** Includes information about the current pipe, including the pipe name, creation date, author of the pipe, and the description of the pipe.
    c. **Notification Area:** Display information about the components, warning messages about mistakes that the user is doing during development, and error message on severe errors from user's or server's site.
2. **Main Menu** (Figure 15 point 2): Consists of:
    a. **Pipe Title:** update the title of the current pipe.
    b. **Search:** search functionality for published pipes from all the users.
    c. **Menu buttons:** include buttons that have functionalities about the current pipe and the editor, including:
        i. **Publish:** publish a pipe.
        ii. **New:** clears current editor to start a new pipe.
        iii. **Save:** save current pipe.

iv. **Properties:** shows current pipe's name and description for update.

3. **Editor** (Figure 15 point 3): A drawing area where the whole process of the pipe design is made. In here the user can drag the control modules from the control panel area, and start placing the input sources and formulating the queries.

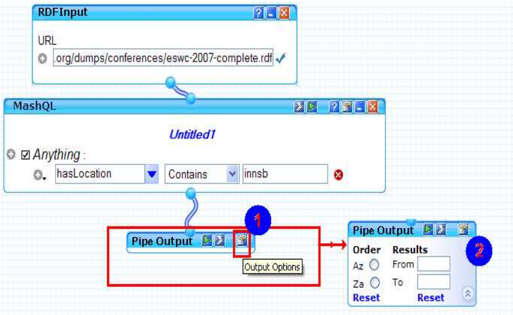
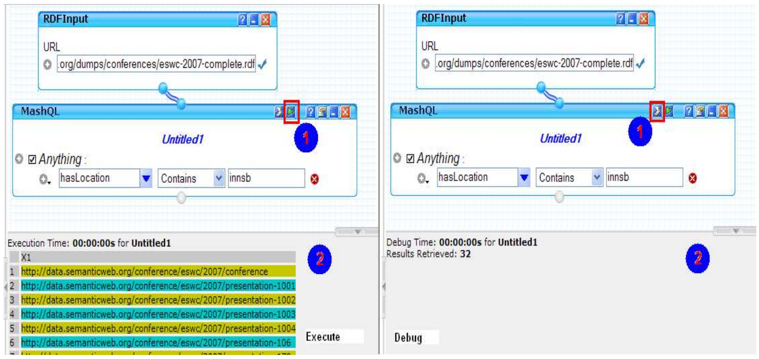


**Figure 16 - Editor's pipe output module**

In figure 16 there is an example of a pipe where an RDF Input source is wired though its terminal to a query module.

4. **Debugger** (Figure 15 point 4): The debugger area is the place where the results of execution or debug of a pipe or a query module are visualized. In figure 17 we can see these two cases are visualized in an example.



Execution Time: **00:00:00s** for **Untitled1**

| | X1 |
|---|---|
| 1 | http://data.semanticweb.org/conference/eswc/2007/conference |
| 2 | http://data.semanticweb.org/conference/eswc/2007/presentation-1001 |
| 3 | http://data.semanticweb.org/conference/eswc/2007/presentation-1002 |
| 4 | http://data.semanticweb.org/conference/eswc/2007/presentation-1003 |
| 5 | http://data.semanticweb.org/conference/eswc/2007/presentation-1004 |
| 6 | http://data.semanticweb.org/conference/eswc/2007/presentation-106 |

Debug Time: **00:00:00s** for **Untitled1**
Results Retrieved: **32**

**Figure 17 - Debugger execute/debug results example**

When the user interacts with any of these components of the interface, a call to the libraries is made to do the appropriate job, and when finish, they respond back to the interface for visualization to the user's browser. Next we are going to present the logic behind the editor.

### 4.2.2 Editor’s Logic

The logic of the MashQL editor resides at the libraries and packages inside the editor. These libraries are called from the main HTML page of the MashQL editor where the presentation and interface exists. When an action is taken in the editor the control is set to these libraries, which are responsible to accomplish the task and respond with the final results to the user interface.

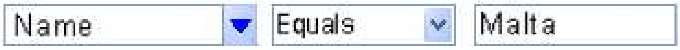
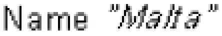
The editor has three kinds of libraries. The JavaScript libraries are executing code to the client site, so that they would not overload the application server. The JSP pages and Java libraries are executing code to the application server site, having mostly functionality which has to interact with the database server.

#### 4.2.2.1 JavaScript Libraries

The JavaScript Libraries (having an extension of .js) include functions that are written in a JavaScript format only. The JavaScript gives the opportunity to dynamically create/update/delete HTML elements (like the MashQL Language Components) and also is executed in the client site which this not overloads the server and we don’t face any extra delays in the asynchronously requests/responses by using AJAX.

In these JavaScript libraries we have implemented functionalities that have nothing to do with the database, but have to do with the presentation and the readability of all the components in the editor. These functionalities are the following:

1. MashQL Language components (section 4.3.1): dynamically generates MashQL components (RDFInput, Query etc.). Also includes functionality to remove components, and add/update restrictions inside the components.
2. Normalizer - URI Normalization (section 4.3.2): takes as an input the retrieved values from the database in the format of *http://hostname.domain:port/filepathname#anchor* and returns a normalized value which is readable, not lengthy, and understandable to the user.
3. Verbalizer - Restrictions Verbalization (section 4.3.3): hides the unnecessary fields in the Query component of the editor, and converts the syntax to a natural language understandable to the user. For example:

| | |
|---|---|
| Edit Mode of a query restriction | <br>Name ▼ Equals ▼ Malta |
| Verbalized mode | <br>Name *"Malta"* |

4. Generator - Scalable drop down lists (section 4.3.4): adds to the components the customized drop-down lists, which have the extra functionality.

We have also included the **WireIt** library, a library from Yahoo!, which is implementing the wiring of the different components in the editor. In other words this library implements the connection between the components in the editor. We have customized these libraries to our application so it works smoothly and right.

#### 4.2.2.2 JSP pages

Java Server Pages are executed in the application server side and are responsible to formulate the queries that will be sending to the database for execution. They are also responsible to create a JDBC connection to the database and send the requests. Afterwards the response is formulated in a specific format depending on the request from the client site. Finally the results are sending back to the client site for further process.

The functionality existing in these libraries is called from the user interface or the JavaScript libraries, depending on the task the client decides to assign. As we have mention previously, in these libraries there are functionalities that have to interact with the database, execute a query, formulate the results, and return them to the calling function. The main functions that are implemented in these libraries are the following:

1. Results Renderer (section 4.4.1): function responsible to send the appropriate query to the database, get the results from the execution of the query, formulate them in the appropriate format and send them back.
2. Background queries (section 4.4.2): function which is responsible to execute asynchronously the small queries that are made during the query formulation process, transparently from the user.
3. Pipes Generator (section 4.4.4): function responsible to save or load a pipe generated in the MashQL editor. It parses the MashQL pipe diagram into XML format and vice versa. The storage of the XML file is made in the database.
4. Loader (section 4.4.5): function to send a request to the database to download the content of the RDF resource.

There are also some other secondary functions implemented in these libraries:

1. Check value: check selected value in a query if exists in the database.

2. Load personal pipes: when the editor is loaded, an asynchronous call is established to the database, so that to load each user's saved pipes, and allows the loading of them for further process.
3. Execute loaded pipe: in the case that the user selects a particular pipe, then this mechanism is responsible to load the MashQL pipe's diagram and execute each component to test its validity.
4. Search function: A function that asynchronously by using AJAX searches the database on each keystroke from the user side, and displays all the pipes that are stated as public (each user saves the created pipe with the public option enable), and much the search criteria.

#### 4.2.2.3 Java classes

The Java library that is used in the editor is consisting of many java classes (section 4.4.3). Each class keeps the attributes and properties that the XML of the MashQL diagram elements has (RDFInput, Query component, etc). It is used in the pipe generator mechanism (section 4.4.4) in the case of saving or loading a pipe. Also it is responsible to parse the XML file to MashQL pipe diagram and visualize it in the editor's drawing area in the case of loading, and parse the MashQL pipe diagram to XML to be saved in the database in the case of saving a pipe.

In the next section we are going to present where each of the most important functionalities are executed and what they actually do.

### 4.3 Client Site

In this section we are presenting functionality that is executed in the client site (browser). The code does not need the database or application server physical resources. Most of the functionality has to do with the elements that are visualized in the client and all the properties that these elements have. The code is included in JavaScript libraries.

#### 4.3.1 MashQL Language components

The Editor is constructed to have the welfare to support various file input formats and not only the RDF (Resource Description Framework) format. As we can observe in figure 18 there is a list of all languages, indicated with the word "Input" at the end. Note that for this version of MashQL editor we implement the RDF Input component.

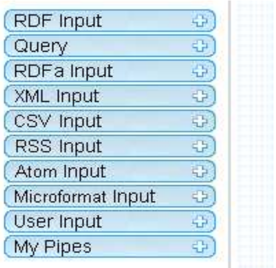


**Figure 18 - MashQL Language Components**

This language component list is located in the Control Panel area of the editor. From this list, the user can drag a module to the editor area, and by letting the left mouse button, the module is generated and is ready to accept the user's inputs. The component generator is responsible to generate the modules in the editor's drawing area. It is implemented by using JavaScript code and is located in the JavaScript libraries.

To understand better the job of the generator the figure 19 shows an example of how to create an RDF Input component.

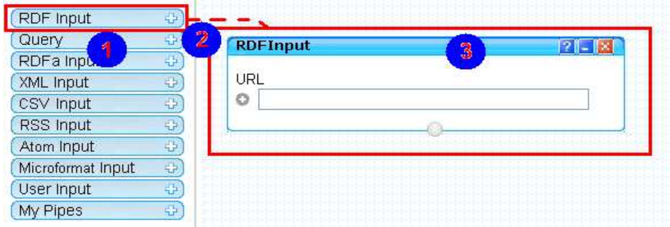


**Figure 19 - How to generate RDFInput module**

The first point in the figure shows all the components of the editor. The second point indicates that on dragging the module into the editor area and letting the button, then the module will be generated by the component generator (3rd point in figure). In the same way every component in this list can be generated and be ready to accept the input sources of the user.

The components that the language component generator can create are the following:

1. **RDFInput**: Accepts RDF (Resource Description Framework) format inputs, as a URI. Must have the extension of .rdf or .nt.
2. **Query**: Component that allows the user to formulate a query by using the triples pattern i.e. <Subject> <Predicate/property> <Object> format. Accepts the Input modules (source inputs) as an input, and can be linked with other query modules or output module as an output.

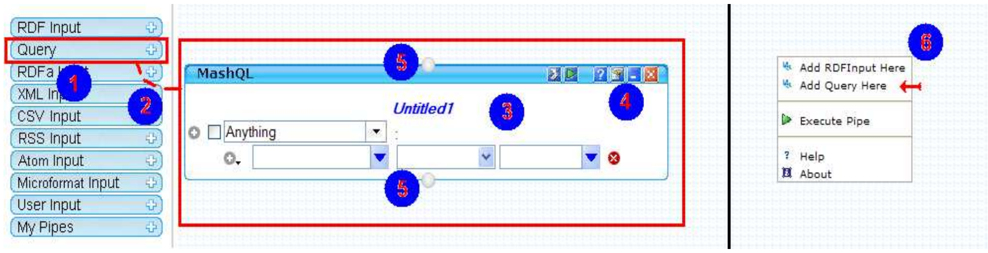
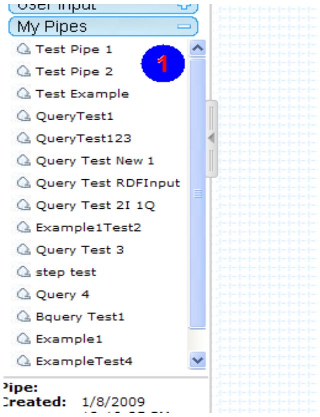


**Figure 20 - Query module formulation**

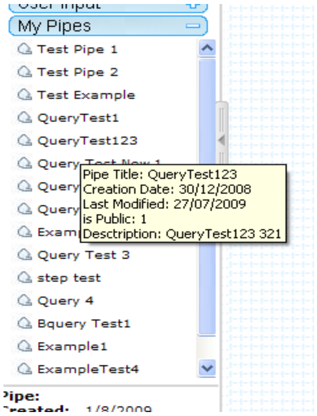
Figure 20, presents how we can create a query module. The first way is by dragging the component in the editor area and the second in to right click in the editor area and select the “Add Query Here” to create it at that point in editor (point 6). Also notice the two terminals (point 5) which one has the role of input sources wired, and the other the output triples to another module input terminal.

3. **User Input**: This module is used as an input module to a query's restriction. In this module the user must specify the datatype (String, Integer, Long, Float, etc.) and specify the values that want to be included in the query restriction.
4. **My Pipes**: This component is responsible to show/hide the pipes that the current user has already design and save. In detailed, when the user clicks on the "My Pipes" component, this is extracted and shows the pipes, as we see in figure 21.



Pipe:
Created: 1/8/2009

**Figure 21 - Pipes already saved**



Pipe Title: QueryTest123
Creation Date: 30/12/2008
Last Modified: 27/07/2009
is Public: 1
Desctription: QueryTest123 321

Pipe:
Created: 1/8/2009

**Figure 22 - Pipe Details**

The user can click on any of those pipe names and load it in the editor for further process. Also the user can hover over a pipe name and see its details as we see in figure 22. Further details of how these components works could be found in Appendixes – MashQL editor online help.

### 4.3.2 Normalizer - URI Normalization

The normalizer function, implemented to normalize the triples' values that are shown in the triples' lists in a query module. Because the data of RDF mostly contain unwieldy URIs, this cause to the MashQL queries to be inelegant and also not readable. For that reason, the URIs needs to be normalized in some way. The editor uses some heuristics for pattern detection of similar URLs and replace with namespaces. Also the function in the case that not any of the namespaces is found, it has an algorithm to normalize the URLs in an elegant way. The pseudo-code of the algorithm is the following:

```
if URI not in R
      take part after #
      if no # then
            take part after last \
      if part after \ or # is less than 3 or starts with number
            take last part of \ before last \
      if no \ found
            take only part after http://www.
      If normalization of URI the same with a URI already
         normalized then
            add a number and a : in front of each URI to distinct
      return normalized value
```

The R represents the namespaces available in the database. In the case that a URI is not found in this namespace then the URI will be normalized by using this algorithm. First the algorithm will take the part after #. For example if we have http://www.bbc.com#news, the value retrieved will be “news”. In the case where no # is found then the algorithm will take the part after the last slash (e.g. http://www.bbc.com/sports will return “sports”). If the value retrieved from the split of the URI in case of “# “or “/” is less than three or starts with a number, then the algorithm will retrieve the part after the second “/” before last slash (e.g. http://www.bbc.com/sports/123 will return “sports/123”. If nothing of the previous conditions is true for a URI, then the algorithm will return the whole URI after the http://www part (e.g. http://www.bbc.co.uk will return “bbc.co.uk”). At the end, if the normalized URI is the same with URIs already normalized, then the algorithm with take those URIs and add a number in front of them so to be distinct. For example if we have three URLs with the same normalized values like:

| URI | Normalized value |
|---|---|
| http://www.bbc.com/sports | sports |
| http://www.euronews.com/sports | sports |
| http://www.phileleftheros.com.cy/sports | sports |

the algorithm will return in the lists the following:

1: sports (with a tip when mouse over it: http://www.bbc.com/sports)

2: sports (with a tip when mouse over it: http://www.euronews.com/sports)

3: sports (with a tip when mouse over it: http://www.phileleftheros.com.cy/sports)

### 4.3.3 Verbalizer – Restrictions Verbalization

The verbalizer is a functionality for the query module again. The purpose of this module is to further improve the elegancy and readability of the restrictions in a query module. When a user clicks the mouse over a restriction, it gets to the editing mode and all the other restrictions get to the verbalized mode. In other words, all boxes and lists are made invisible and their content is verbalized and displayed in their place. The content shown to each verbalized restriction is readable and closer to natural language, and also guides the user to validate whether the content shows is what they intend to formulate in a particular restriction.

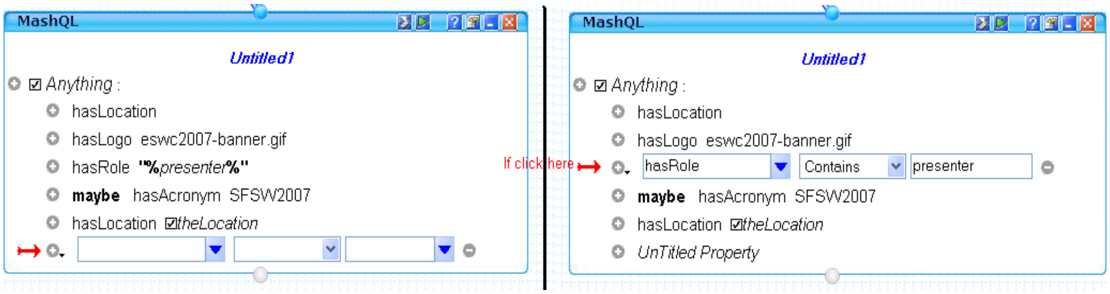


**Figure 23 - Verbalization of query restrictions**

As we can see in figure 23, the query module in picture 1, has 1 returned subject and five restrictions. In the case we press the plus button to add a new restriction; we can see that a new restriction at the end of the last restriction will appear. Moreover, this new restriction will be in its edit mode, whether the other restrictions will be in their verbalized mode. Note that the check box in front of the subject "*Anything*" represents that the subject will be a return value in this query. The second picture in figure 23, shows that if we now click on the third restriction of the query, this particular restriction now gets to its editing mode, whether the others gets to the verbalized mode. Also note that if we are in the editing mode of a restriction, we have the opportunity to remove this current restriction from our query. In addition if the value selected or typed by the user in each of the triples is in italics if and only if the original value (URI) of that value does not exist in the database (variable); else is a constant (exists in the database – selected from the list).

The structure of the editing mode for the subject in the following:

```
(1)[returned value check box] (2) Subject value
(1) Every time the subject value is a constant then is the checkbox is hidden
```

The verbalized mode of the subject is the following:

```
(1) [returned value check box]      (2) subject value
(1) show it if and only if it is checked
(2) italic style if the subject is a variable
```

In the case of editing mode of each of the restrictions the structure is as follows:

```
⊕ (1)[checkbox] (2)P value (3)[function] (4)[checkbox] (5)O value ⊖
(1) hidden if P is constant)
(3) Equals, Contains, MoreThan, LessThan, OneOf, Between, negation of them.
(4) hidden if O is constant)
```

The verbalized mode of the restrictions has the following structure:

```
(1)[checkbox] (2)P value (3)[function] (4)[checkbox] (5)O value
(1) check-box is hidden if P is constant
(2) P value in italics if is a variable
(3) Equals function is verbalized to "{O value}"
    Contains is verbalized to "%{O value}%"
    More than is verbalized to > {O value}
    Less than is verbalized to < {O value}
    Between than is verbalized to {O value Maximum} - {O value Minimum}
    OneOf is verbalized to OneOf{O1,O2,…,On}
    Negation of all above are expressed exactly the opposite
(4) check-box shown only if O is variable and checked, and no function
    selected.
(5) O value represented depended to the {3}. If function selected then O
    is automatically a variable and shown in italics.
```

### 4.3.4 List Generator - Scalable lists

The list generator functionality is used to fill the triples' list inside the query module. This is done on each user's browser. The lists implementation does not follow the usual drop-down list as we know it. Because we have cases of querying large datasets, the well-known drop-down list is not so scalable to be used in the editor. So a new custom list has been developed which is scalable, user-friendly and supports the following functionality:

1. search
2. auto-complete
3. sorting: ascending order, descending order
4. types: a. Types (show every entity that has an rdf:type property)

   b. Individuals (show every entity from the database including and types)

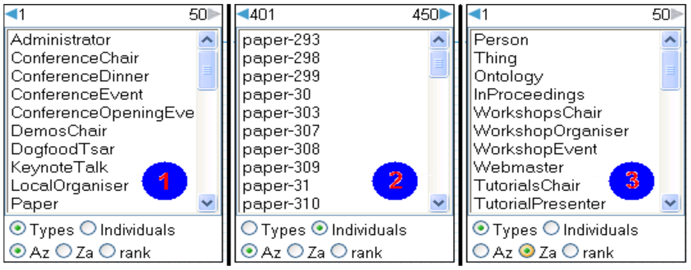
In addition the list generator has the functionality to show 50-by-50 results every time so to avoid showing all the results at a time because of the large datasets to be queried. The user can navigate to next fifty if available or previous fifty. The list generator is responsible to generate those lists for each entity of the triples. In other words, the generator creates three types of lists:

1. Subject list – including all the S property of a particular source/sources. It has all the functionalities described above.
2. Predicate list – including all the P property of particular source/sources. It has all the functionalities except the "types".
3. Object list – includes the O properties. Supports all the functionalities described above.

As we can see in figure 24, there is an example of the list generator. In picture 1 of the figure we can see that the list shows all “Types” from 1 to 50, sorted in ascending order (Az – A to Z). We can also observe that the results do not exceed 50, so there are no next results, and consequently the button to show the next fifty is disable. Also note that if the user moves the mouse over a value in the list, he/she can see the original value (the URI) of that displayed value. The values shown in the list are the normalized one and not the original one, for readability and efficiency purpose.

The lists consist of three sub modules:

1. The results navigator: navigate to next/previous results.
2. The value list: display the values from the database and a scroller to navigate though the results.
3. The menu area: has functionality to select the type of value to retrieve and also the sorting mechanism.



● Types ○ Individuals
● Az ○ Za ○ rank

○ Types ● Individuals
● Az ○ Za ○ rank

● Types ○ Individuals
○ Az ● Za ○ rank

**Figure 24 - List generator**

In addition the figure 24 shows an example of normalized values retrieved from 401-450 for type “Individuals” in an ascending order (picture 2), and all types “Types” from 1-50 in a descending order (picture 3).

The list generator as we already mention, is responsible to generate the list for the S, P, and O atoms. The lists are generated exactly under the text field of the entity that the user wants to add a value, as we can observe in figure 25. In screenshot 1 of the figure we can see the list generated for the Subject entity. Screenshot 2 shows the list generated for the Predicate entity, and screenshot 3 shows the list for the Object entity. Notice that in picture 4, the list generated has fewer results than the same list generated in picture 2.

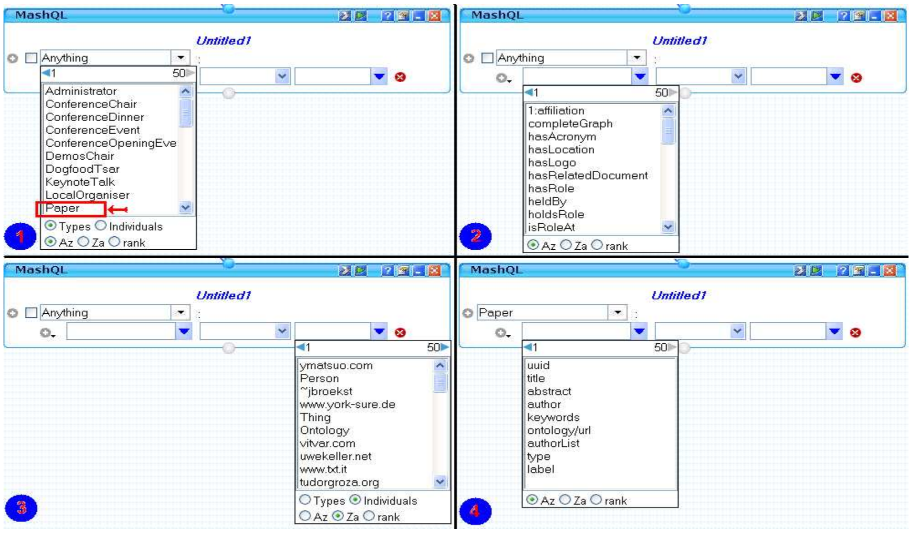


**Figure 25 - List generator for each entity in triples (S,P,O)**

The reason is that in picture 4, we already select a value for the subject entity "Paper" as we can observe in picture 1 where the "Paper" value is shown in the red box. As soon as a value is a constant (selected from the database) this affects the behavior of the child lists. To understand this better, the query executed to fill the predicate list in picture 2 was:

*P1: (?S rdf:type ?O) (?O ?P1 ?O1)*
*Select anything that has a property of type something*

which returns every predicate that has a type. However in picture 4 the query executed is the following:

*P1: (?S rdf:type <Paper>) (<Paper> ?P1 ?O1)*
*Select anything that has a property of type something and subject Paper*

This will limit the results depending on the Paper value (the URI of it). So the behavior of the list generator depends on the previous user's selections and also is affected by the RDF sources that the user assigns as an input to the query module.

The auto-complete functionality of the list generator is also an important and helpful mechanism for the user. As we can see in figure 26, there is an example of how auto-complete works.

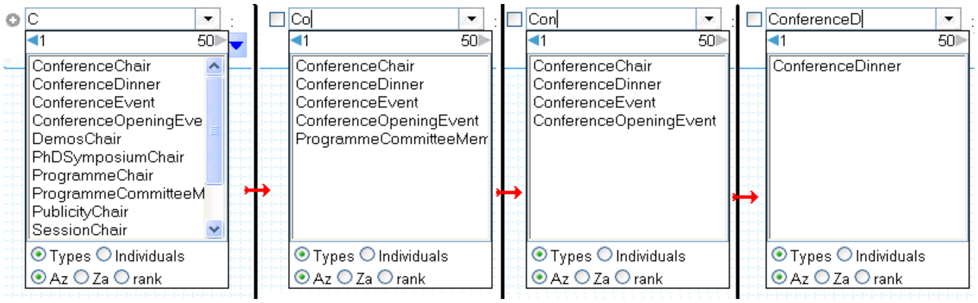


**Figure 26 - List generator's auto-complete mechanism**

In the case that the user starts writing in the text field of an entity, the list is regenerated depending on the keystrokes. As a result, the results are limited every time. This will help the user to decide the value to choose easier, since the datasets to query from are huge.

## 4.4 Application Server

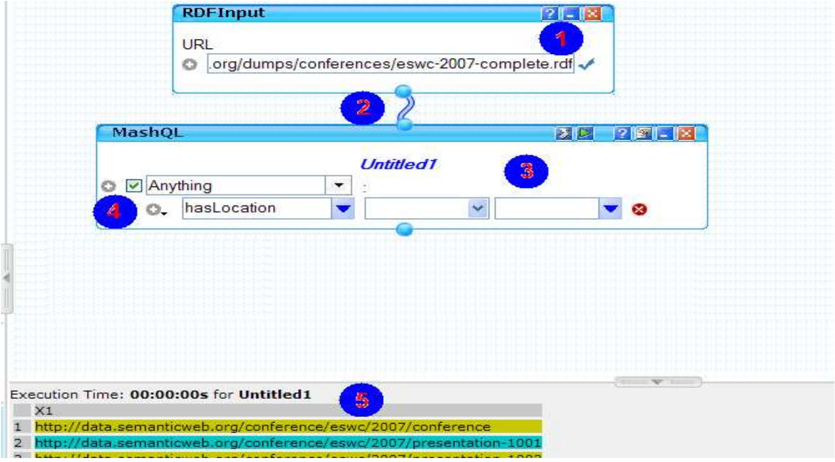
In this section we present the functionality that resides at the application server and is responsible to act as a middleware between the client's request and the database. These functions are responsible to send a formulated query to the database for an execution and retrieve the results back through a reference cursor (Oracle's reference cursor), formulate the results in an appropriate format and return the results back to the client.

### 4.4.1 Results Renderer

The debugger of the MashQL editor is the area where any query or pipe results are visualized inside. The process that is responsible to display those results is the result renderer. This mechanism is called after pressing the “execute” or “debug” button of a query module or a pipe’s output module. The process of what happens in this mechanism is illustrated in figure 28. The process is trigged only if:

1. The user provides a source in an RDF Input module.
2. Connect this RDF Input module to a Query module.
3. Formulate a valid query
4. Select at least one attribute to be returned from the query.

If one of these requirements is not valid then the appropriate message will appear in the notification area of the editor, providing further information of what the user must do. An example of those steps is illustrated in figure 27.



URL: .org/dumps/conferences/eswc-2007-complete.rdf

☑ Anything

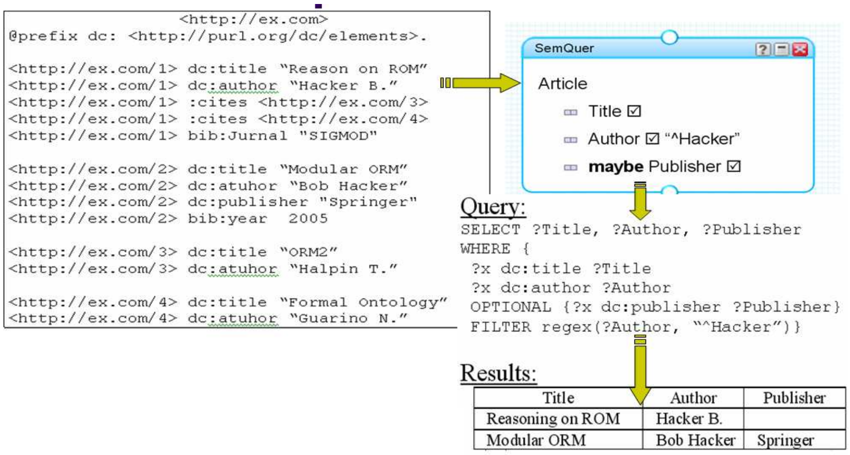
hasLocation

```
Execution Time: 00:00:00s for Untitled1
  X1
1 http://data.semanticweb.org/conference/eswc/2007/conference
2 http://data.semanticweb.org/conference/eswc/2007/presentation-1001
```

**Figure 27 - Query Formulation process**

The figure shows the steps that we already refer to as the prerequisites of a valid query execution or debug. At point 5, the results of the query are illustrated, after pressing the “execute” or “debug” button.

The results renderer mechanism is using an AJAX call to the server, which executes the query and returns the results depending on that query. Then the process visualizes the results in the debugger area. The main idea of the process is illustrated in figure 28.



```
                  <http://ex.com>
@prefix dc: <http://purl.org/dc/elements>.

<http://ex.com/1> dc:title “Reason on ROM”
<http://ex.com/1> dc:author “Hacker B.”
<http://ex.com/1> :cites <http://ex.com/3>
<http://ex.com/1> :cites <http://ex.com/4>
<http://ex.com/1> bib:Jurnal "SIGMOD"

<http://ex.com/2> dc:title “Modular ORM”
<http://ex.com/2> dc:atuhor “Bob Hacker”
<http://ex.com/2> dc:publisher "Springer"
<http://ex.com/2> bib:year  2005

<http://ex.com/3> dc:title “ORM2”
<http://ex.com/3> dc:atuhor “Halpin T.”

<http://ex.com/4> dc:title “Formal Ontology”
<http://ex.com/4> dc:atuhor “Guarino N.”
```

Query:

```
SELECT ?Title, ?Author, ?Publisher
WHERE {
 ?x dc:title ?Title
 ?x dc:author ?Author
 OPTIONAL {?x dc:publisher ?Publisher}
 FILTER regex(?Author, “^Hacker”)}
```

Results:

| Title | Author | Publisher |
|---|---|---|
| Reasoning on ROM | Hacker B. | |
| Modular ORM | Bob Hacker | Springer |

**Figure 28 - Query execution example**

By using the RDF source, the user formulates the MashQL query. By executing the MashQL query, it is transformed to a SPARQL query and then to an Oracle Sparql query. Then this is send to the database and the response in formulated into the format shown in the figure, which is the one that will be displayed in the debugger area.

What is actually happen behind the scenes is a mixture of JavaScript, AJAX, Java Server Pages (JSPs), SPARQL and Oracle Sparql. For the results renderer mechanism the JavaScript is used to instantiate an AJAX object and translate the MashQL query to a SPARQL query. The AJAX takes the query and calls a JSP page which is now responsible to translate the SPARQL to Oracle Sparql because the database is Oracle 11g, having a slightly different syntax that pure SPARQL. After the translation, the JSP sends the request to the Oracle database which executes the Oracle Sparql query and return the execution of the query back to the JSP. Now the JSP sends the results back with AJAX to the JavaScript method that calls that AJAX object, and transform the results to a readable format. Then the results are rendered to the graphical user interface in the debugger area. The diagram in figure 29 shows the renderer mechanism.

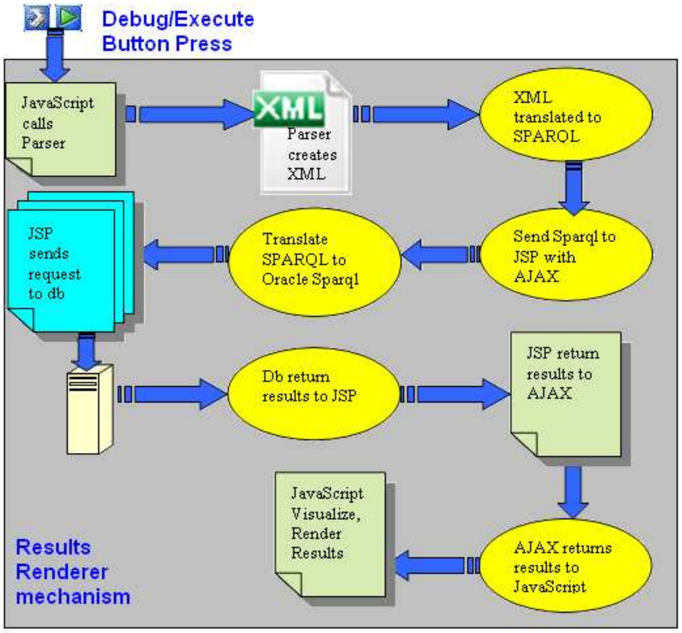

Debug/Execute Button Press
JavaScript calls Parser
XML
Parser creates XML
XML translated to SPARQL
Send Sparql to JSP with AJAX
Translate SPARQL to Oracle Sparql
JSP sends request to db
Db return results to JSP
JSP return results to AJAX
AJAX returns results to JavaScript
JavaScript Visualize, Render Results
Results Renderer mechanism

**Figure 29 - Results renderer process**

If we see this process as a black box, the renderer accepts a MashQL diagram as an input and returns well-formed results of that query diagram in the debugger area of the editor's graphical user interface.

### 4.4.2 Background queries

In this section we are going to present the mechanism behind the vital part of the MashQL editor. The background queries are used in the query components and their purpose is to query the database dynamically and transparently from the user (with AJAX calls) while a query is on the formulation state, to retrieve the values for each of the entity of the triples (S, P, O) and display them in their lists (figure 30). By saying dynamically querying, we mean that the query changes every time, depending on the user actions. In addition the query depends on the result set to query from and to (e.g. to query from row 51 to 100 of the result-set), by keeping a counter of the results size fetched the last time. Finally the query is dependent on the previous selections of the user in the triples hierarchy and the input source specified in the RDF Input module.

The background query mechanism is triggered as soon as the user starts typing in the text-field of each of the triples entity in the query module, or if the user clicks on the icon next to that field, or the user selects a different type or ordering in the lists.

The way that the background queries mechanism is called can be seen in figure 30. In point 1 when the user starts typing on the subject's text-field or if he/she clicks on the ▼ icon to show the list, the mechanism is called and fills the list exactly below the text-field. Also note that if the user selects to change anything in the control area of the list, then the background query is called again to retrieve the filtered values from the database. In the second point the same stands and for the predicate entity text-field. In point 3 the background query is called in the object entity of the triple.

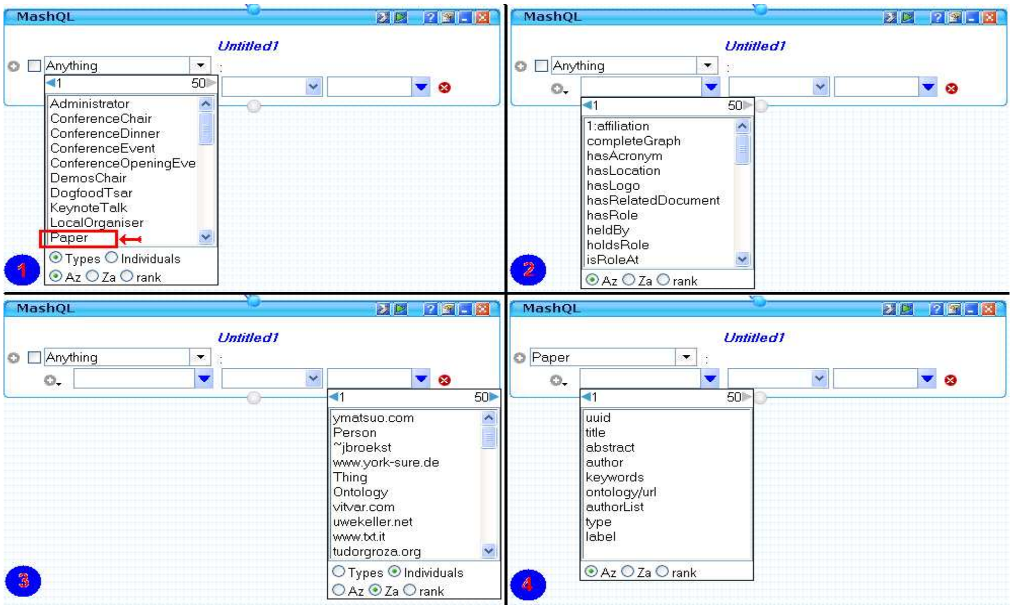


**Figure 30 - Cases of calling background query**

The left arrow and right arrow at the top of the list indicates to show the next fifty or previous fifty respectively. If the user clicks one of those two arrows the background query is called to bring the next or previous values depending on the "from" and "to" conditions and the other conditions than we already refer. Finally note that the background query is restricted also from the previous values that are selected in the triple's entity. For example the point 4, shows the list of the predicates that has a "Paper" as a subject. So now the list will take count the "Paper" condition in the query.

The background query mechanism uses an AJAX call to a procedure in the database when is triggered. This procedure has the responsibility to call the specific dataset's table, and return all the results back to the mechanism that is responsible to formulate the values and display then in the lists.

To sum up, the background query mechanism take into count the following conditions:

1. The sources (defined in RDFInput modules by the user)
2. The query formulated so far
3. The type selected in the list conditions ("Types" or "Individuals")
4. The ordering of results (Az – Ascending, Za – Descending, rank)
5. The from value (the row number to start the query e.g 1)
6. The to value (the row number till the query will be made e.g 51)
7. The condition entered in the entity's text-box of triple
8. Conditions that the parent query connected to this particular query has (if any)

The background query mechanism is called in these cases:

1. typing in the text-field of the entities of a triple
2. clicking the ▾ or ▾ buttons of the text-field of the entities of a triple
3. selecting an option from the list (type kind, ordering)

4. changing the result set from – to conditions

### 4.4.3 Parser of MashQL to SPARQL

The Parser module is a java class that is responsible to translate the MashQL query into SPARQL query. To do that the MashQL query must be translated to an XML file. This XML file is then added as an input to the parser, which returns the SPARQL query. The xsd schema and structure, and an example of the XML file could be found in Appendixes. The java file of the parser is implemented so that it recognizes and transforms the XML elements of the MashQL query design into java classes. The class diagram of the parser is illustrated in figure 31.

The Pipe class has three subclasses: RDFInput, Query, and UserInputType classes. Those three classes represent the entities for the input sources, queries, and user input modules respectively. Each of those classes extends the properties of their subclasses that have all the properties and values needed to formulate the modules. Those properties are the position of the modules and the content of them.

The parser module is called in two cases:

1. Execute/Debug a query/pipe. The parser becomes a part of the results renderer mechanism which we describe in section 4.4.1.
2. Saving/Loading a pipe. Parser is a part of the pipes generator mechanism which we describe in section 4.4.4.

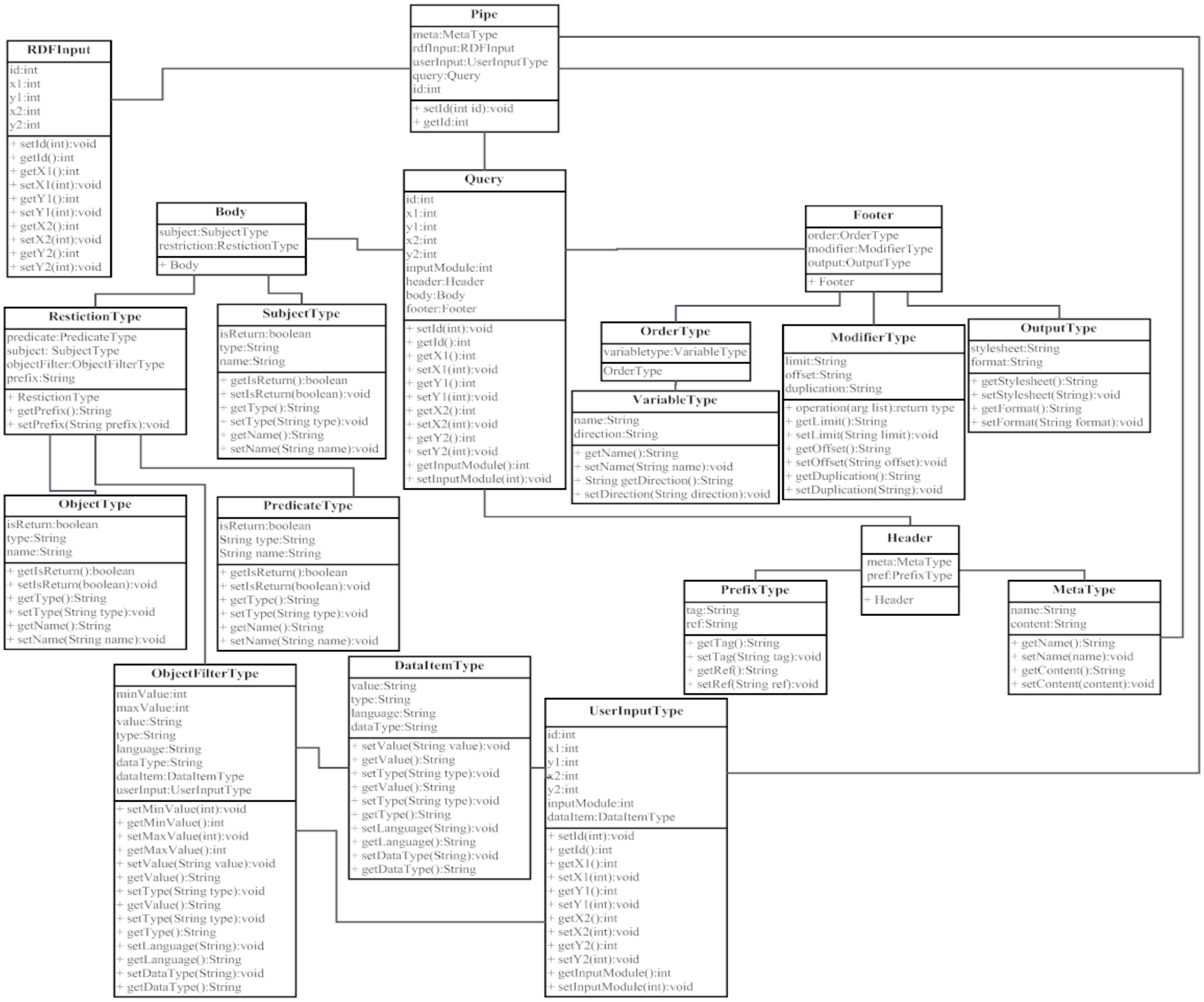


**Figure 31 - Class diagram of Parser module**

As we already mention, the parser is a java class consisting of many other subclasses. To call this java class, the program has to make a call to it by using AJAX. The whole process is illustrated in figure 32.

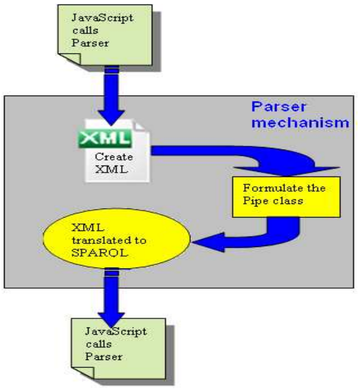


**Figure 32 - Parser mechanism**

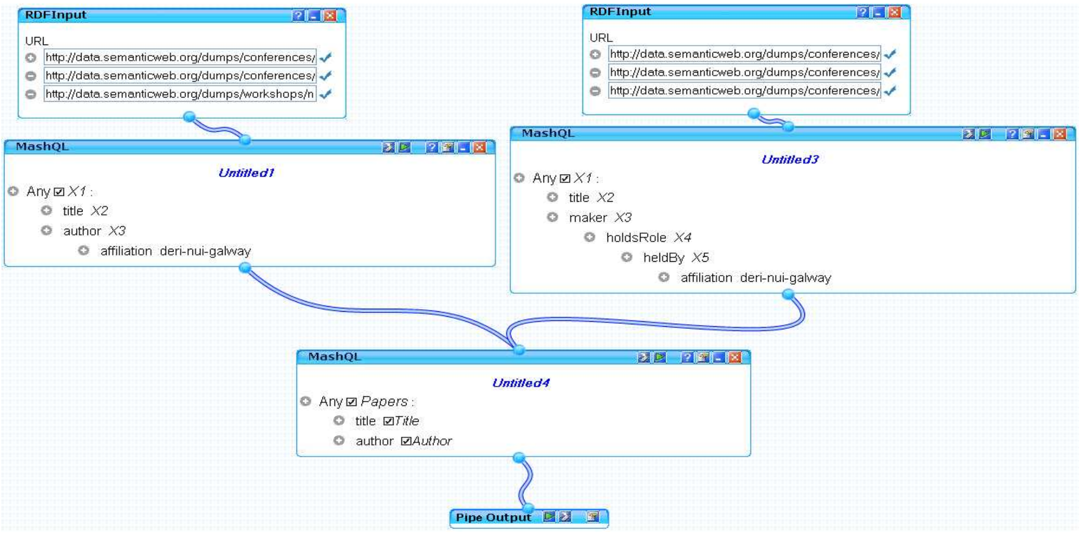
The way that this module is called is the following:

1. A JavaScript method creates an AJAX object
2. The JavaScript method calls a function to transform the MashQL query diagram into an XML file.
3. The JavaScript method sends the request to the parser's Java file through the AJAX object with the XML file as an input.
4. Parser converts the XML file into SPARQL.
5. Parser returns the SPARQL query as a string back to the JavaScript method by using the same AJAX object.
6. JavaScript object continue the processing.

### 4.4.4 Pipes Generator

To save a pipe we have created an xsd schema available in Appendixes. The current xml implementation of a pipe is divided into four child elements and one attribute. The four child elements are the Meta, RDFInput, UserInput, and Query elements. It also has an attribute which keeps the id of the pipe.

When the user desires to save the developed pipe in the editor, and presses the save button, automatically the pipe generator module is called, which is responsible to generate an xml file according to this schema and the inputs that the user has select in the editor. Let's say that the user decides to create the query that is shown in figure 33.



URL
http://data.semanticweb.org/dumps/conferences/
http://data.semanticweb.org/dumps/conferences/
http://data.semanticweb.org/dumps/workshops/n

URL
http://data.semanticweb.org/dumps/conferences/
http://data.semanticweb.org/dumps/conferences/
http://data.semanticweb.org/dumps/conferences/

Untitled1
Any ☑ X1 :
title X2
author X3
affiliation deri-nui-galway

Untitled3
Any ☑ X1 :
title X2
maker X3
holdsRole X4
heldBy X5
affiliation deri-nui-galway

Untitled4
Any ☑ Papers :
title ☑Title
author ☑Author

**Figure 33 - MashQL query example**

For information purpose, the sample query in figure 33 has six different input sources, divided in two input modules. Those input modules are connected to a query module respectively. The results of those two queries are then used as an input to a new query module. This module acts like a filtering to the previous results. This query module is connected to the pipe output module, which is responsible to execute the whole pipe.

In the case the user wants to save this pipe, and presses the save button from the main menu, the pipe generator module is called to do the job. It is responsible to translate the MashQL pipe design to an xml format and then save this xml to the database. In this scenario we have two input modules, three query modules, four connections between them, and one connection to the pipe output module. The xml of this MashQL example is shown in Appendix- XML file example.

The structure of this XML file is validated though the MashQL.xsd schema, which was created to satisfy the editor’s need to create/load a pipe. So every XML file follows this structure and all limitations of it. This structure is shown in the Appendix-XSD Schema to save a Pipe.

To sum up, the cycle to save a pipe has the following procedure. First the user uses the editor to create a pipe like the pipe shown in figure 33. Secondly, when the save button is pressed, the pipe generator function is called, which is responsible to parse the MashQL diagram into an XML file as shown in Appendix- XML file example, and then validate it though the xsd schema shown in Appendix-XSD Schema to save a Pipe. When this job finished, the function saves the XML file of the pipe in the oracle database together with the other information of the pipe, like the description, the title, the created date, the id of the user, and the latest modified date.

In case a pipe is loaded to the editor, exactly the opposite procedure is followed. First the user selects from the editor the pipe to load. Secondly, the pipe generator function retrieves from the database the particular pipe's XML file. Then translate the XML file to a MashQL diagram and shown shows the pipe in the editor area for further process by the user.

### 4.4.5 Loader

The loader function is responsible to check if the requested RDF resource's URL is already available in the database. In the case that the URL of the resource exist in the database, the context of it is already downloaded, the downloaded date is less than 1 day, and the downloaded status is success, then this function is responsible to inform the user that he/she can use that resource.

In any other case, the function has to trigger the database to download the context of the resource by providing the URL. The Oracle inside has its own version of JENA which is responsible to download an RDF source into Oracle's semantic tables.

When the download starts, the loader function is responsible to check the status of the downloading every six seconds. This is done transparently from the user by using AJAX calls. When the downloading finishes, the function is responsible to inform the user that he/she can use the resource if it is downloaded successfully, or that he/she cannot use the resource in case that something went wrong.

## 4.5 Database Server

In this section we present the back-end of the application. We use Oracle's 11g semantic technology, because this enables querying RDF resources that are saved in the semantic technology's tables (transparently from the user), by using Oracle's SPARQL.

We have use two functions that are available to this semantic technology for our implementation. The first one is the SEM_MATCH which is a table function to query semantic data from the database. The second is the SEM_MODELS which is a table function to retrieve models name (RDF resource). To be clearer, a model represents a URL's content. In the case a user would like to query a URL, the application has to know the name of that model. With this function the application can get the name and apply it in the SEM_MATCH function to formulate the query in the right way.

In addition we have used two other procedures that are called from the application server side. The Enqueue PLSQL procedure is responsible to place an RDF resource in a queue for downloading its content to the database. The other procedure is used to formulate the query in Oracle SPARQL syntax and return a reference cursor with the results of the executed query back to the application server.

Finally we have use two tables that are responsible to keep the details of the users registered with the editor, and keep the details of the saved pipes for each user respectively. The first table keeps username, password, name, email of the users. It is used to give access to the users and also to know which pipes to load for each user. The second keeps the pipe name, the owner, the last modified date, the XML schema of the MashQL diagram that is saved, the description, and the published status of the pipe. It is used to retrieve the pipe's details (for display purpose), and also to provide the XML file of the pipe to be use and visualize the MashQL diagram in the editor's drawing area.

The interaction to the database is made from the libraries that the logic of the editor is. Also note that the call is made from the JSP libraries only. These libraries declare a JDBC connection and interact with the database to execute the queries by using the execution procedure available in the database, or save any details in the database tables.

As we already mention, we have use the Oracle's 11g Semantic technology to implement the back-end of the editor. So we have implemented a JDBC connection Java class in the application server, including the connection string that will gain access to the database. In the case that someone would like to mitigate to another server like SQL Server 2008 for example, the only thing he/she has to do is to change this class, appropriately so that it will include a connection string that will be recognized from the new database. In this version of MashQL we use Oracle because it allows us to use the semantic technology.

# Chapter 5

## Experiments/Evaluation

In this chapter we present two types of evaluations: 1) the usability of the MashQL editor, and 2) the time-cost of formulating a MashQL query and time-cost of loading RDF sources by using MashQL input module into oracle's semantic technology.

### 5.1 Experiment Setting

The evaluation is based on RDF sources existing in semanticweb.org public datasets. In these datasets there are information on papers that were presented, people who attended, and other things that have to do with the main conferences and workshops in the area of Semantic Web research.

The first experiment has to do with usability testing (section 5.2). We have used this dataset to make the tasks that our users will interact with the MashQL editor and get their comments. For this test we have ask 20 users, mainly having experience in using the Web for our experiment. The users also are divided into two groups: the I.T. and non I.T. people, so that we can say confidently if the editor can be used practically from both groups. With this experiment we have evaluated the usability of the editor by using different RDF sources from semantic web site. For the setup of the test, we use two PCs. The first is a 2.13GHz dual core CPU, 2GB RAM, 150GB HHD, on a Microsoft XP SP3 OS, and IE 7. The second is a 1.7 Centrino core, 512MB RAM, 40GB HHD, on a Microsoft XP SP2 OS, and IE 7.

Secondly, we have evaluated the time-cost on loading RDF sources and formulating a query by using the editor (section 5.3.). For this experiment we use four different RDF sources files of the semantic web dataset; a data source with content length 1MB, a source with content length 2M, a source of 5M, and the last with 10MB content length. Each of these sources are loaded into a separate RDF model in Oracle 11g, by using the MashQL editor. The Oracle is installed on a database server with 2GHz dual core CPU, 2GB RAM, 500GB HHD, on a 32-bit Unix OS. The editor was installed in an application server with 2.13 GHz dual core CPU, 2GB RAM, 150GB HHD, on a Microsoft Windows XP SP3 OS. We evaluate the response-time of loading and formulating a query by using RDF sources with 1MB, 2MB, 5MB, and 10MB content length respectively.

## 5.2 MashQL Usability

This usability test is intended to determine the level to which a user can complete routine tasks using the MashQL editor and also to quantify the effectiveness and efficiency of the editor's interface. The test was conducted with a group of potential users that know how to use and navigate in the web, and took place at the HPCL labs of the University of Cyprus.

The test took place from October 19th to October 30th, 2009. It involved assessing the usability of the application's interface design and information flow, with a special focus on efficiency (for example comparing the efficiency of information retrieval using the traditional mode of navigation and navigation via the editor).

20 participants were involved, participating in the test either as individuals or in groups of two. We also divided the participants into two groups: a) the I.T. group, which included people who had more experience in using the web and other web applications, and b) the non I.T. group, which included participants that had some basic knowledge of using the web. Each individual test lasted from 1:30 to 2:30 hours, depending on the skills of each participant.

All participants found the MashQL editor to have a clear purpose and 85% thought that the editor was easy to use, while 65% thought that the interface of the editor was attractive and efficient. However, the participants have recommended some changes concerning mostly the editor's user interface. Most of the users have recommended more training sessions and tutorial material on how to use the editor, in particular for the query formulation part. Also, a number of participants have recommended a better presentation of the gathered information (results visualization), and some extra functionality to export the results to a file or print them. Finally, the need for cross-browser compatibility of the editor was stated, since the editor was implemented only for Internet Explorer.

In the next sections we present participants' feedback, satisfaction ratings, completion ratings, time spent on tasks, preferences and recommendations for redesign. Copies of the questionnaires and scenarios that have been used throughout the usability test are presented in the Appendices.

### 5.2.1 Methodology

**Sessions**

The people that we have contacted for the usability test are from the University of Cyprus' student lists (including both undergraduate and postgraduate students) that have some experience and knowledge of using a browser and/or other web applications. After we prepared a list of 20 participants we separated them into groups of two or individuals. We chose to have small groups so that we could handle the participants better, observing their interaction (facial expressions etc.) and the difficulties they faced during the completion of the scenarios. We contacted with these people via the phone and arranged a date and time for the test. Each session lasted from 1:30 to 2:30 hours, which depended on each participant's skills.

Before presenting the scenarios, we formulated a 15 minute training session for the participants, to demonstrate the editor and present some query formulation examples. This would help them understand how to use the editor and what kind of functionality they could expect from it. We also presented the participants the web pages that they would use in the scenarios. After the short presentation and training session, we asked the participants to write down their first impressions of the editor.

During the test we explained the main purpose of the test session and asked the participants to fill out a questionnaire that included demographics and information about how much they use the PC and the internet, and what they usually use them for. Then we explained what they should do during the interaction scenarios. To help the participants, we provided them with a description of the test's purpose, and the URLs that they would use in the scenarios. Participants read the tasks and tried to gather the information from the website and then gather the same information via the editor.

After each task completion, the participants had to write down their comments and perceptions of usability. Main emphasis was on the efficiency dimension and more particularly the time needed to retrieve information and compare the typical navigation method of a website with the navigation method via the editor. After they had completed the scenarios, the participants had to evaluate the editor by answering 15 questions presented using a 5-point Likert, and 8 open-ended questions regarding their preferences and recommendations.

**Participants**

All participants were chosen so as to have some basic knowledge of how to use and navigate in a browser and how to collect information from a web site. As mentioned above, they worked either alone or in groups of two, and this usability test was conducted in a two week period schedule. Fourteen participants were male and six were female. All participants were randomly selected with ages ranging from 18 – 44. They came from different professions and had different skills. We present below a summary of the participants' professions and age ranges in tabular form:

Profession:

| Programmer/ Analyst | Merchant | Store Manager | DBA | IT | Student | Teacher | Other (engineers, chemic. Eng.) |
|---|---|---|---|---|---|---|---|
| 25% | 5% | 5% | 10% | 15% | 10% | 10% | 20% |

Age Range:

| 18-24 | 25-34 | 35-44 |
|---|---|---|
| 15% | 80% | 5% |

The participants also use the computer in their free time and at work (except for the students), and also use the internet. The average time per week spend in front of a home PC is 12.8 hours and 15.25 hours at work. All the participants use a PC with either Windows or Linux operating system. However, their preferences with respect to browsers differ. Most of the participants use Mozilla Firefox while others use Internet Explorer and Opera browsers. 75% of the participants use Firefox, 60% use Internet Explorer, and 5% of the participants also use Opera. 35% of the participants use both Internet Explorer and Mozilla Firefox.

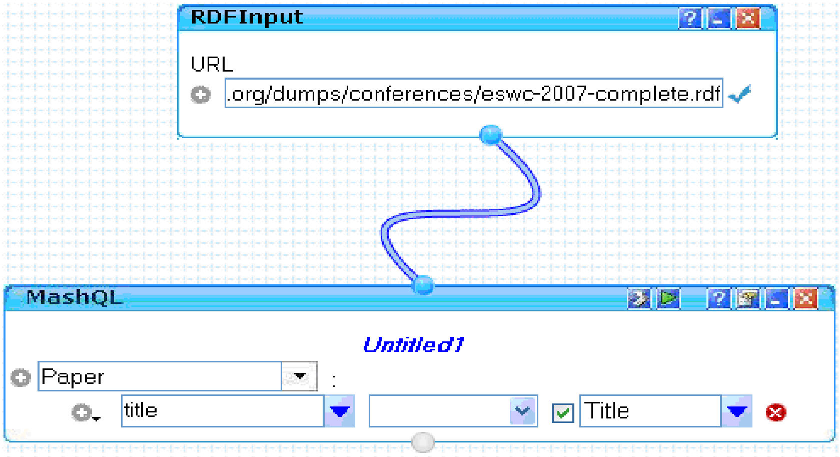
## Evaluation Tasks / Scenarios

Participants had to complete six scenarios with specific tasks which we will introduce next in this section. In all of the scenarios, we required the retrieval of information to be completed both by using web page navigation and formulating queries in the editor. The scenarios that participants had to complete were the following:

Scenario# 1:

Main task: ‘Find the titles of the papers presented in the 4th European Semantic Web Conference.’ This was an easy task, and was aimed at helping the participant learn and understand how to formulate a query in the editor.

The solution in MashQL is the following:



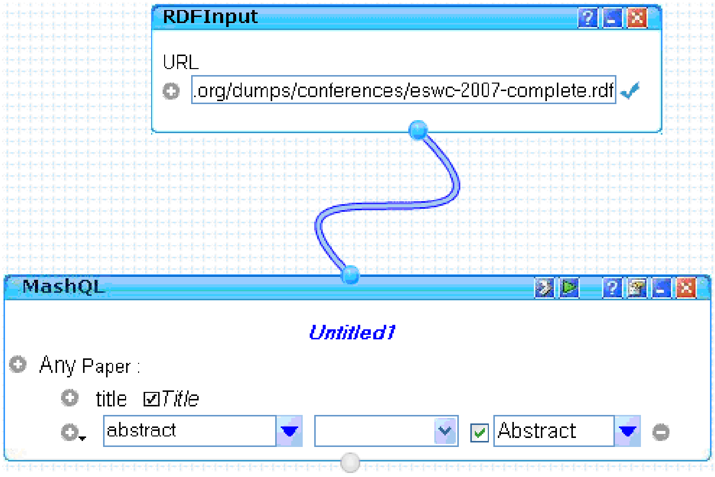
URL: .org/dumps/conferences/eswc-2007-complete.rdf

Paper :
title — ☑ Title

**Figure 34 - Scenario #1 MashQL solution**

Scenario#2:

Main task: ‘You want to learn more about the papers of the conference so you have to create a file that contains the titles and the abstracts of the papers presented in the 4th European Semantic Web Conference.’ This scenario is an extension of the first task, but this time the participants have to compare the ease of use of the typical web navigation and the navigation via the editor.



URL
.org/dumps/conferences/eswc-2007-complete.rdf

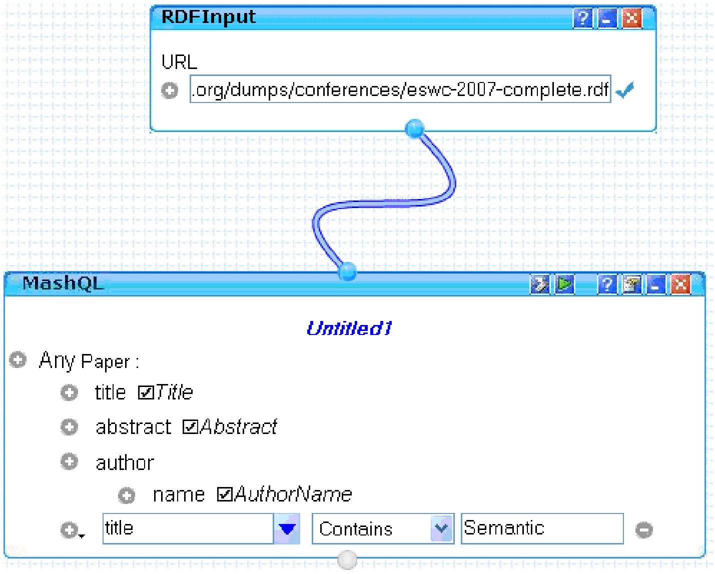
*Untitled1*

Any Paper :
title ☑*Title*
abstract ☑ Abstract

**Figure 35 - Scenario #2 MashQL solution**

Scenario #3:

Main task: ‘Retrieve all the titles, abstracts of the papers, and the author names for each paper from the 4th European Semantic Web Conference that have a title that contains the word ‘Semantic’.’ This task is even more difficult and time consuming than the previous ones, with more information to collect, in order to highlight the difference in usability and efficiency of the two methods.



URL
.org/dumps/conferences/eswc-2007-complete.rdf ✓

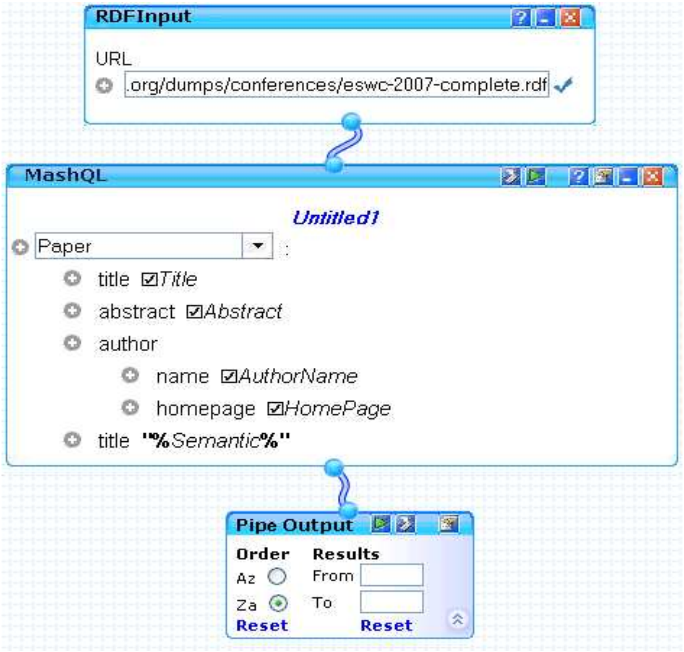
Untitled1

Any Paper :
- title ☑Title
- abstract ☑Abstract
- author
  - name ☑AuthorName
- title | Contains | Semantic

Figure 36 - Scenario #3 MashQL solution

Scenario #4:

Main task: 'Update the information from the previous scenario by retrieving also and the homepages of the authors, and also order the results.' This scenario emphasizes the ordering functionality of the editor compared to the manual ordering of the information, after gathering them in a file.



URL
.org/dumps/conferences/eswc-2007-complete.rdf ✓

Untitled1

Paper :
- title ☑Title
- abstract ☑Abstract
- author
  - name ☑AuthorName
  - homepage ☑HomePage
- title "%Semantic%"

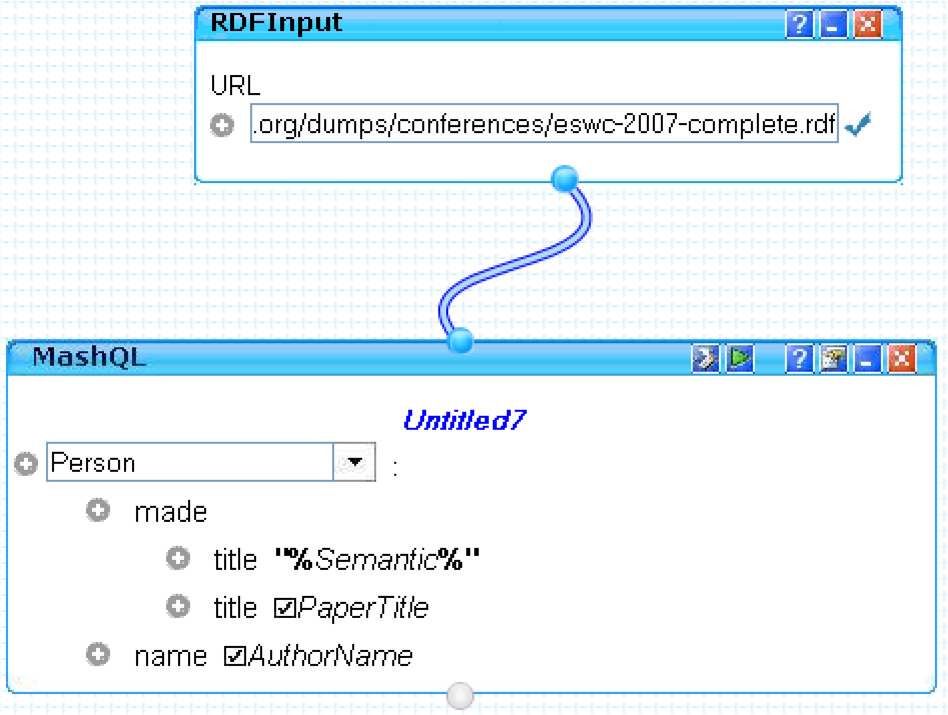
Order
Az ○
Za ●

Results
From
To

Figure 37 - Scenario #4 MashQL solution

Scenario #5:

Main task: 'Ask the participants to retrieve the names of all the authors of papers that contain the word 'Semantic' that were presented in the 4th European Semantic Web Conference. We want the authors' names and the titles of their papers.' This scenario is used to show that the editor gives the users the opportunity to extract the same information by formulating a different query. In the previous scenarios the participants start the formulation of query by using the word "Paper", but in this scenario they start by using the word "Person".



URL
.org/dumps/conferences/eswc-2007-complete.rdf

Person :
made
title "%Semantic%"
title ☑PaperTitle
name ☑AuthorName

**Figure 38 - Scenario #5 MashQL solution**

Scenario #6:

Main task: 'Retrieve the names and homepages of the authors that attended the 4th European Semantic Web Conference or the 16th International World Wide Web Conference, and then find all the authors' names that contain the word Thomas inside.'

In this scenario there is a combination of more than one source to gather information, so the participants have to compare the efficiency of each method by using more resources.

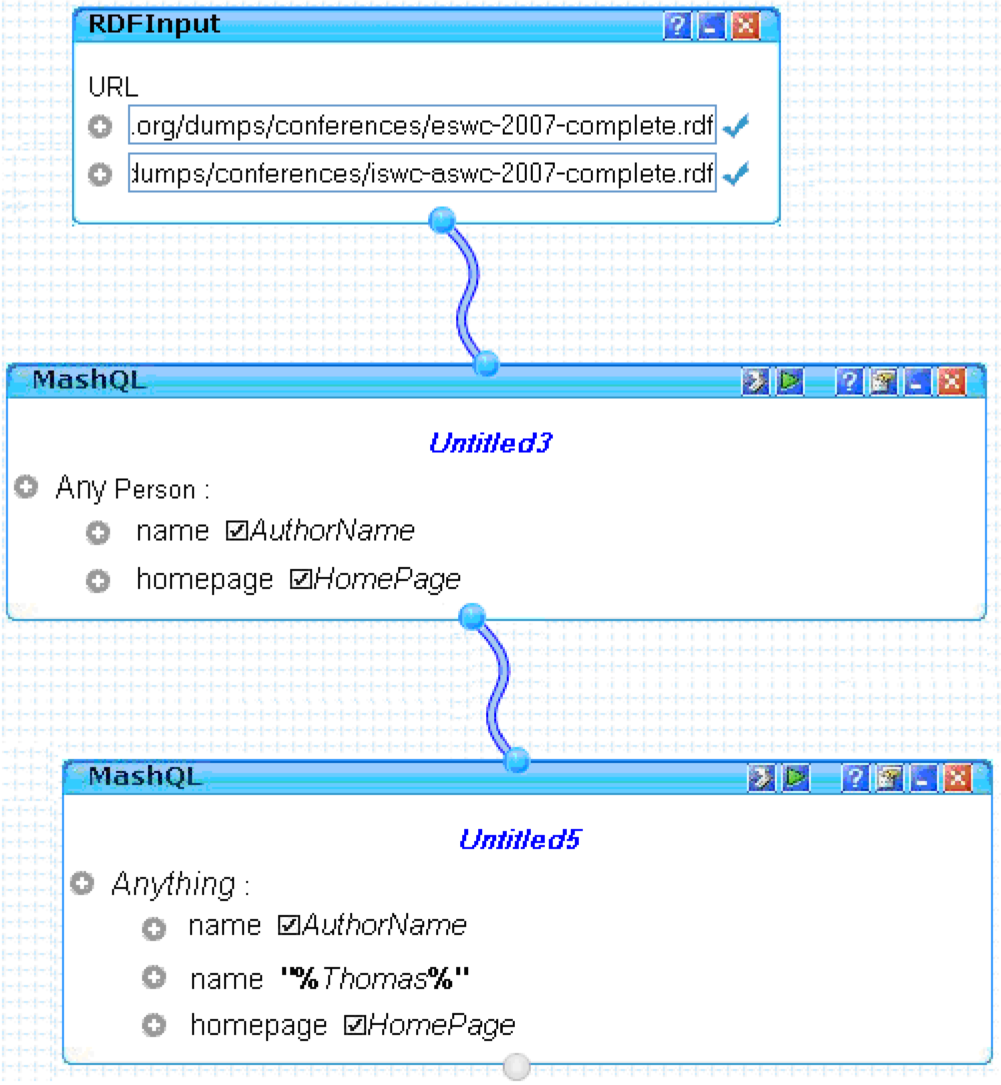


**Figure 39 - Scenario #6 MashQL solution**

The sources used in this test are the following:

1. The website of the 4th European Semantic Web Conference: http://data.semanticweb.org/conference/eswc/2007/html
2. The website of the 16th International World Wide Web Conference: http://data.semanticweb.org/conference/www/2007/html

Information about these two conferences is available in a structured way at the following URLs respectively (these sources were used in the editor):

1. For the 4th European Semantic Web Conference, the information is available at: http://data.semanticweb.org/dumps/conferences/eswc-2007-complete.rdf
2. For the 16th International World Wide Web Conference, the information is available at: http://data.semanticweb.org/dumps/conferences/iswc-aswc-2007-complete.rdf

’Structured’ in this case denotes that data is represented in an electronic way, where each piece of information has an assigned format and meaning, and is organized to allow identification and separation of its context from its content. These sources contain information on papers that were presented, people who attended, and other things that have to do with the main conferences and workshops in the area of Semantic Web research. It is also known as the Semantic Web Conference Corpus. In more detail, these sites contain information about Papers, Titles, Abstracts, Authors, Names, Affiliations, Related Documents and other information that has been used in the scenarios.

The main purpose of this usability test was to compare two ways of searching for information: a) by using the typical method of navigating (finding your way from page to page on the World Wide Web, clicking on websites’ links and looking for the required information) and b) the proposed method of searching for information using the MashQL editor.

## 5.2.2 Results

### 5.2.2.1 Time on Task – Web vs. Editor

Main emphasis here is on the measurement of the time each participant needed to accomplish each of the scenarios by using the browser and then by using the editor to collect the requested information.

Figure 40 illustrates the time the users spent on each task, by using these two methods of information gathering. The time spent for each scenario is measured by taking the average time of all the participants.

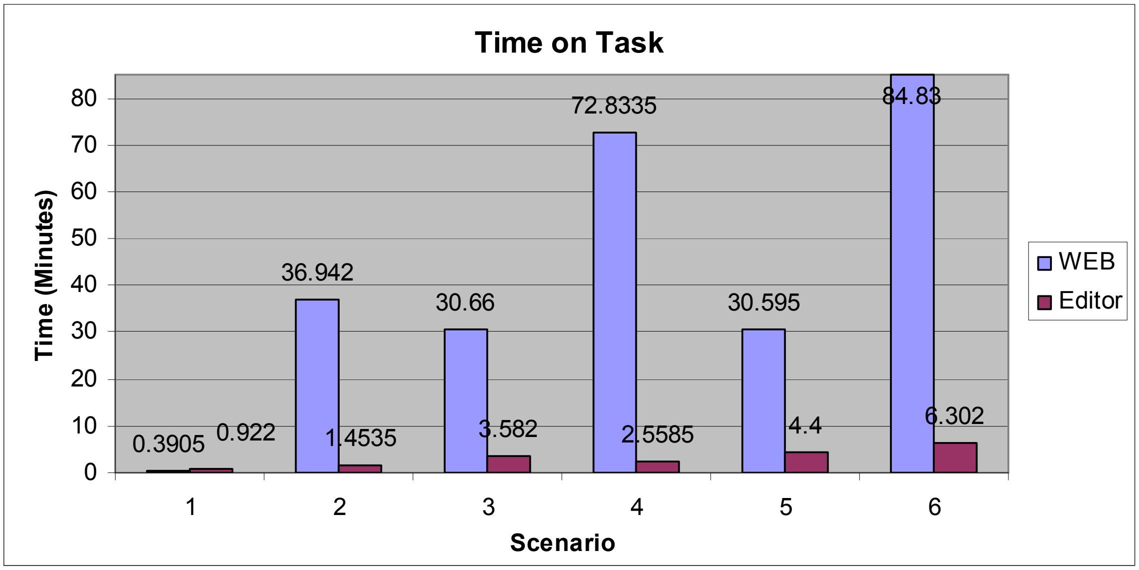


**Figure 40 - Comparison of time between browser & editor**

In this diagram the x-axis represents the scenario number and the y-axis represents the completion time in minutes. Inspecting this diagram more closely, we observe that in the first task the navigation time required to retrieve the information needed by using the browser is less than the time required using the editor. This is because the information is all available in one page, which makes things easier. However, for all other tasks, the information needed is spread across different webpages (this depends on the structure of the web site), so this makes the time for information retrieval by traditional navigation much greater than performing a query to extract the same information using the editor. In particular, if we observe the final task, the completion time for web navigation is nearly 13.5 times more than using the editor. This is because the final task involves two sources to collect the information from. This is time consuming using traditional methods, whereas when using the editor the participant has just to add the new URL source. Furthermore, the time needed for a participant to use the web in tasks 4 and 6 was too long, making users frustrated when retrieving the results from the browser. On the other hand, when users completed these two tasks using the editor, they were relieved that they could complete the task in much less time.

#### 5.2.2.2 Editor's Time on Task – I.T. vs. non I.T participants

The focus here is on the difference in time spent while performing the tasks using the editor between the participants in I.T. group and participants in the non I.T. group.

Our expectation was that the I.T. people would complete the task is less time than the non I.T people, and also that some non I.T. people would not understand the whole idea of the editor and consequently not complete most of the tasks. However, the results were better than we had anticipated, since everyone completed the tasks successfully. Non I.T people needed a little help, but this was due to the fact that none of the users had ever used this tool, or a similar application. Figure 41 shows the time spent on a task, using the editor, for participants of each category. The x-axis represents the task number and the y-axis represents the task completion time in minutes. To calculate the time on task for each scenario, we took the average time of all the I.T people and then the average time for all the non I.T people.

As we can observe from this graph, I.T. people completed each tasks in less time than the non I.T. people. This is because I.T. people already have experience with similar programs (applications that include query formalization), drag and drop mechanisms, queries, and SQL logic. As participants don't have knowledge of this particular technology, any previous similar experience relevant to the editor is helpful. The difference in time spent when comparing these two categories of participants is bigger when the task is more complicated. To explain this, if we observe the time required to complete tasks 1, 2, and 4, which were easy tasks, the difference is very small between the two groups. This is particularly evident in task 1 and 2 which were the easiest tasks. On the other hand, by observing the time required to complete tasks 3, 5, and 6, the difference between the two groups grows as the task becomes more difficult. Looking at task 6, which was the hardest task to complete, we can see that the difference in the time required for the two groups of participants approximately doubles. Of course this may be due to the fact that this was the first experience of these groups with the tool. If the users had already tried this

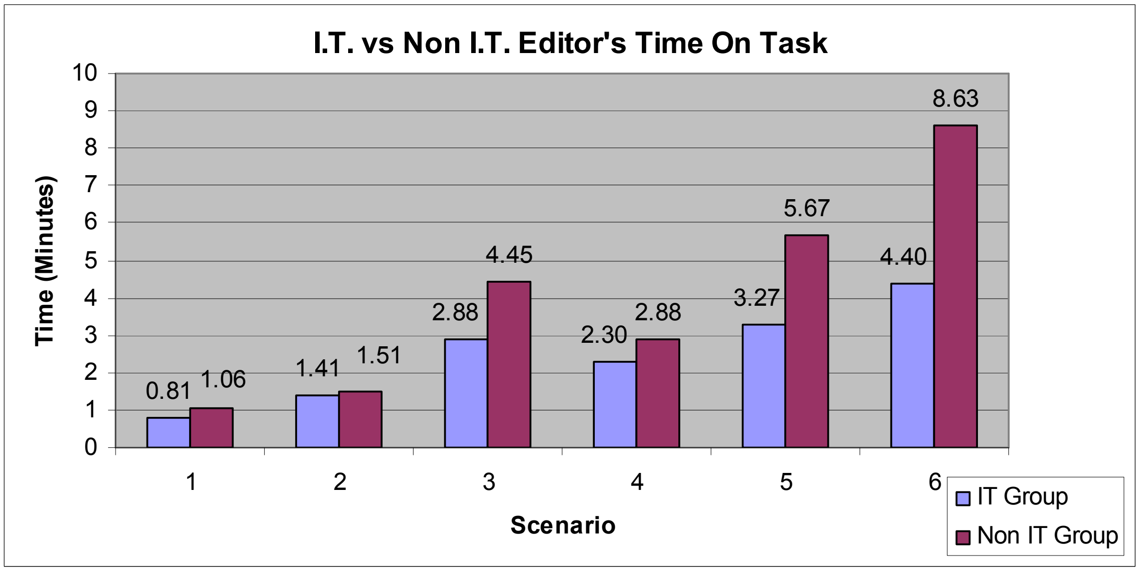


**Figure 41 - Time on Task of I.T and non I.T users**

tool before, the time difference might have been a lot smaller. Another observation about task 6 is that the time spent on the task by non IT people was approximately 8 minutes. However, the participants who took this long did not get frustrated by the difficulty of this particular task for two reasons: First, they completed the task 13.5 times faster than doing it via the browser, and second, they found it pleasant and fun due to the fact that they didn’t have to write down anything and everything was done visually.

#### 5.2.2.3 Characteristics

In figure 42 we present the qualities of the editor from the users’ point of view. To measure these qualities from the users’ perspective, we asked them to write at most three words that characterize the editor as a tool, focusing on the usability issues.

As we can observe from this graph, nearly all of our users find the editor convenient and easy to learn. Even with a short training time on the use of the editor and its functionality, the final conclusion is that it is easy to learn the editor and gain control of the whole process of formulating queries. In addition, more than half of the participants find the editor pleasant, with a usable interface that makes it easy to use, since it does not require the user to write down specific commands in order to query the input sources. Interaction is conducted through direct manipulation and this makes the users consider that they can construct and understand the queries that they want to formulate. In addition the efficiency for the users is related to time, as they can perform a potentially time consuming information search with a query formulation. This information retrieval is much faster than retrieving the same information using browser as we can see in figure 40.

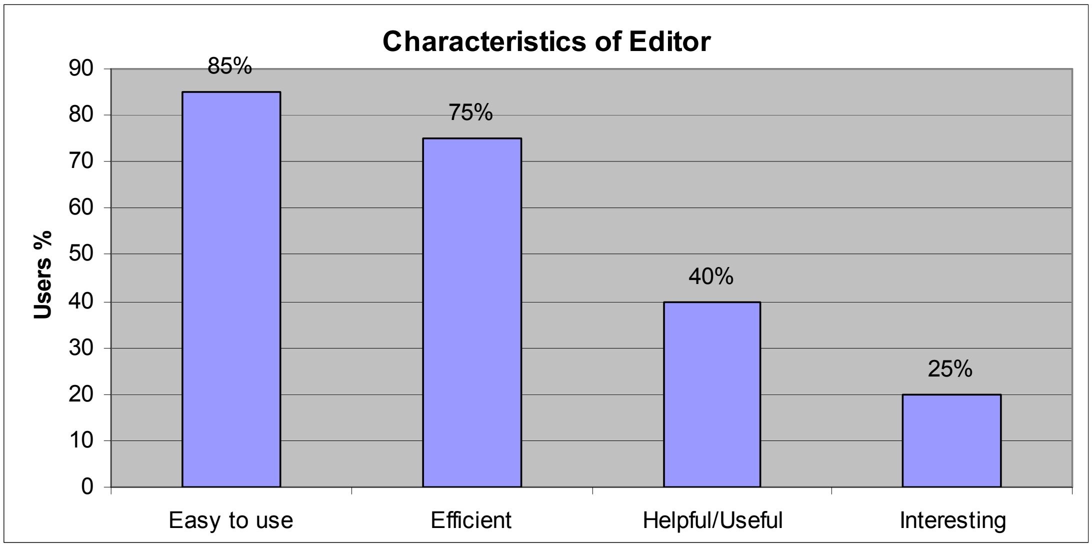


**Figure 42 - Characteristics of editor from the users**

Furthermore, 40% of our users characterize the editor helpful and useful, in the sense that they can practically use it for their needs. Some users said that they can use it to find the music they want, others to find restaurants with their specifications, and others for financial purposes. Along the same line, the 20% of the users find the editor interesting in the sense that this tool can offer something new in the web technology.

### 5.2.2.4 Overall Ratings

#### 5.2.2.4.1 Subjective Measures

We also provide a questionnaire to the participants, to evaluate the editor by answering 15 questions presented using a 5-point Likert Scale [31]. In these questions the participants had to rate the attractiveness, ease of use, environment, and whether they understand what they were doing with the editor. The rating may range from 1 to 5, with 1 denoting a poor rating and 5 denoting an excellent rating. Figure 43 is a bar chart showing the average of users' ratings for attractiveness, graphics, screen elements efficiency, readability, ease of use and finding information, interest for the tool, and how much they understand the purpose of the editor. The x-axis of the graph represents the rating from 1 to 5, and the y-axis represents the subjective measure that is rated.

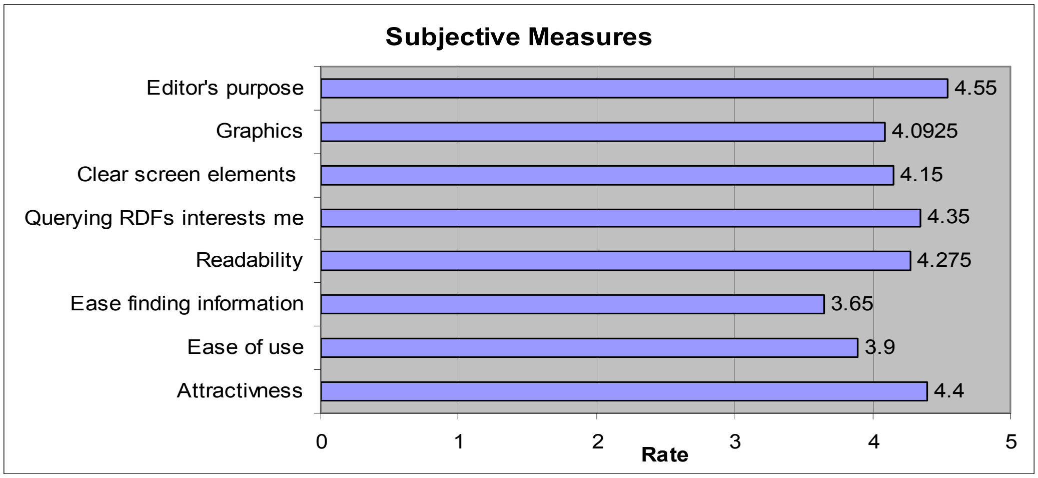


**Figure 43 - Users' rating for the editor**

The ratings for all the subjective measures were between 3 (Average) and 5 (Excellent). The ease of finding information and ease of use had ratings of 3.65 and 3.9 respectively, which lie between the average and good ratings. These ratings are relatively low due to the fact that users had little time to interact with the editor and had difficulties using it, especially at the beginning. They had to understand the logic behind how things work in the editor, so that they could better understand the whole mechanism. For that reason, users also have commented out that they needed more training, and more time to experiment and understand the application better. On the other hand, due to time limitations on our side, the usability test had to be completed fast. If the users had a whole day to experiment with the editor, then we believe that information finding and ease of use would have had have a better rating.

The other subjective measures are between 4 (good) and 5 (excellent) rating. The best rating was for the editor’s purpose, which shows that the participants understood what they were doing with the editor. They understood the philosophy behind the editor, which was to gather information in a diagrammatical way from some web pages which contain RDF structured data. Additionally another important measure, the interest of users in the technology, was also rated highly, with 4.35/5. They found this technology applicable to many situations. Each user was able to state an application of this technology concerning their needs.

Finally, the users found the editor very attractive, with nearly excellent graphics, the information easy to read, and the usage of each element of the editor (drop-down lists, auto complete functionality, buttons usage, drag and drop mechanism etc.) was fairly good enough to understand. The feeling that we got from the participants was that they were excited by the tool and that they felt it was very useful.

#### 5.2.2.4.2 User preferences and Recommendations

Next we will consider users' opinions after the experience they had using the editor for the scenarios described above. We will highlight what the participants like or dislike and consider their recommendations for this application.

##### 5.2.2.4.2.1 Likes & Dislikes

We have asked the participants to write at most three things that they like about the editor and three things that they didn't like much. In figures 44 and 45 we illustrate what users like and dislike respectively about the editor. In both graphs the x-axes represent aspects of the editor that the users commented on and the y-axes represent the percentage of participants that commented on an aspect.

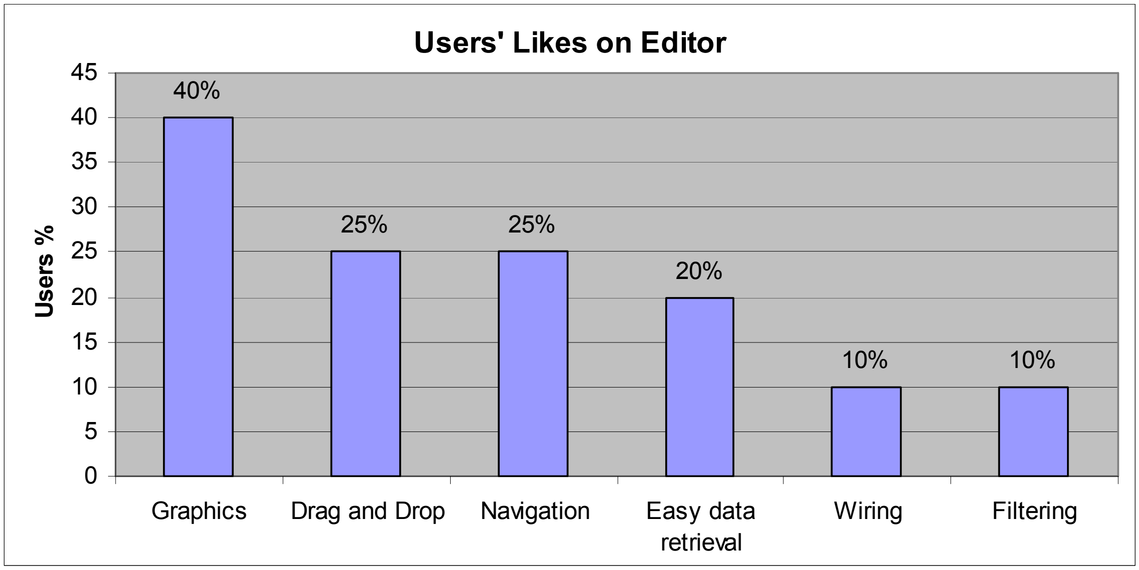


**Figure 44 - What users like in the editor**

In figure 44, we see that most of the participants liked the drag and drop feature that the editor provides. This allows users to drag a module (RDF Input, Query etc.) to the editor area so that they can add the URL sources and formulate the queries. Furthermore, they liked the editor's graphics, i.e. the colors, the buttons, the modules' design, and the presentation. The navigation inside the editor is easy, since everything is done with the mouse. The users can drag the module they want, formulate a query by only selecting values, add new restrictions in the query module by pressing the add button, change the properties of the pipe in an easy way, and finally access a help file by using the help button of each module, or by clicking the right mouse button and selecting the help link. Participants also liked the ease with which they could retrieve information from a source. The issue here was that the users needed more training sessions so as to be better able to understand the query mechanism and philosophy. Finally, users liked the wiring feature that is used to connect an input source with the query module in order to query a particular input source, and the filtering mechanism that is used in the query module (equals, contains, more than, less than, one of etc.), to filter the results depending on their needs.

We also had some comments from the participants about editor features that were still a little vague after the test. In figure 45 we present what they didn't like much while using the application.

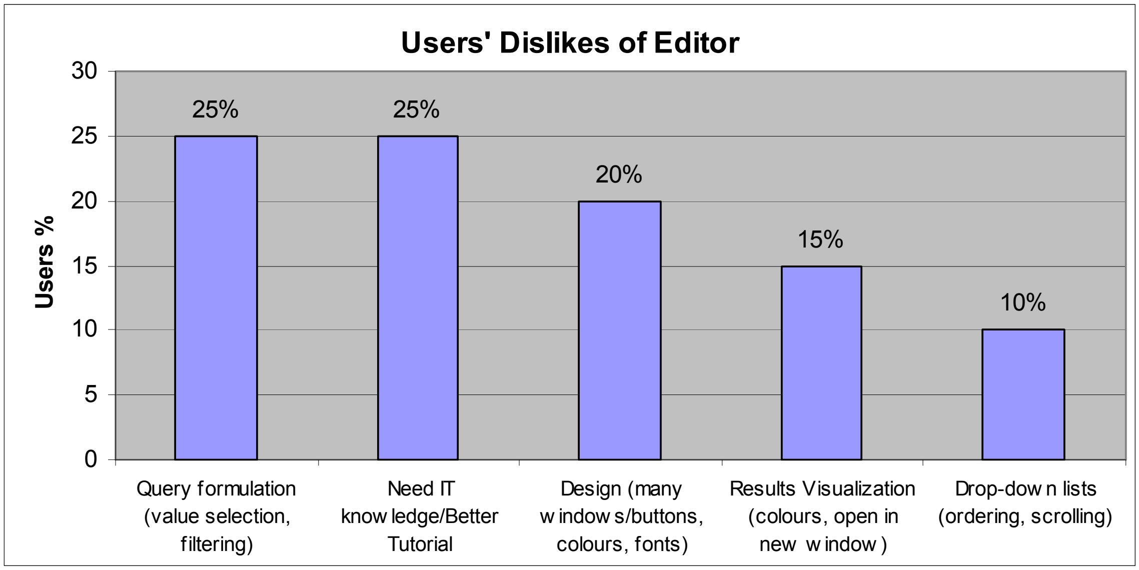


**Figure 45 - What users dislike about the editor**

Most of the comments that users made were mainly concerned with the limited time spent on training and interacting with the editor. These comments came mostly from users that were not in the I.T. group. They needed more time to experiment with the features and how the editor works so that they could accomplish the tasks easier. Consequently, 25% of the participants did not fully understand how the query module works, with particular problems with filtering the results. In addition, the same percentage of users pointed out the need for more training and interaction time. 20% of participants pointed out that there were many windows in the editor, and many buttons in each of the modules. However, this is inevitable since for each different query a participant has to use a new module and this consequently fills up the editor area. Finally, some participants didn’t like the way that the information was presented in the debugger area (results visualization). This had to do with the colors used. The truth is that we haven’t dealt much with the presentation of the results, because for this version of MashQL we just wanted to show information in table format only. In the future we will concern ourselves more with this issue as we point out in the ‘future work’ section.

**5.2.2.4.2.2 Recommendations**

We also asked participants to write down one significant recommendation that they believe is worth considering to improve the editor. As we can see in figure 47, most of the participants recommended modifying the way in which the results are presented. In detail, they would like the editor to have a print button, so that they can print the information, and an export button, to allow them to export the retrieved information in a file. They would also like to change the colors used in the visualization of the results. Additionally, the participants recommended a scaling mechanism (resize of the windows in the editor area), so that they could have more space in the editor area. In this version, for representing of the resize for the windows in the editor we have a minimize/maximize button at the top-right menu of each module.

A smaller number of participants recommended reducing the number of buttons in the modules’ menu, and also making the execute button bigger so that they could locate it easier, since this button is the one they use the most. Here as the figure 46 shows, we have to say that all the buttons are helpful, especially in the query modules. We have the

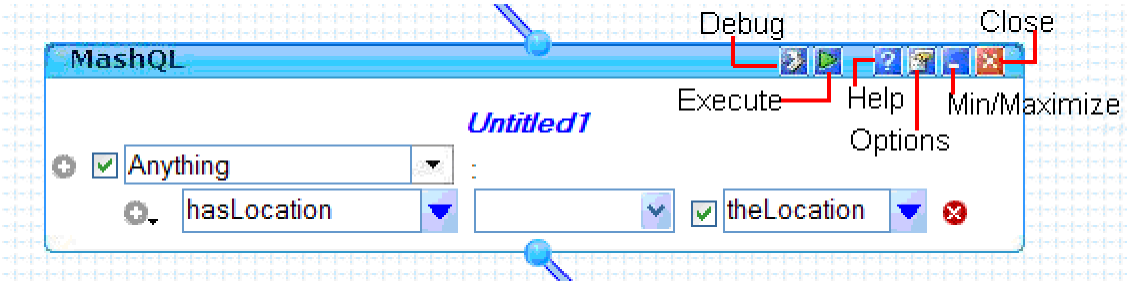


**Figure 46 - Buttons in the modules' menu**

close button, the minimize button to free up space in the editor, the help button to find information about the module, the query options button to find the formulated query in SPARQL for users to understand query syntax and, in a separate location, we have the execute and debug buttons. We think that those buttons are all useful for the users. With more interaction with the editor, users will probably feel more confident with these buttons.

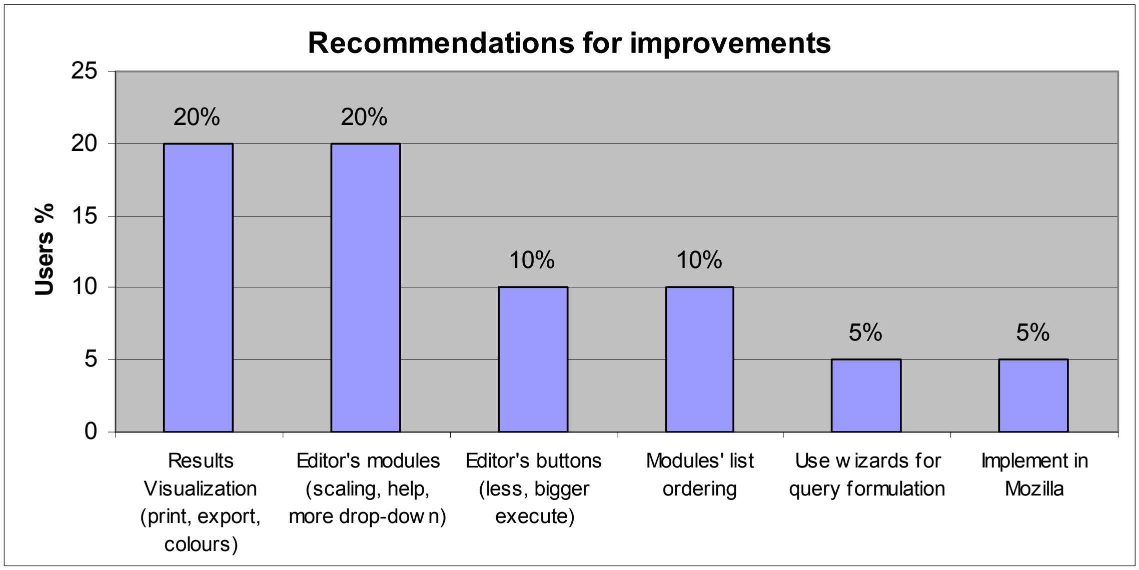


**Figure 47 - What users would like to recommend for the editor**

Finally, in this graph we include two other recommendations that participants pointed out and we believe are worth considering. The first recommendation is that users should be able to formulate a query through wizards and the second is that they should be able to use the editor with the Mozilla Firefox browser. The first one was pointed out by a non I.T. participant who believes that this will make the life of a user easier, and the second one was pointed out by an I.T. participant who only uses Mozilla Firefox. The use of wizards might not be the best idea for a web application whose purpose is to query a schema-free source, but this is a good idea to hear from a non I.T. user. From this comment we can see that the participant has fully understood what he/she was doing with the editor and thought of a better process to achieve the same results. The use of the editor in other browsers is under consideration at the moment, since 80% of the implementation is in JavaScript and 5% in AJAX and each of these technologies have different methods for each browser. So, it is not an easy task to modify the editor to work in each browser.

### 5.2.3 Conclusion

Most of the participants find the editor to be a well-structured and organized web application, with a usable interface, which is easy to use and learnable, as well as attractive in terms of aesthetics. Comparing an information search conducted with a typical navigation method (using a web browser) and a search conducted formulating a query in the editor shows that the editor is much faster and efficient than the web except for one scenario. The participants also argued that the editor is very useful, with a clear purpose and they express their intention to use it for their own searches over the web in the future.

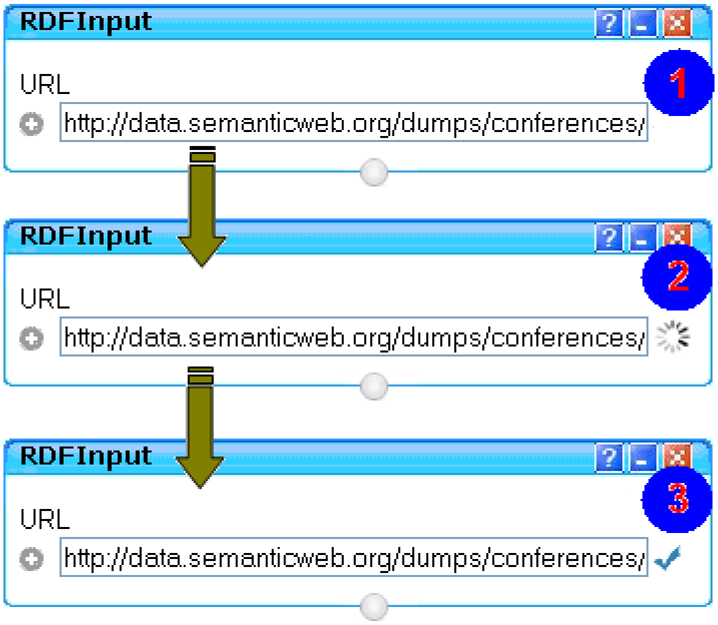
## 5.3 MashQL Response-Time Evaluations

In this section we are going to evaluate the response-time of loading an RDF source, and then evaluate the response-time of the background queries when formulating a query in the

MashQL editor. We are going to use RDF sources with content length 1MB, 2MB, 5MB, and 10MB respectively. By evaluating these scenarios, we will know the potentials of our editor and also know where we can modify and evolve our editor in a future version.

We use the MashQL editor's RDFInput module as illustrated in figure 48, to set the URL's of the 4 RDF input sources, so that we can measure the time till they are downloaded in the oracle database. We assume that the RDF source is set for the first time. In the case that the URL of the source is found in the downloaded URLs in the database, the source will be ready for querying it instantaneously and no downloading is needed.

When we set the URL in the text-field of an RDFInput module, we have implemented a JavaScript function that starts a timer. Meanwhile, the content of the RDF source starts downloaded (indicated with the AJAX loading icon, figure 8 point 2) in the oracle database in a new RDF model. Every six seconds there is a procedure that checks the downloading status of the source. While the status of the downloading is not 100 or 400 (source downloaded successfully), the procedure still requesting for the status. The timer stops on status 100 or 400 (indicated with check icon, figure 8 point 3) and returns a time in milliseconds. This time indicates the total time needed for a source to be downloading in our database, and the time needed to be ready for querying it.



URL http://data.semanticweb.org/dumps/conferences/

URL http://data.semanticweb.org/dumps/conferences/

URL http://data.semanticweb.org/dumps/conferences/

**Figure 48 - RDF source downloading process**

By using this mechanism we have calculated the response-time for each RDF source to be downloaded and get ready to be used in a query formulation. In figure 49 we illustrate the results of this experiment. In this point we have to clarify that the content length of each source that we have use in this experiment are close to the one indicated in our context length examples (i.e.: 1MB, 2MB, 5MB, and 10MB respectively).

In this point we have to say that the downloading of these sources is done from Cyprus. However all the RDF resources are placed in US. This slows down the downloading time since the downloading speed and bandwidth is less than the connections available in US. In the case that a user located in US tries to download an RDF resource from the resources that we have use in our examples, it will be downloaded faster then attempting the same scenario from Cyprus. For our experiment we have downloaded the sources which are located in US from the server we have in Cyprus. So the results that we will present are focused with the network infrastructure that we have here. In any other case, the results would not be representative to the actual.

As we can observe in this graph, the time needed for an RDF source of content length 1MB, 2MB, and 5MB is defined acceptable for the user to wait. When we use a 10MB source to use for query formulation, the response-time for downloading is close to 11 minutes. In our opinion this response-time is way too far from the acceptable time limits. In other words, the user is not expected to wait for so long for a source to be downloaded.

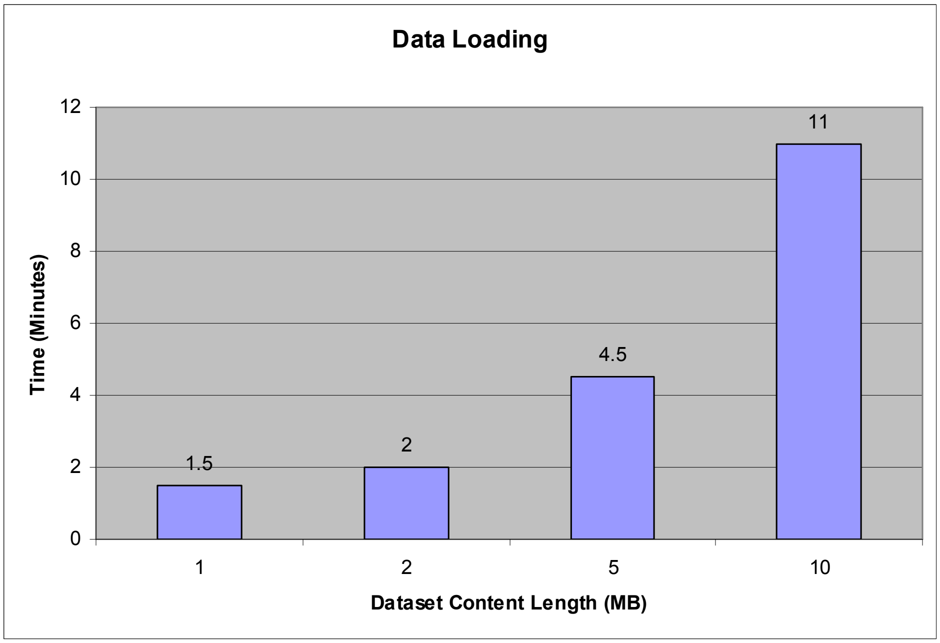


**Figure 49 - Time response for downloading content**

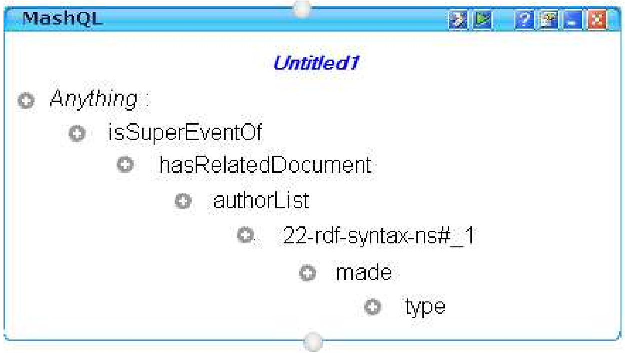
When downloading an RDF source of approximately 1MB the time needed is 1.5 minutes. RDF source with 2MB content length needs 2 minutes for downloading, RDF source with content length 5MB needs 4.5 minutes, and RDF source with 10MB content length needs 11 minutes. We also notice that the downloading time, expect from the content length, is also affected by the number of triples that each source has. For example a 1MB RDF source may contain 10,000 triples. The downloading time will be a bit faster than downloading a 1MB RDF source that contains 15,000 triples. We have observed in the RDF models, that this difference in the number of the triples is affected mostly on the length of the

O (object) element of the triples (remember that the triples in an RDF source are in the (S P O) format). The measures for each RDF source that we have in this graph is the average of downloading 10 RDF sources for each case, so that we cover RDF sources containing any number of triples and be more accurate to our results.

To conclude with the response-time for downloading an RDF source experiment, the acceptable in time range downloading content length is any source that has content length till 5 – 6 MB long.

In the second experiment we will evaluate the response-time of the MashQL editor's user interaction. We are interested in evaluating the execution of the queries that the editor performs in the background to generate the drop-down lists, rather than the query it self. We have use three kinds of queries that can be formulated in the editor, and evaluate them by using 1MB, 2MB, 5MB, and 10MB content length of RDF sources respectively, after we loaded them in the loading evaluation experiment. We have use the Oracle 11g Semantic Technology to do this experiment.

We have formulated a query which is expanded till branch 6 in the query module. Figure 50 illustrates the query used for this experiment.

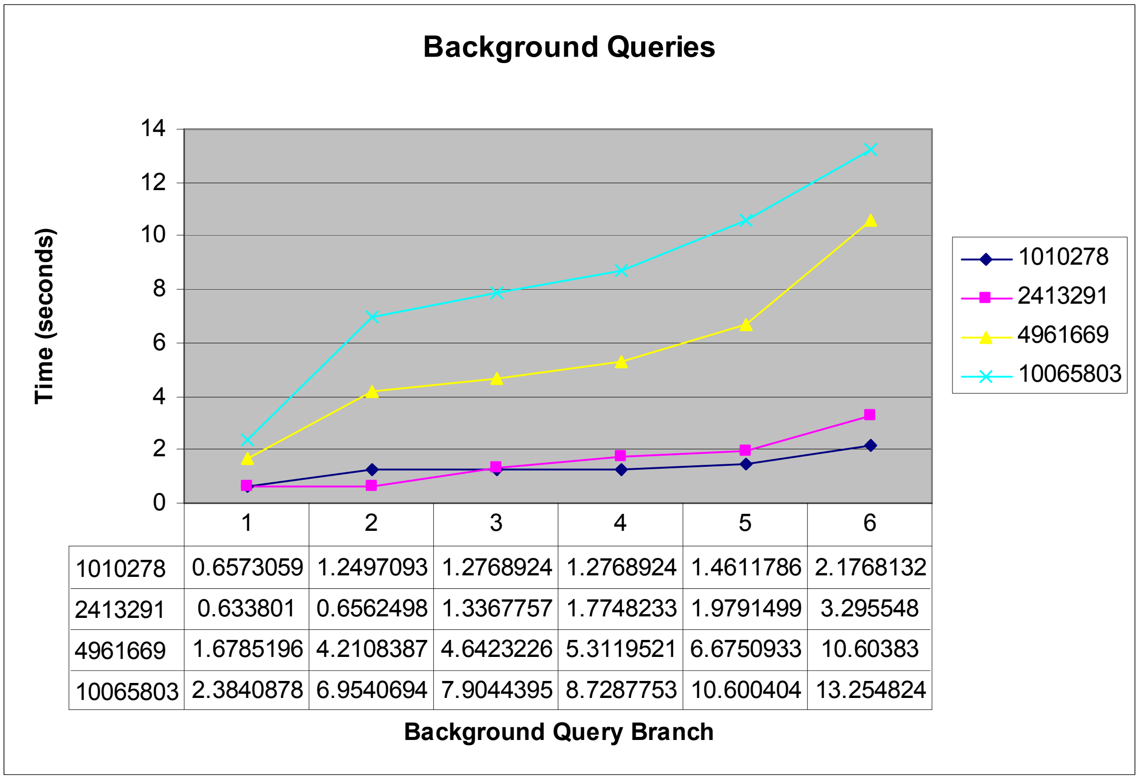


*Untitled1*

*Anything* :
isSuperEventOf
hasRelatedDocument
authorList
22-rdf-syntax-ns#_1
made
type

**Figure 50 - Query formulation in MashQL**

In this experiment we have set the values in each of the predicate till branch 6. We try this with RDF sources which have content length ~1MB, ~2MB, ~5MB, and ~10MB content length. In figure 51 we illustrate the response-time in each of the query branch for each RDF source's content length.



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1010278 | 0.6573059 | 1.2497093 | 1.2768924 | 1.2768924 | 1.4611786 | 2.1768132 |
| 2413291 | 0.633801 | 0.6562498 | 1.3367757 | 1.7748233 | 1.9791499 | 3.295548 |
| 4961669 | 1.6785196 | 4.2108387 | 4.6423226 | 5.3119521 | 6.6750933 | 10.60383 |
| 10065803 | 2.3840878 | 6.9540694 | 7.9044395 | 8.7287753 | 10.600404 | 13.254824 |

**Figure 51 - Experiment 1 graph**

The RDF source with content length ~1MB brings the values in the lists pretty fast till branch 6. The slowest execution of the background query is in branch 6, which takes 2.17 seconds. The same stands and for the RDF source with content length ~2MB, with branch 6 background query execution close to 3.29 seconds. As the content length grows, we can see that the execution of the background queries becomes slower, even in an early stage (1st and 2nd branch). In other words, the only acceptable execution time for background query for RDF sources ~5MB and ~10MB content length is only at the first branch. After the 1st branch the execution is slow and becomes extremely slower especially at branch 5 and 6. So the ideal content length to be used in the MashQL editor is till 2-3MB long.

To conclude, as shown by these two experiments, we have illustrated the limits of MashQL regarding the maximum content length to be used when setting a URL of an RDF source to be downloaded, and the maximum content length that can be used to formulate a query in an acceptable response-time for the user. We have concluded that the ideal RDF source for downloading is till 5-6MB long, and the ideas content length for formulating a query is till 2-3MB long.

# Chapter 6

## Conclusions and Future Work

This chapter contains the concluding arguments on building such a system and presents what future work could be done towards this direction.

### 6.1 Conclusions

In this work we have propose a query-by-diagram language called MashQL in order to allow building data mash-ups easily. MashQL is user-friendly for non-IT people and also allows querying and navigating RDF sources without having to know the schema or any technical details of the data sources.

We have presented a new query formulation approach that allows people to mash-up and query structured data without any prior knowledge of the schema, vocabulary, structure, and technical details of the datasets. The MashQL is easy to learn as it is close to the “logic” and natural language that people use when asking questions. Also it enables semantic pipes for mashing up RDF data easily though the user-friendly module designs.

MashQL is not simply an interface of SPARQL, but also it can be used as a general query language by its own. It also allows people to experiment a bit in the SPARQL query that is generated from the MashQL query formulation, by changing some values in the query, so that to give a flavor of the syntax of SPARQL.

We have use Oracle’s 11g semantic technology as a back-end of the implementation not only because of its scalability, but also because Oracle’s SPARQL inherits all the functionalities of SQL, including aggregation and grouping function, which are not supported by the pure SPARQL. Note that pure SPARQL is translated to Oracle’s SPARQL, because we are convinced that by using oracle we can handle large datasets.

MashQL is a powerful tool that allows caching remote sources, materializing query results, distributing queries, and publishing among multiple users. However, it still has open issues to be explored and implemented.

## 6.2 Future Work

There is a lot of work that can be done in order to improve the developed system in various aspects.

One of the major parts of the editor is the background queries. Without them the system will be nothing. We aim to improve the mechanism that executes and fill the lists of the triple patterns in the viewpoint of time-cost. This is feasible since the back-end of our implementation is supported by Oracle 11g semantic technology, which is made to handle this type of datasets that we are currently using. This is an advantage in our implementation, since this technology allows us to make any modifications to our code, so that to improve the performance and reliability of our mechanism, and also make things faster.

As we already pointed out, the implementation except of RDF sources will accept and other type of inputs. We refer to XML, CSV, RSS, ATOM, and Microformat input sources. These language component controls have been implemented in the design level inside the MashQL editor. However, the whole procedure and the logic of each of these kinds of inputs is still an open issue to consider. For now we assume that these modules will work with the same way as we have implemented the algorithm for the RDF sources, so that we have something to start with.

Similarly we have developed the design of the User Input module that its task was to declare the data type of a variable and the values refer to it. In this version of MashQL editor we have 10 types of object filter functions (equals, contains, between, more that, less than, one of, and the negation of those). The “OneOf” and “NotOneOf” object filter functions are been implemented in a query’s restriction by setting the values separated by commas. Our task now, is to transfer this logic that we have from this format to the User Input logic. In other words, the responsibility of these two functions will be up the User Input module. In the case that the function will be one of these two, there will be a terminal so that the user can connect this particular query’s restriction to the User Input module. In this module the user will set the data type and the values of that type, in order to complete the restriction.

Concerning the SPARQL that we use in our implementation, we plan to modify our functionality to accept language tags like “Person”@En, or “Άτομο”@Gr and data types tags like “1”^^xsd:integer, or “2009-01-01”^^xsd:date for the object literals. This is not an important issue to consider, but it will be nice to have some more advance flavor in our implementation.

In addition we plan to extend our query formulation algorithm by allowing unions between the triples atoms. In other words the algorithm will handle cases of subjects, predicates, or objects unions. For example in a query formulation in this version of MashQL we can formulate this: *Select anything that has year more than 2007*. In this example we have a subject S, a predicate “Year”, object-filter function “MoreThan” and object value “2007”. By implementing a union we may have: *Select anything that has year or puplicationYear more than 2007*. In here, we have a subject S, a predicate “Year/PuplicationYear”, and so on. The predicate “Year” and “PuplicationYear” may be from a different source, but they have the same meaning.

Also the results displayed in the debugger when executing a pipe or query could be visualized in a more elegant way than in this version. We can also show some useful metadata of each row rather than only show the actual returned values. Also it’s a good idea in the future in the cases that we have sources that includes images or any other type of multimedia, to show these multimedia images, or links to the multimedia sources and their metadata in the results inside the debugger.

Finally we plan to extend the system in other browsers. This version is running only under Internet Explorer 6+. The reason why is not working properly in other browsers is that the JavaScript and AJAX have a different object, declarations, and functionality in each browser. So the code has to be modified in order to support more browsers.

# Glossary

## A

**AJAX**: Short for Asynchronous JavaScript and XML, it is a term that describes a new approach to using a number of existing technologies together, including the following: HTML or XHTML, Cascading Style Sheets, JavaScript, the Document Object Model, XML, XSLT, and the XMLHttpRequest object. When these technologies are combined in the Ajax model, Web applications are able to make quick, incremental updates to the user interface without reloading the entire browser page.

**ATOM**: The name Atom applies to a pair of related standards. The Atom Syndication Format is an XML language used for web feeds, while the Atom Publishing Protocol (AtomPub or APP) is a simple HTTP-based protocol for creating and updating web resources. Web feeds allow software programs to check for updates published on a web site. To provide a web feed, a site owner may use specialized software (such as a content management system) that publishes a list (or "feed") of recent articles or content in a standardized, machine-readable format. The feed can then be downloaded by web sites that syndicate content from the feed, or by feed reader programs that allow Internet users to subscribe to feeds and view their content. A feed contains entries, which may be headlines, full-text articles, excerpts, summaries, and/or links to content on a web site, along with various metadata.

**B**

**Beans class (Java Beans)**: EJB component. Session and entity bean classes implement the bean's business and life-cycle methods. The bean class for session and entity beans usually doesn't implement any of the bean's component interfaces directly.

**C**

**CSV**: Comma separated values (CSV) file is used for the digital storage of data structured in a table of lists form, where each associated item (member) in a group is in association with others also separated by the commas of its set.

Each line in the CSV file corresponds to a row in the table. Within a line, fields are separated by commas, each field belonging to one table column. Since it is a common and simple file format, CSV files are often used for moving tabular data between two different computer programs, for example between a database program and a spreadsheet program.

**D**

**Deep Web**: (also called Deepnet, the invisible Web, dark Web or the hidden Web) refers to World Wide Web content that is not part of the surface Web, which is indexed by standard search engines.

**E**

**EER**: (Extended Entity-Relationship Model). It's based on ER (Entity Relationship diagram). is a language for definition of structuring (and functionality) of database or information systems. It uses inductive development of structuring. Basic attributes are assigned to base data types. Complex attributes can be constructed by applying constructors such as tuple, list or set constructors to attributes that have already been constructed.

**F**

**Function**: is a sequence of code which performs a specific task, as part of a larger program, and is grouped as one or more statement blocks.

**Firebug**: It’s extension for Mozilla Firefox allows the debugging, editing, and monitoring of any website's CSS, HTML, DOM, and JavaScript, and provides other Web development tools.[1] It also has a JavaScript console for logging errors and watching values, as well as a "Net" feature which monitors the amount of time in milliseconds it takes to load and execute scripts and images on the page.

**FOAF**: Friend of a friend (FOAF) is a phrase used to refer to someone that one does not know well — literally, a friend of a friend. In here we use it as an example RDF source.

**G**

**GUI**: A program interface that takes advantage of the computer's graphics capabilities to make the program easier to use. Well-designed graphical user interfaces can free the user from learning complex command languages. On the other hand, many users find that they work more effectively with a command-driven interface, especially if they already know the command language.

## H

**Hierarchy**: is the mapped relationship of sub- and super classes.

**HTML**: Hypertext Markup Language is the authoring software language used on the Internet's World Wide Web. HTML is used for creating World Wide Web pages.

## I

**Identifier**: In metadata, an identifier is a language-independent label, sign or token that uniquely identifies an object within an identification scheme. We use it in MashQL, to distinct the types of subjects, predicate, or objects to retrieve the appropriate values.

## J

**Java**: is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities.

**JavaScript**: A type of programming which can add interactivity and function to a web site. Some examples include drop down menus navigation button effects, interactive forms, slide shows, and pop open windows. There are many applications available to enrich a web site.

**JENA**: is an open source Semantic Web framework for Java. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract "model". A model can be sourced with data from files, databases, URLs or a combination of these. A Model can also be queried through SPARQL.

**JSON**: short for JavaScript Object Notation. It's a lightweight computer data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects).

**JSP**: Java Server Pages. The JavaServer Pages (JSP) is an extension of the Java Servlets Technology that is permitting the combination of HTML/XML with Java scriptlets to produce pages with dynamic content.

**K**

**L**

**M**

**Mashup**: s a web page or application that combines data or functionality from two or more external sources to create a new service. The term mashup implies easy, fast integration, frequently using open APIs and data sources to produce results that were not the original reason for producing the raw source data.

Metadata

**Method**: is a section of a program that contains code instructing the computer to take some action, a method is a predefined programming procedure or a named sequence of statements that perform some task.

**MicroFormat**: A microformat is a web-based approach to semantic markup that seeks to re-use existing XHTML and HTML tags to convey metadata and other attributes. This approach allows information intended for end-users (such as contact information, geographic coordinates, calendar events, and the like) to also be automatically processed by software. Although the content of web pages is technically already capable of "automated processing," and has been since the inception of the web, such processing is difficult because the traditional markup tags used to display information on the web do not describe what the information means. Microformats are intended to bridge this gap by attaching semantics, and thereby obviate other, more complicated, methods of automated processing, such as natural language processing or screen scraping. The use, adoption and processing of microformats enables data items to be indexed, searched for, saved or cross-referenced, so that information can be reused or combined. Current microformats allow the encoding and extraction of events, contact information, social relationships and so on. More are being developed. Version 3 of the Firefox browser, as well as version 8 of Internet Explorer are expected to include native support for microformats.

**Microsoft Script Debugger**: is relatively minimal debugger for Windows Script Host-supported scripting languages, such as VBScript and JScript. Its user interface allows the user to set breakpoints and/or step through execution of script code line by line, and examine values of variables and properties after any step. In effect, it provides a way for developers to see script code behavior as it runs, thus eliminating much of the guess-work when things don't quite work as intended.

**N**

**Normalization**: (URI Normalization). Converts URLs into their canonical form

**O**

**Object**: a concrete realization of a class that consists of data and the operations associated with that data. It's an item that a user can manipulate as a single unit to perform a task. An object can appear as text, an icon, or both.

**Oracle 11g semantic technology**: oracle's support for semantic technologies, specifically Resource Description Framework (RDF) and a subset of the Web Ontology Language (OWL). These concepts are implemented in Oracle.

**ORM**: (Object Role Modeling) is a fact-oriented method for performing systems analysis at the conceptual level. The quality of a database application depends critically on its design. To help ensure correctness, clarity, adaptability and productivity, information systems are best specified first at the conceptual level, using concepts and language that people can readily understand.

**OWL**: (Web Ontology Language) is a markup language for publishing and sharing data using ontologies on the Internet. A vocabulary extension of the Resource Description Framework (RDF), OWL represents the meanings of terms in vocabularies and the relationships between those terms in a way that is suitable for processing by software.

**Query**: is the primary mechanism for retrieving information from a database and consist of questions presented to the database in a predefined format. Many database management systems use the Structured Query Language (SQL) standard query format.

**P**

**Property**: is a named attribute of an object. Properties define object characteristics such as size, color, and screen location, or the state of an object such as enabled or disabled.

**Q**

**Query**: An object that requests information from a database and creates a dataset of the requested information

**R**

**RDF**: The Resource Description Framework (RDF) is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources; using a variety of syntax formats.

**RDFa**: RDFa (or Resource Description Framework - in - attributes) is a set of extensions to XHTML that is now a W3C Recommendation. RDFa uses attributes from XHTML's meta and link elements, and generalises them so that they are usable on all elements. This allows annotating XHTML markup with semantics. A simple mapping is defined so that RDF triples may be extracted. The W3C RDF in XHTML Taskforce is also working on an implementation for non-XML versions of HTML. The primary issue for the non-XML implementation is how to handle the lack of XML namespaces.

**RSS**: RSS (most commonly translated as "Really Simple Syndication" but sometimes "Rich Site Summary") is a family of web feed formats used to publish frequently updated works—such as blog entries, news headlines, audio, and video—in a standardized format. An RSS document (which is called a "feed", "web feed", or "channel") includes full or summarized text, plus metadata such as publishing dates and authorship. Web feeds benefit publishers by letting them syndicate content automatically. They benefit readers who want to subscribe to timely updates from favored websites or to aggregate feeds from many sites into one place. RSS feeds can be read using software called an "RSS reader", "feed reader", or "aggregator", which can be web-based, desktop-based, or mobile-device-based. A standardized XML file format allows the information to be published once and viewed by many different programs. The user subscribes to a feed by entering into the reader the feed's URI – often referred to informally as a "URL" (uniform resource locator), although technically the two terms are not exactly synonymous – or by clicking an RSS icon in a browser that initiates the subscription process. The RSS reader checks the user's subscribed feeds regularly for new work, downloads any updates that it finds, and provides a user interface to monitor and read the feeds.

RSS formats are specified using XML, a generic specification for the creation of data formats.

**S**

**Schema**: is its structure described in a formal language supported by the database management system (DBMS). In a relational database, the schema defines the tables, the fields, relationships, views, indexes, packages, procedures, functions, queues, triggers, types, sequences, materialized views, synonyms, database links, directories, Java, XML schemas, and other elements.

**SEM_MATCH**: is a Table Function to Query Semantic Data in oracle's 11g semantic technology.

**SEM_MODELS**: is a Table Function to retrieve models attribute that identifies the model or models to use. It has the following definition: TABLE OF VARCHAR2(25).

**SPARQL**: is an RDF query language; its name is a recursive acronym that stands for SPARQL Protocol and RDF Query Language. It was standardized by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is considered a key semantic web technology. On 15 January 2008, SPARQL became an official W3C Recommendation.

**SQL**: (Structured Query Language) is a database computer language designed for managing data in relational database management systems (RDBMS). Its scope includes data query and update, schema creation and modification, and data access control.

**T**

**Triple**: (RDF triple). The underlying structure of any expression in RDF is a collection of triples. Triples contains three components: the subject, which is an RDF URI reference or a blank node, the predicate, which is an RDF URI reference, the object, which is an RDF URI reference, a literal or a blank node.

**U**

**URI**: Uniform Resource Identifier is a formatted string that serves as an identifier for a resource, typically on the Internet. URIs are used in HTML to identify the anchors of hyperlinks. URIs in common practice include Uniform Resource Locators (URLs)[URL] and Relative URLs [RELURL].

**URL**: Uniform Resource Locator, an HTTP address used by the World Wide Web to specify a certain site. This is the unique identifier, or address, of a web page on the internet. URL can be pronounced "you-are-ell" or "earl." It is how web pages, ftp's, gophers, newsgroups and even some email boxes are located.

**V**

**Variable**: it's a name used to represent data that can be changed while the program or procedure is running.

**W**

**W3C**: Worldwide Web Consortium, which develops industry standards for XML and other areas.

**Web 2.0:** is commonly associated with web development and web design that facilitates interactive information sharing, interoperability, user-centered design and collaboration on the World Wide Web.

**Web 3.0**: The predicted third generation of the World Wide Web usually conjectured to include semantic tagging of content.

**Web feeds**: (or news) is a data format used for providing users with frequently updated content. Content distributors syndicate a web feed, thereby allowing users to subscribe to it. Making a collection of web feeds accessible in one spot is known as aggregation, which is performed by an Internet aggregator. A web feed is also sometimes referred to as a syndicated feed.

**Web tier**: displays product information and personalized content, and makes site content available to customers through page navigation. You can consider the servers in this tier to be in the run-time environment, because they respond to client requests for static (or stateless)

Web site content, and run business logic, such as pipeline components, activated by the Web pages.

## XYZ

**XML**: XML (Extensible Markup Language) is a set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification produced by the W3C and several other related specifications; all are fee-free open standards. XML's design goals emphasize simplicity, generality, and usability over the Internet. It is a textual data format, with strong support via Unicode for the languages of the world. Although XML's design focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services. There are a variety of programming interfaces which software developers may use to access XML data, and several schema systems designed to aid in the definition of XML-based languages. As of 2009, hundreds of XML-based languages have been developed, including RSS, Atom, SOAP, and XHTML. XML has become the default file format for most office-productivity tools, including Microsoft Office, OpenOffice.org, AbiWord, and Apple's iWork.

**XHTML**: A reformulation of HTML to conform to the XML specification.

**XQuery**: is a query and functional programming language that is designed to query collections of XML data.

# Bibliography

[1] Allemang D., Mashups and Semantic Mashups, SYS-CON Media, January 2007

[2] Bloesch A., and Halpin T., Conceptual Queries using ConQuer-II, ER, 1997

[3] Chong E., Das S., Eadon G., and Srinivasan J., An efficient SQL-based RDF querying Scheme. VLDB'05, Springer. 2005

[4] Czejdo B., Elmasri R., Rusinkiewicz M., and Embley D., An algebraic language for graphical query formulation using EER model. Computer Science conference. ACM 1987

[5] De Troyer O., Meersman R., and Verlinden P., RIDL on the CRIS Case: A Workbench for NIAM, Proc. of IFIP WG 8.1 Working, 1988

[6] Ganguli M., Making use of JSP, Wiley Publishing, Inc., 2002

[7] Goldman R., and Widom J., DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, Stanford University, 1997

[8] Google: Getting Started Guide – Google Mashup Editor – Google Code, <http://code.google.com/gme/docs/gettingstarted.html>, March 2008

[9] Google: Google Mashup Editor, <http://editor.googlemashups.com/editor>, March 2008

[10] Hall M., More Servlets and Java Server Pages, Prentice Hall PTR, 2002

[11] Halpin R., Comceptual Querues, vol. 26, no. 2 of Database Newsletter, 1997

[12] Harms, D., JSP, Servlets, and MySQL, Hungry Minds, Inc, 2001

[13] Hofstede A., Proper H., and Wide T., Computer Supported Query Formulation in an Evolving Context, Australasian DB Conference, 1995

[14] Horowitz E., Java Server Pages, http://www-scf.usc.edu/~csci571/Slides/jsp.ppt, 2005

[15] Hu L., Chang Y., and Lang C., QueryScope: Visualizing Queries for Repeatable Database Tuning, PVLDB '08, August 23-28, 2008, pp. 1488-1491

[16] Iskold A., Semantic Web: Difficulties with the Classic Approach. The Read Write Web online magazine. Sep. 19, 2007

[17] Jarrar M., Dikaiakos D. M., MashQL: A Query-by-Diagram Language -Towards Semantic Data Mashups, Proceedings of the 2nd International Workshop on Ontologies and Information Systems for Semantic Web (ONISW 2008), in conjunction with the ACM 17th Conference on Information and Knowledge Management, Napa Valley, California, October 26-30, 2008, ACM, pp. 89-96

[18] Jarrar M., Dikaiakos D. M., Querying the Data Web -The MashQL approach., IEEE Internet Computing, University of Cyprus, Nicosia, 2008

[19] Johnson S., QUERY-BY-EXAMPLE (QBE), 2004, pp. 177-192

[20] Kaushik R., Shenoy P., Bohannon P., and Gudes E., Exploiding local similarity for indexing of paths in graph structured data. ICDE 2002

[21] Li Y., Yang H., and Jagadish H.V., NaLIX: an Interactive Natural Language Interface for Querying XML, SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland, USA, 2005

[22] Liu H., Suvarna R., and Slec K., Resource Description Framework: Metadata and its Applications, Arizona State University, April 2002

[23] Morbidoni C., Tummarello G., Polleres A., and Le Phuoc D., Semantic Web Pipes, DERI Galway, November 2007

[24] Murray C., Semenatic Techonogies Developer's Guide 11g Release (11.1), 2008

[25] Mustafa J. and Dikaiakos M., (SemQuer) A Query-by-Diagram Language Topping SPARQL, Technical Report, University of Cyprus, Nicosia, April 2008

[26] O'Reilly T., (May 2008) http://radar.oreilly.com/archives/2007/02/pipes-and-filters-for-the-inte.html

[27] Snell P. (2006) Access 2000: Query By Form, 2006

[28] TopQuadrant: http://www.topquadrant.com/products/SPARQLMotion.html

[29] Tummarello G., Pollares A., and Morbidoni C., Who the FOAF knows Alice? A needed step toward Semantic Web Pipes, DERI Galway, National University of Ireland, Galway, 2007

[30] Wikipedia, Mashup (web application hybrid), Wikipedia the free encyclopedia, March 2008

[31] William M., "Likert Scaling". Research Methods Knowledge Base, 2nd Edition, <http://www.socialresearchmethods.net/kb/scallik.php>, April 30, 2009

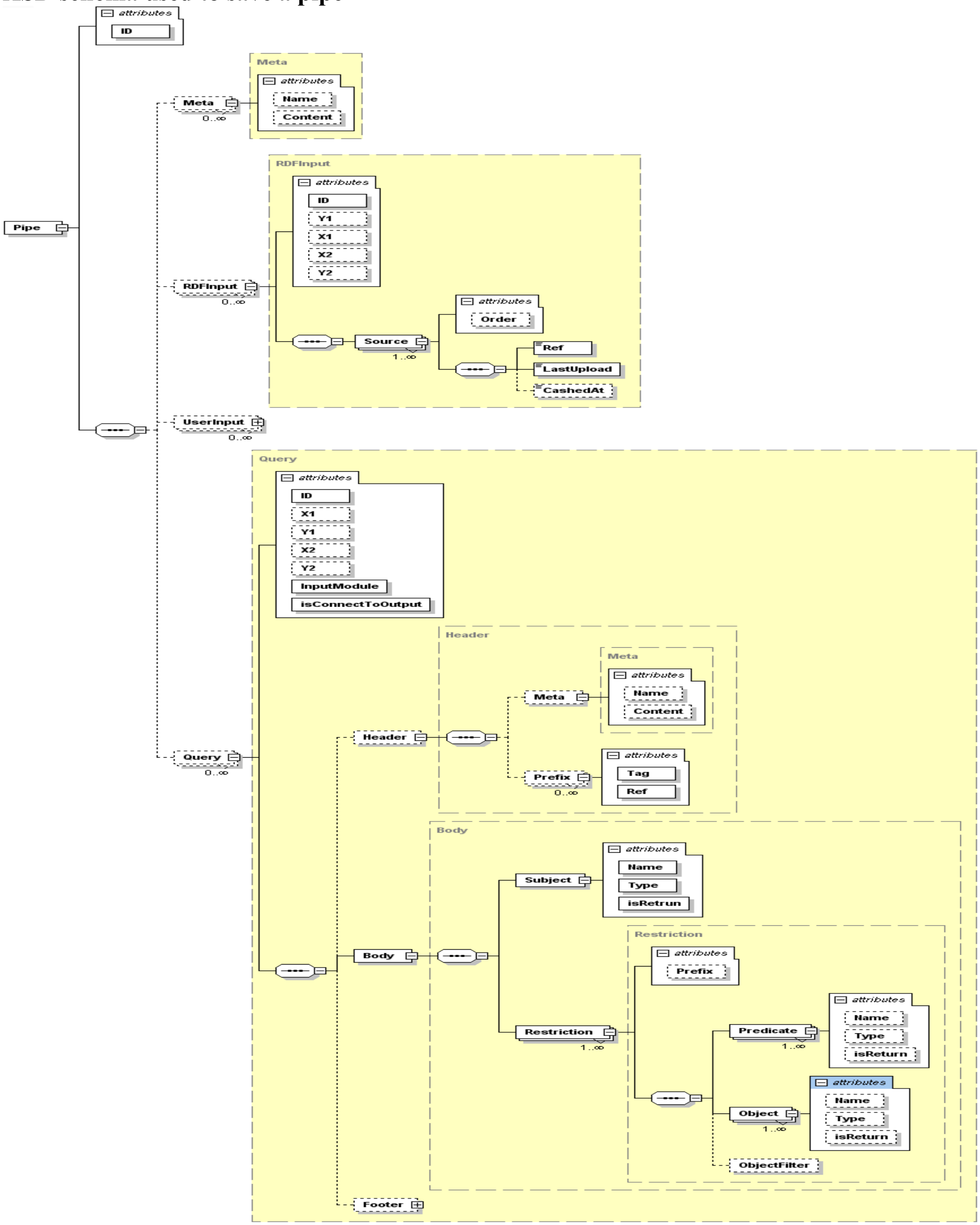
[32] Wikipedia: "XML", http://en.wikipedia.org/wiki/XML, March 2008

[33] Wikipedia: "Web Syndication", <http://en.wikipedia.org/wiki/Web_syndication>, March 2008

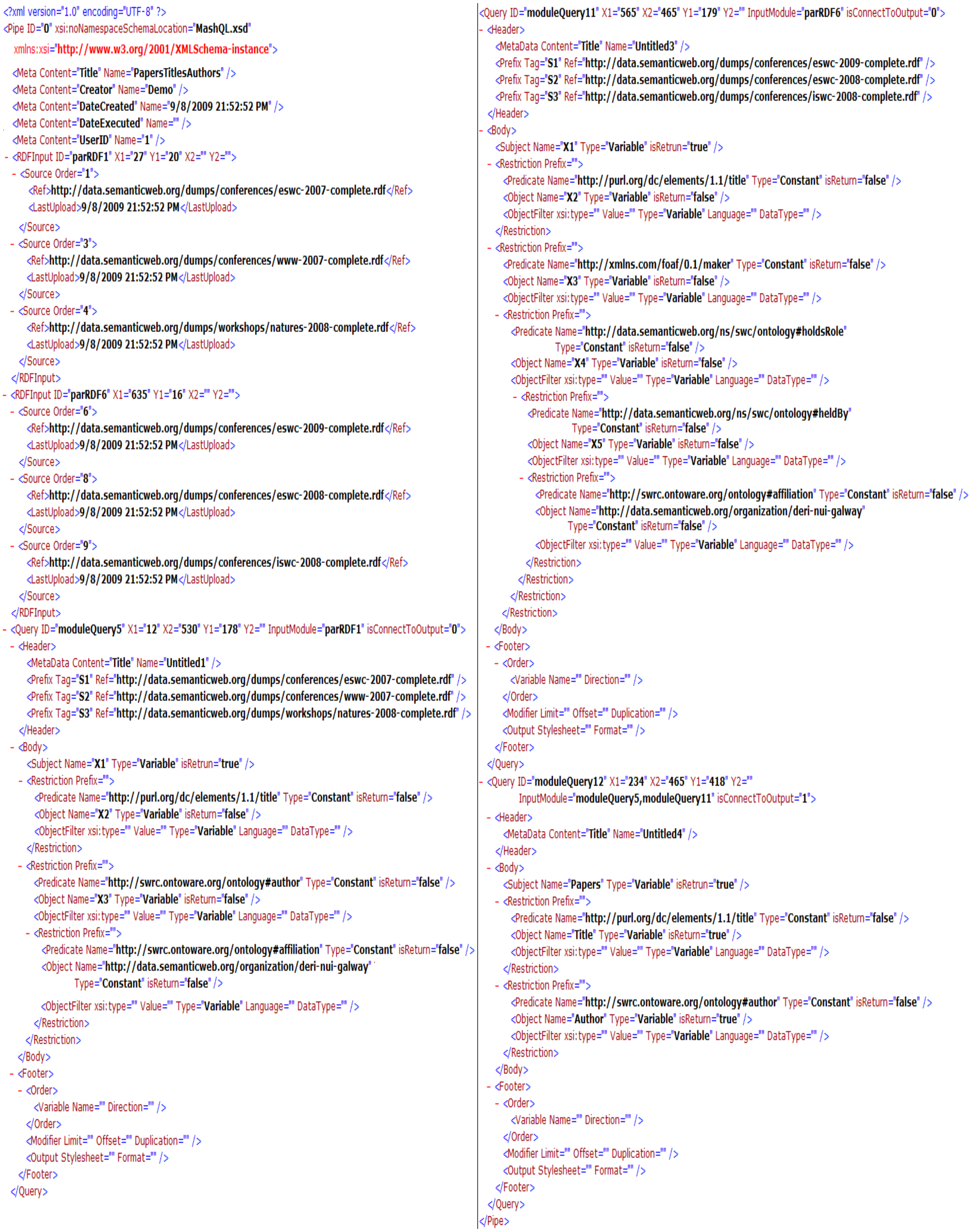
[34] Yahoo: Yahoo Pipes Editor, Available from: http://pipes.yahoo.com/pipes/ <http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29>

# Appendix

## XSD schema used to save a pipe



# XML file example



```
<?xml version="1.0" encoding="UTF-8" ?>
<Pipe ID="0" xsi:noNamespaceSchemaLocation="MashQL.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Meta Content="Title" Name="PapersTitlesAuthors" />
  <Meta Content="Creator" Name="Demo" />
  <Meta Content="DateCreated" Name="9/8/2009 21:52:52 PM" />
  <Meta Content="DateExecuted" Name="" />
  <Meta Content="UserID" Name="1" />
- <RDFInput ID="parRDF1" X1="27" Y1="20" X2="" Y2="">
  - <Source Order="1">
      <Ref>http://data.semanticweb.org/dumps/conferences/eswc-2007-complete.rdf</Ref>
      <LastUpload>9/8/2009 21:52:52 PM</LastUpload>
    </Source>
  - <Source Order="3">
      <Ref>http://data.semanticweb.org/dumps/conferences/www-2007-complete.rdf</Ref>
      <LastUpload>9/8/2009 21:52:52 PM</LastUpload>
    </Source>
  - <Source Order="4">
      <Ref>http://data.semanticweb.org/dumps/workshops/natures-2008-complete.rdf</Ref>
      <LastUpload>9/8/2009 21:52:52 PM</LastUpload>
    </Source>
  </RDFInput>
- <RDFInput ID="parRDF6" X1="635" Y1="16" X2="" Y2="">
  - <Source Order="6">
      <Ref>http://data.semanticweb.org/dumps/conferences/eswc-2009-complete.rdf</Ref>
      <LastUpload>9/8/2009 21:52:52 PM</LastUpload>
    </Source>
  - <Source Order="8">
      <Ref>http://data.semanticweb.org/dumps/conferences/eswc-2008-complete.rdf</Ref>
      <LastUpload>9/8/2009 21:52:52 PM</LastUpload>
    </Source>
  - <Source Order="9">
      <Ref>http://data.semanticweb.org/dumps/conferences/iswc-2008-complete.rdf</Ref>
      <LastUpload>9/8/2009 21:52:52 PM</LastUpload>
    </Source>
  </RDFInput>
- <Query ID="moduleQuery5" X1="12" X2="530" Y1="178" Y2="" InputModule="parRDF1" isConnectToOutput="0">
  - <Header>
      <MetaData Content="Title" Name="Untitled1" />
      <Prefix Tag="S1" Ref="http://data.semanticweb.org/dumps/conferences/eswc-2007-complete.rdf" />
      <Prefix Tag="S2" Ref="http://data.semanticweb.org/dumps/conferences/www-2007-complete.rdf" />
      <Prefix Tag="S3" Ref="http://data.semanticweb.org/dumps/workshops/natures-2008-complete.rdf" />
    </Header>
  - <Body>
      <Subject Name="X1" Type="Variable" isRetrun="true" />
    - <Restriction Prefix="">
        <Predicate Name="http://purl.org/dc/elements/1.1/title" Type="Constant" isReturn="false" />
        <Object Name="X2" Type="Variable" isReturn="false" />
        <ObjectFilter xsi:type="" Value="" Type="Variable" Language="" DataType="" />
      </Restriction>
    - <Restriction Prefix="">
        <Predicate Name="http://swrc.ontoware.org/ontology#author" Type="Constant" isReturn="false" />
        <Object Name="X3" Type="Variable" isReturn="false" />
        <ObjectFilter xsi:type="" Value="" Type="Variable" Language="" DataType="" />
      - <Restriction Prefix="">
          <Predicate Name="http://swrc.ontoware.org/ontology#affiliation" Type="Constant" isReturn="false" />
          <Object Name="http://data.semanticweb.org/organization/deri-nui-galway"
                Type="Constant" isReturn="false" />
          <ObjectFilter xsi:type="" Value="" Type="Variable" Language="" DataType="" />
        </Restriction>
      </Restriction>
    </Body>
  - <Footer>
    - <Order>
        <Variable Name="" Direction="" />
      </Order>
      <Modifier Limit="" Offset="" Duplication="" />
      <Output Stylesheet="" Format="" />
    </Footer>
  </Query>
<Query ID="moduleQuery11" X1="565" X2="465" Y1="179" Y2="" InputModule="parRDF6" isConnectToOutput="0">
- <Header>
    <MetaData Content="Title" Name="Untitled3" />
    <Prefix Tag="S1" Ref="http://data.semanticweb.org/dumps/conferences/eswc-2009-complete.rdf" />
    <Prefix Tag="S2" Ref="http://data.semanticweb.org/dumps/conferences/eswc-2008-complete.rdf" />
    <Prefix Tag="S3" Ref="http://data.semanticweb.org/dumps/conferences/iswc-2008-complete.rdf" />
  </Header>
- <Body>
    <Subject Name="X1" Type="Variable" isRetrun="true" />
  - <Restriction Prefix="">
      <Predicate Name="http://purl.org/dc/elements/1.1/title" Type="Constant" isReturn="false" />
      <Object Name="X2" Type="Variable" isReturn="false" />
      <ObjectFilter xsi:type="" Value="" Type="Variable" Language="" DataType="" />
    </Restriction>
  - <Restriction Prefix="">
      <Predicate Name="http://xmlns.com/foaf/0.1/maker" Type="Constant" isReturn="false" />
      <Object Name="X3" Type="Variable" isReturn="false" />
      <ObjectFilter xsi:type="" Value="" Type="Variable" Language="" DataType="" />
    - <Restriction Prefix="">
        <Predicate Name="http://data.semanticweb.org/ns/swc/ontology#holdsRole"
              Type="Constant" isReturn="false" />
        <Object Name="X4" Type="Variable" isReturn="false" />
        <ObjectFilter xsi:type="" Value="" Type="Variable" Language="" DataType="" />
      - <Restriction Prefix="">
          <Predicate Name="http://data.semanticweb.org/ns/swc/ontology#heldBy"
                Type="Constant" isReturn="false" />
          <Object Name="X5" Type="Variable" isReturn="false" />
          <ObjectFilter xsi:type="" Value="" Type="Variable" Language="" DataType="" />
        - <Restriction Prefix="">
            <Predicate Name="http://swrc.ontoware.org/ontology#affiliation" Type="Constant" isReturn="false" />
            <Object Name="http://data.semanticweb.org/organization/deri-nui-galway"
                  Type="Constant" isReturn="false" />
            <ObjectFilter xsi:type="" Value="" Type="Variable" Language="" DataType="" />
          </Restriction>
        </Restriction>
      </Restriction>
    </Restriction>
  </Body>
- <Footer>
  - <Order>
      <Variable Name="" Direction="" />
    </Order>
    <Modifier Limit="" Offset="" Duplication="" />
    <Output Stylesheet="" Format="" />
  </Footer>
</Query>
- <Query ID="moduleQuery12" X1="234" X2="465" Y1="418" Y2=""
      InputModule="moduleQuery5,moduleQuery11" isConnectToOutput="1">
  - <Header>
      <MetaData Content="Title" Name="Untitled4" />
    </Header>
  - <Body>
      <Subject Name="Papers" Type="Variable" isRetrun="true" />
    - <Restriction Prefix="">
        <Predicate Name="http://purl.org/dc/elements/1.1/title" Type="Constant" isReturn="false" />
        <Object Name="Title" Type="Variable" isReturn="true" />
        <ObjectFilter xsi:type="" Value="" Type="Variable" Language="" DataType="" />
      </Restriction>
    - <Restriction Prefix="">
        <Predicate Name="http://swrc.ontoware.org/ontology#author" Type="Constant" isReturn="false" />
        <Object Name="Author" Type="Variable" isReturn="true" />
        <ObjectFilter xsi:type="" Value="" Type="Variable" Language="" DataType="" />
      </Restriction>
    </Body>
  - <Footer>
    - <Order>
        <Variable Name="" Direction="" />
      </Order>
      <Modifier Limit="" Offset="" Duplication="" />
      <Output Stylesheet="" Format="" />
    </Footer>
  </Query>
</Pipe>
```

### MashQL language XSD schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2008 rel. 2 (http://www.altova.com) by mazuki (darksiderg) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:complexType name="ObjectFilter" abstract="true"/>
	<xs:complexType name="NotContains">
		<xs:complexContent>
			<xs:extension base="ObjectFilter">
				<xs:attribute name="Value"/>
				<xs:attribute name="Type"/>
				<xs:attribute name="DataType"/>
				<xs:attribute name="Language"/>
			</xs:extension>
		</xs:complexContent>
	</xs:complexType>
	<xs:complexType name="Contains">
		<xs:complexContent>
			<xs:extension base="ObjectFilter">
				<xs:attribute name="Value"/>
				<xs:attribute name="Type"/>
				<xs:attribute name="DataType"/>
				<xs:attribute name="Language"/>
			</xs:extension>
		</xs:complexContent>
	</xs:complexType>
	<xs:complexType name="NotEquals">
		<xs:complexContent>
			<xs:extension base="ObjectFilter">
				<xs:attribute name="Value"/>
				<xs:attribute name="Type"/>
				<xs:attribute name="DataType"/>
				<xs:attribute name="Language"/>
			</xs:extension>
		</xs:complexContent>
	</xs:complexType>
	<xs:complexType name="Equals">
		<xs:complexContent>
			<xs:extension base="ObjectFilter">
				<xs:attribute name="Value"/>
				<xs:attribute name="Type"/>
				<xs:attribute name="DataType"/>
				<xs:attribute name="Language"/>
			</xs:extension>
		</xs:complexContent>
	</xs:complexType>
	<xs:complexType name="NotLessThan">
		<xs:complexContent>
			<xs:extension base="ObjectFilter">
				<xs:attribute name="Value"/>
				<xs:attribute name="Type"/>
				<xs:attribute name="DataType"/>
			</xs:extension>
		</xs:complexContent>
```

```
</xs:complexType>
<xs:complexType name="LessThan">
    <xs:complexContent>
        <xs:extension base="ObjectFilter">
            <xs:attribute name="Value"/>
            <xs:attribute name="Type"/>
            <xs:attribute name="DataType"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="NotBetween">
    <xs:complexContent>
        <xs:extension base="ObjectFilter">
            <xs:attribute name="minValue"/>
            <xs:attribute name="maxValue"/>
            <xs:attribute name="Type"/>
            <xs:attribute name="DataType"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="Between">
    <xs:complexContent>
        <xs:extension base="ObjectFilter">
            <xs:attribute name="minValue"/>
            <xs:attribute name="maxValue"/>
            <xs:attribute name="Type"/>
            <xs:attribute name="DataType"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="NotMoreThan">
    <xs:complexContent>
        <xs:extension base="ObjectFilter">
            <xs:attribute name="Value"/>
            <xs:attribute name="Type"/>
            <xs:attribute name="DataType"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="MoreThan">
    <xs:complexContent>
        <xs:extension base="ObjectFilter">
            <xs:attribute name="Value"/>
            <xs:attribute name="Type"/>
            <xs:attribute name="DataType"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="NotOneOf">
    <xs:complexContent>
        <xs:extension base="ObjectFilter">
            <xs:choice>
                <xs:element name="DataItem" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:attribute name="Value"/>
                        <xs:attribute name="Type"/>
                        <xs:attribute name="DataType"/>
                        <xs:attribute name="Language"/>
                    </xs:complexType>
                </xs:element>
                <xs:element name="UserInput">
                    <xs:complexType>
                        <xs:attribute name="ModuleID"/>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="OneOf">
    <xs:complexContent>
        <xs:extension base="ObjectFilter">
            <xs:choice>
                <xs:element name="DataItem" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:attribute name="Value"/>
                        <xs:attribute name="Type"/>
                        <xs:attribute name="DataType"/>
                        <xs:attribute name="Language"/>
                    </xs:complexType>
                </xs:element>
                <xs:element name="UserInput">
                    <xs:complexType>
                        <xs:attribute name="ModuleID"/>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="SubQuery">
    <xs:sequence>
        <xs:element name="Restrictions" type="Restriction"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Body">
    <xs:sequence>
        <xs:element name="Subject">
            <xs:complexType>
                <xs:attribute name="Name" use="required"/>
                <xs:attribute name="Type" use="required"/>
                <xs:attribute name="isRetrun" type="xs:boolean"
                   use="required"/>
            </xs:complexType>
        </xs:element>
            <xs:element name="Restriction " type="Restriction"
              maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Header">
    <xs:sequence>
        <xs:element name="Meta" type="Meta" minOccurs="0"/>
        <xs:element name="Prefix" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="Tag" use="required"/>
                <xs:attribute name="Ref" use="required"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Footer">
    <xs:sequence>
        <xs:element name="Order">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Variable">
                        <xs:complexType>
                            <xs:attribute name="Name"/>
                            <xs:attribute name="Direction"/>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Modifier">
            <xs:complexType>
                <xs:attribute name="Duplication"/>
                <xs:attribute name="Limit"/>
                <xs:attribute name="Offset"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="Output">
            <xs:complexType>
                <xs:attribute name="Format"/>
                <xs:attribute name="Stylesheet"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Restriction">
    <xs:sequence>
        <xs:element name="Predicate" maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="Name"/>
                <xs:attribute name="Type"/>
                <xs:attribute name="isReturn" type="xs:boolean"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="Object" maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="Name"/>
                <xs:attribute name="Type"/>
                <xs:attribute name="isReturn" type="xs:boolean"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="ObjectFilter" type="ObjectFilter" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Prefix"/>
```

```
</xs:complexType>
<xs:element name="Pipe">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Meta" type="Meta" minOccurs="0"
                  maxOccurs="unbounded"/>
            <xs:element name="RDFInput" type="RDFInput" minOccurs="0"
                  maxOccurs="unbounded"/>
            <xs:element name="UserInput" type="UserInput" minOccurs="0"
                  maxOccurs="unbounded"/>
            <xs:element name="Query" type="Query" minOccurs="0"
                  maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="ID" type="xs:integer" use="required"/>
    </xs:complexType>
</xs:element>
<xs:complexType name="Meta">
    <xs:attribute name="Name"/>
    <xs:attribute name="Content"/>
</xs:complexType>
<xs:complexType name="RDFInput">
    <xs:sequence>
        <xs:element name="Source" maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Ref" type="xs:anyURI"/>
                    <xs:element name="LastUpload" type="xs:dateTime"/>
                    <xs:element name="CashedAt" minOccurs="0"/>
                          </xs:sequence>
                <xs:attribute name="Order" type="xs:integer"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="ID" type="xs:integer" use="required"/>
    <xs:attribute name="Y1" type="xs:integer"/>
    <xs:attribute name="X1" type="xs:integer"/>
    <xs:attribute name="X2" type="xs:integer"/>
    <xs:attribute name="Y2" type="xs:integer"/>
</xs:complexType>
<xs:complexType name="UserInput">
    <xs:sequence>
        <xs:element name="DataItem" maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="Value"/>
                <xs:attribute name="Type"/>
                <xs:attribute name="DataType"/>
                <xs:attribute name="Langauge"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="ID" type="xs:integer" use="required"/>
    <xs:attribute name="X1" type="xs:integer"/>
    <xs:attribute name="Y1" type="xs:integer"/>
    <xs:attribute name="X2" type="xs:integer"/>
    <xs:attribute name="Y2" type="xs:integer"/>
</xs:complexType>
```

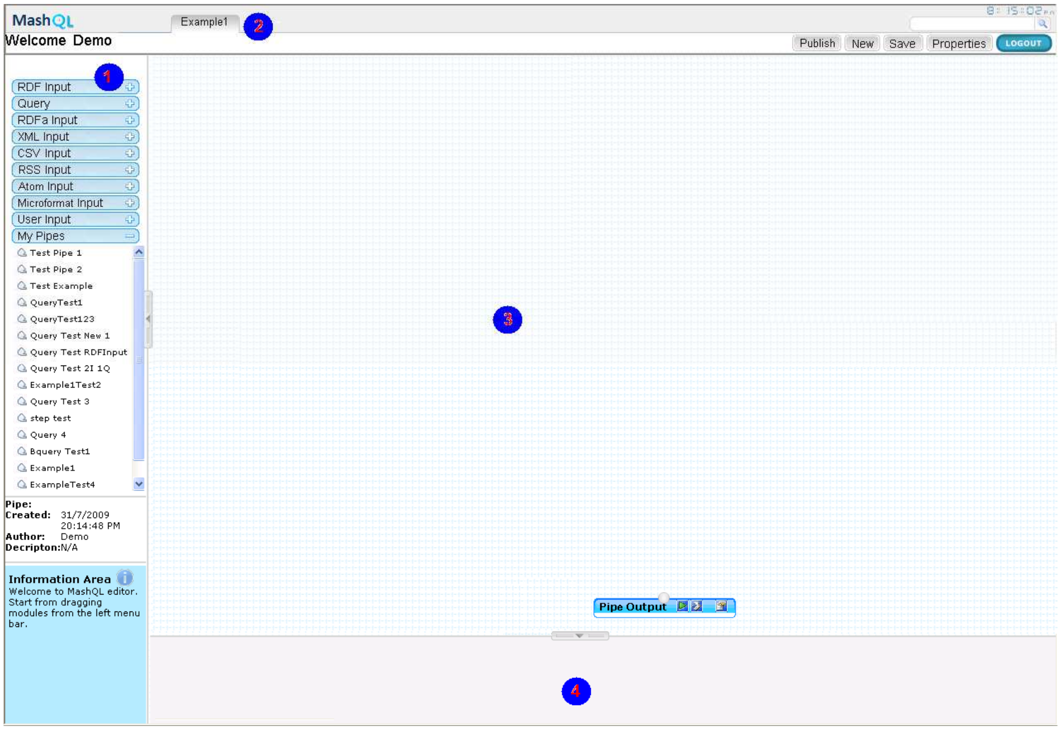
```
        <xs:complexType name="Query">
                <xs:sequence>
                        <xs:element name="Header" type="Header" minOccurs="0"/>
                        <xs:element name="Body" type="Body"/>
                        <xs:element name="Footer" type="Footer" minOccurs="0"/>
                </xs:sequence>
                <xs:attribute name="ID" type="xs:integer" use="required"/>
                <xs:attribute name="X1" type="xs:integer"/>
                <xs:attribute name="Y1" default="xs:integer"/>
                <xs:attribute name="X2" type="xs:integer"/>
                <xs:attribute name="Y2" type="xs:integer"/>
                <xs:attribute name="InputModule" type="xs:integer" use="required"/>
                <xs:attribute name="isConnectToOutput" type="xs:boolean" use="required"/>
        </xs:complexType>
</xs:schema>
```

**MashQL editor online help (Tutorial):**

# The editor

The **MashQL** editor looks likes the following image. It consists of 4 main modules that will help you formulate your pipes accordingly. The four modules are the following.

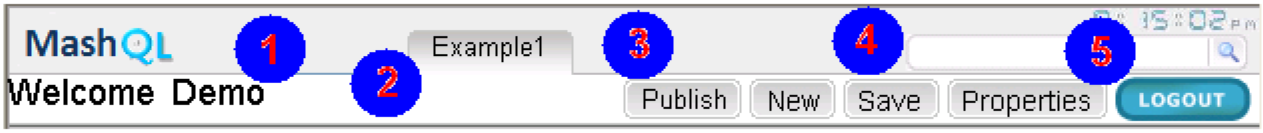
- Main Menu (2)
- Control Panel (1)
- Editor (3)
- Debugger (4)



**Pipe:**
**Created:** 31/7/2009 20:14:48 PM
**Author:** Demo
**Decripton:**N/A

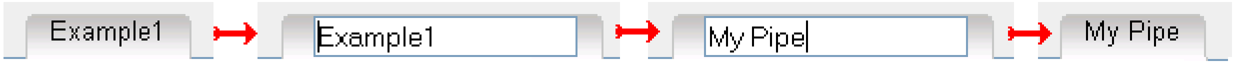
**Information Area**
Welcome to MashQL editor. Start from dragging modules from the left menu bar.

## Main Menu



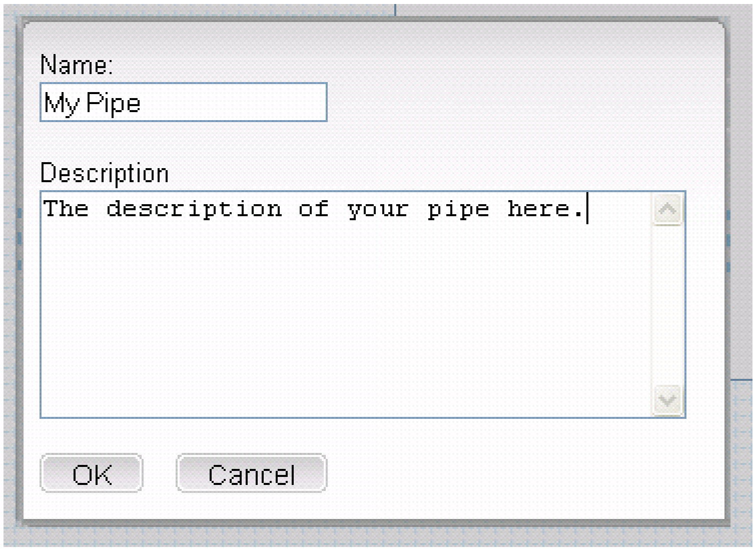
In the Main menu you can find the following modules:

- The current user name that you are logged in (1).
- The name of the Pipe (2). The default one is **Example 1**. You have to change the title to the desired one so that you can save the pipe in a later stage. You can click on that title and change it like the following example.



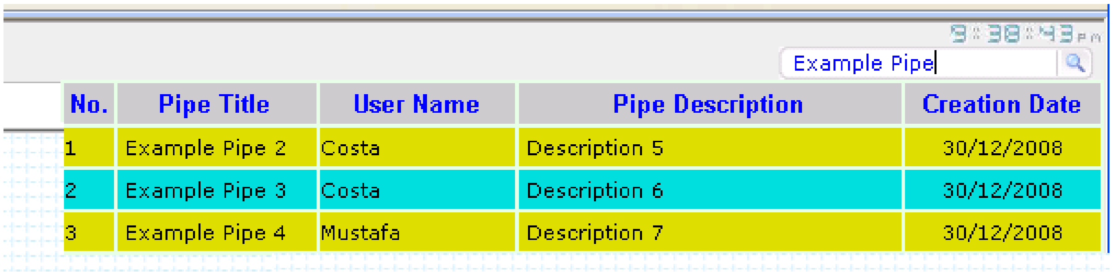
When you click in the title, you can write your own title and when you click anywhere else you can see the title as is shown in the last image in the above figure.

- Menu Buttons (3). You have the following options that you can choose from.
    - **Publish**. If you want to publish your pipe to the other users.
    - **New**. If you want to start a new pipe.
    - **Save**. If you want to save your current pipe.
    - **Properties** Options for entering title and description of your pipe, as you can see in the next figure.


Name:
My Pipe

Description
The description of your pipe here.

- Search Puplished Pipes (4). You can type a word that you lookking for to find an already published pipe as in the following example.



Example Pipe

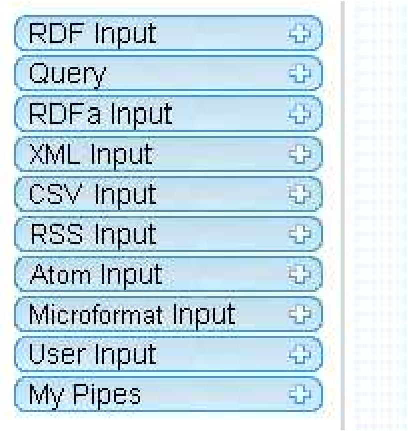
| No. | Pipe Title | User Name | Pipe Description | Creation Date |
|---|---|---|---|---|
| 1 | Example Pipe 2 | Costa | Description 5 | 30/12/2008 |
| 2 | Example Pipe 3 | Costa | Description 6 | 30/12/2008 |
| 3 | Example Pipe 4 | Mustafa | Description 7 | 30/12/2008 |

- LogOut (5). This button logs you out of the editor and brings you back to the login and register area.

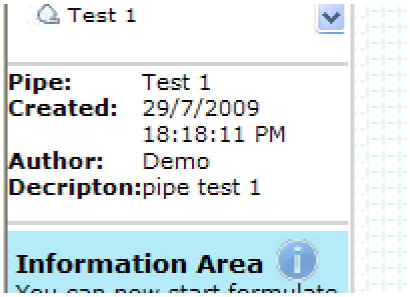
## Conrol Panel

The control panel consists of three other modules

- The module bar containing the (RDFInput, Query, RDFa, XML, CSV, RSS, Atom, Microformat, User Input, My Pipes modules) that could be dragged to the editor.



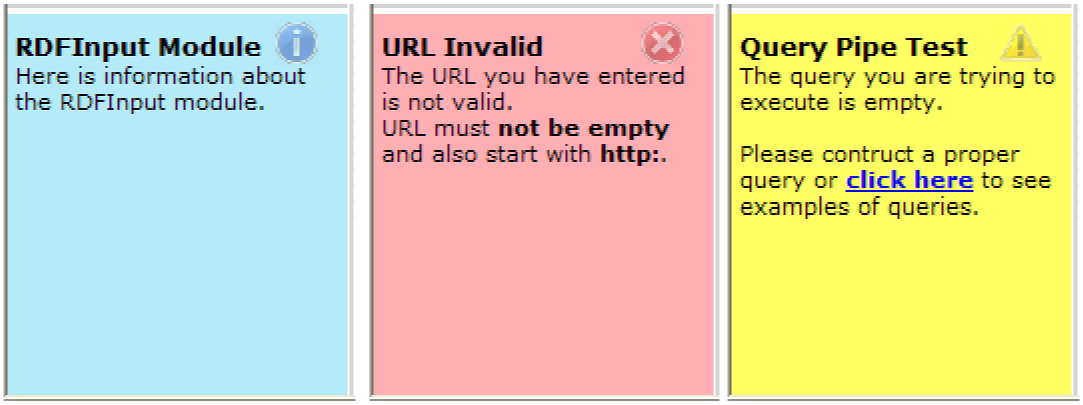
- The pipe information box (containing information of the current pipe).



**Pipe:** Test 1
**Created:** 29/7/2009 18:18:11 PM
**Author:** Demo
**Decripton:**pipe test 1

**Information Area**
You can now start formulate

- The information area (show current information, warning, error message to the user at real time).



**RDFInput Module**
Here is information about the RDFInput module.

**URL Invalid**
The URL you have entered is not valid.
URL must **not be empty** and also start with **http:**.

**Query Pipe Test**
The query you are trying to execute is empty.

Please contruct a proper query or **click here** to see examples of queries.

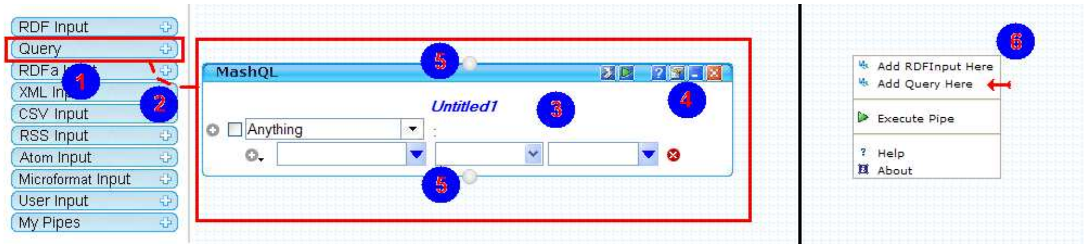
The information box is categorized in three categories depending on severity:

- **Infomation Message**. Blue background with ℹsign.
- **Waring Message**. Yellow background with ⚠sign.
- **Error Message**. Red background with ⊗sign.

## Editor

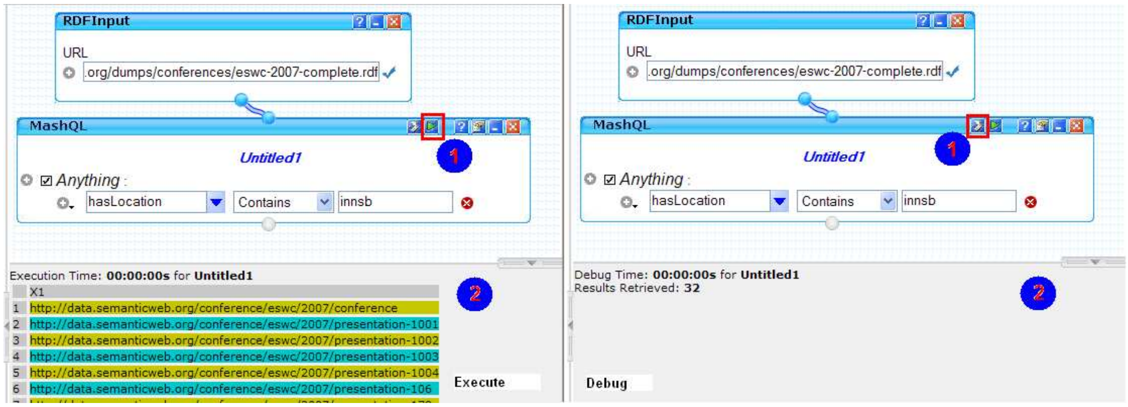
The editor is the actual place where you can generate the pipe, and connect the different modules between them to produce the desired results from the URLs that you have.
You can add modules to the editor in two ways

By dragging a module from the control panel to the editor. By leaving the left mouse button in the editor area the module is generated. By using right click in the editor area and selecting the module you want to generate from the popup menu.
You can see the example in the next figure of how you can handle modules' creation in the editor



## Debugger

The debugger is placed at the bottom of the editor and it's purpose is to show the **execution** results , or the **debug** result of a query or a pipe.
It could be resized up or down, so that if not needed to be small, or if is needed to take more space in the screen. En example of how the debugger is used are shown in the next figure/



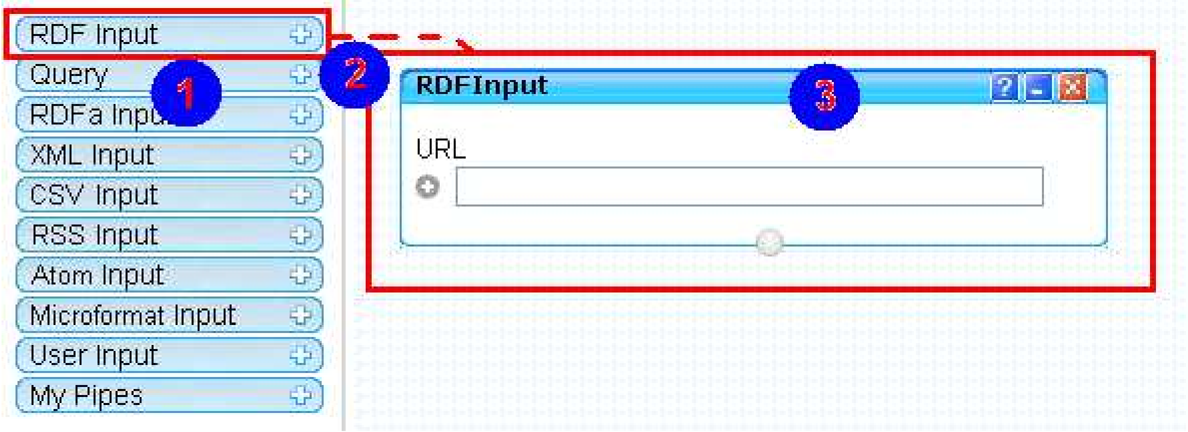
```
Execution Time: 00:00:00s for Untitled1
  X1
1 http://data.semanticweb.org/conference/eswc/2007/conference
2 http://data.semanticweb.org/conference/eswc/2007/presentation-1001
3 http://data.semanticweb.org/conference/eswc/2007/presentation-1002
4 http://data.semanticweb.org/conference/eswc/2007/presentation-1003
5 http://data.semanticweb.org/conference/eswc/2007/presentation-1004
6 http://data.semanticweb.org/conference/eswc/2007/presentation-106
```

```
Debug Time: 00:00:00s for Untitled1
Results Retrieved: 32
```

The point (2) shows the results when executing and when debugging a pipe.

# RDFInput Module

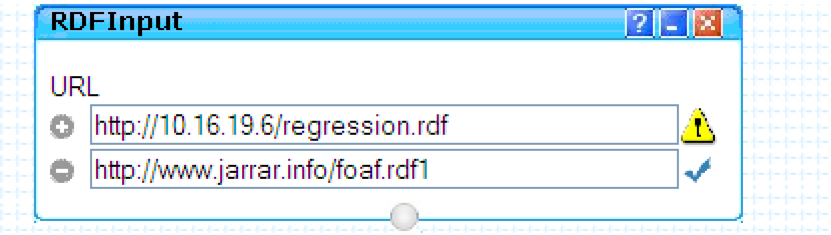
The **RDF Input** module is used, so you can add the URLs that you want to query from. You can find this module at the menu module as shown at point 1 in the next figure. You can then drag that module to the editor module as shown at point 2. When you leave the left mouse button then the RDF Input module is drawn in the editor's area and is ready to accept URLs inputs as shown at point 3 in the next figure.



You can write/paste the URL in the area next to the ⊕button. When you write the URL and leave from that area you can notice that next to it, there is a processing image ☼ idicating that the cotent of the URL is downlaoded. When the downloading finishes there are 2 posibilities.

- The content is downlaoded successfully, indicated with ✓image.
- There was an error on downloading and cannot be used for querying, indicated with ⚠ image.

You can see an example of this in the next figure.



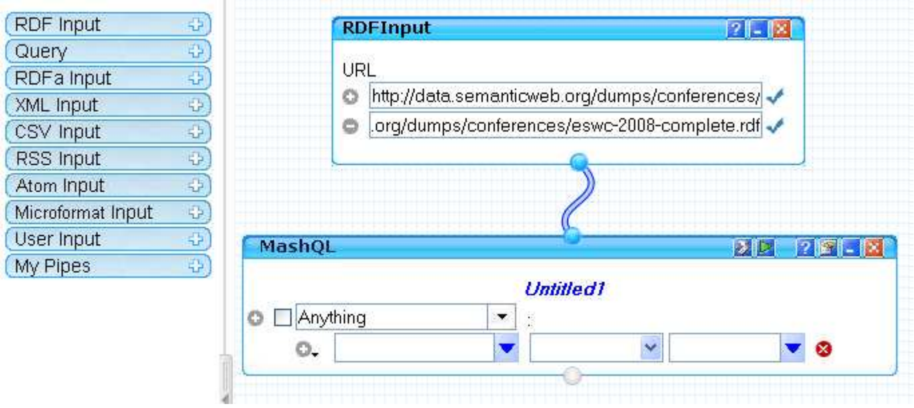
http://10.16.19.6/regression.rdf ⚠

http://www.jarrar.info/foaf.rdf1 ✓

There are also three helpfull buttons at the top right RDFInput module bar.

- The ?button shows module information the information area.
- The -button minimizes the module.
- The ☒button removes the module from the editor area.

The ○ sign is used as a terminal to connect the **RDFInput** module with a **Query** module as shown in the next example, so you can start querying the loaded URLs entered.

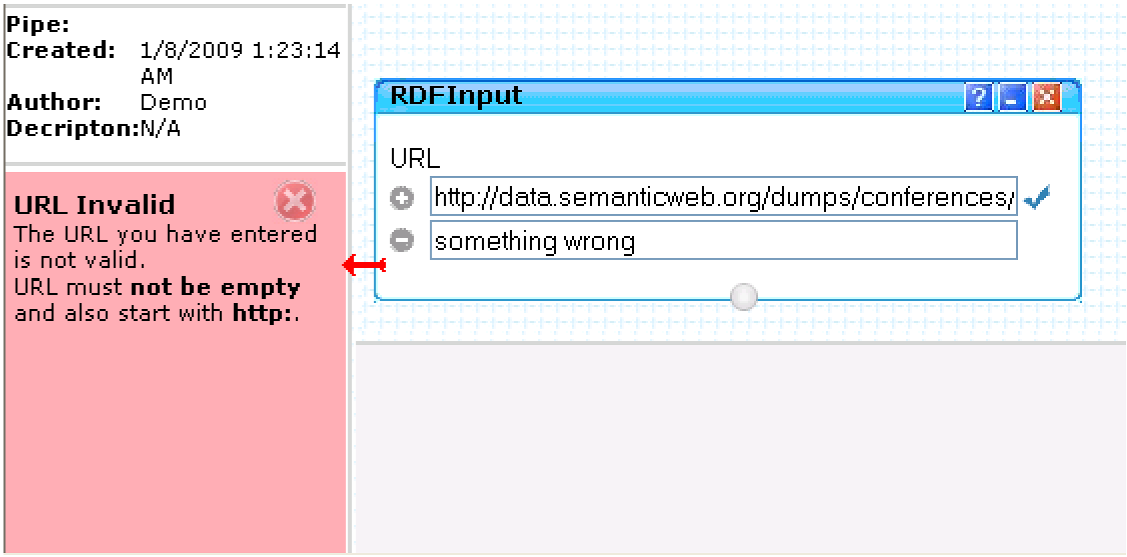


http://data.semanticweb.org/dumps/conferences/ ✓

.org/dumps/conferences/eswc-2008-complete.rdf ✓

□ Anything

Also notice the ⊕button which allows you to add more URL link to the module and the ⊖ button which removes a URL link from the module.
Also notice that if any **error, warning, or information** is activated in the module, then it will appear in the **information area box.**
For example if you don't insert a correct URL in the module then you can notice an error message in the information box as the following example



**Pipe:**
**Created:** 1/8/2009 1:23:14 AM
**Author:** Demo
**Decripton:**N/A

**URL Invalid**
The URL you have entered is not valid.
URL must **not be empty** and also start with **http:**.

RDFInput

URL

http://data.semanticweb.org/dumps/conferences/

something wrong

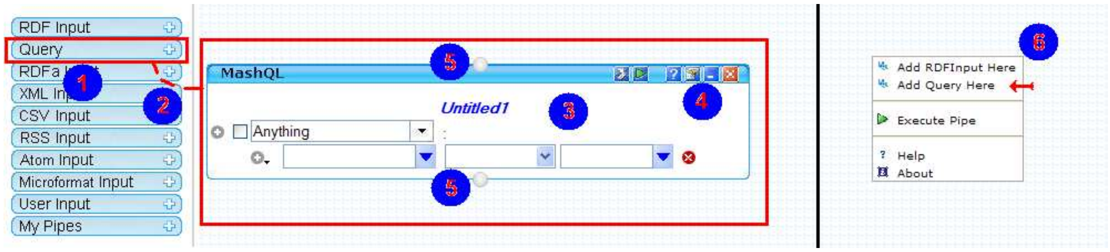
# Query Module

The **MashQL** module is the soul of the editor, since the queries are formulated with this module.

This module accept **RDFInput** and other **MashQL** modules as input and it can be connected with other **MashQL** or **Output** modules, as output.

The module could be found in the control panel and can be generated in two ways:
- Drag from control panel to the editor (1,2,3).
- Right click on the editor and click on "Add Query Here" (6).

You can see an example on the figure.



There are several helpful buttons for the query module as shown in the above figure with number (4):
- The **Debug** button, that debugs the current query ⏭.
- The **Execute** button, that executes the current query and shows the results in the dubugger ▶.

- The **Help** button, that show help info in the notification area.
- The **Query Options** button, that show options for the query to insert a title and view/change(exepriment) on the generated SPARQL of the query.
- The **Minimize/Maximize** button, that minimizes/maximizes the current query module.
- The **Close** button, that removes the current query module from the editor.

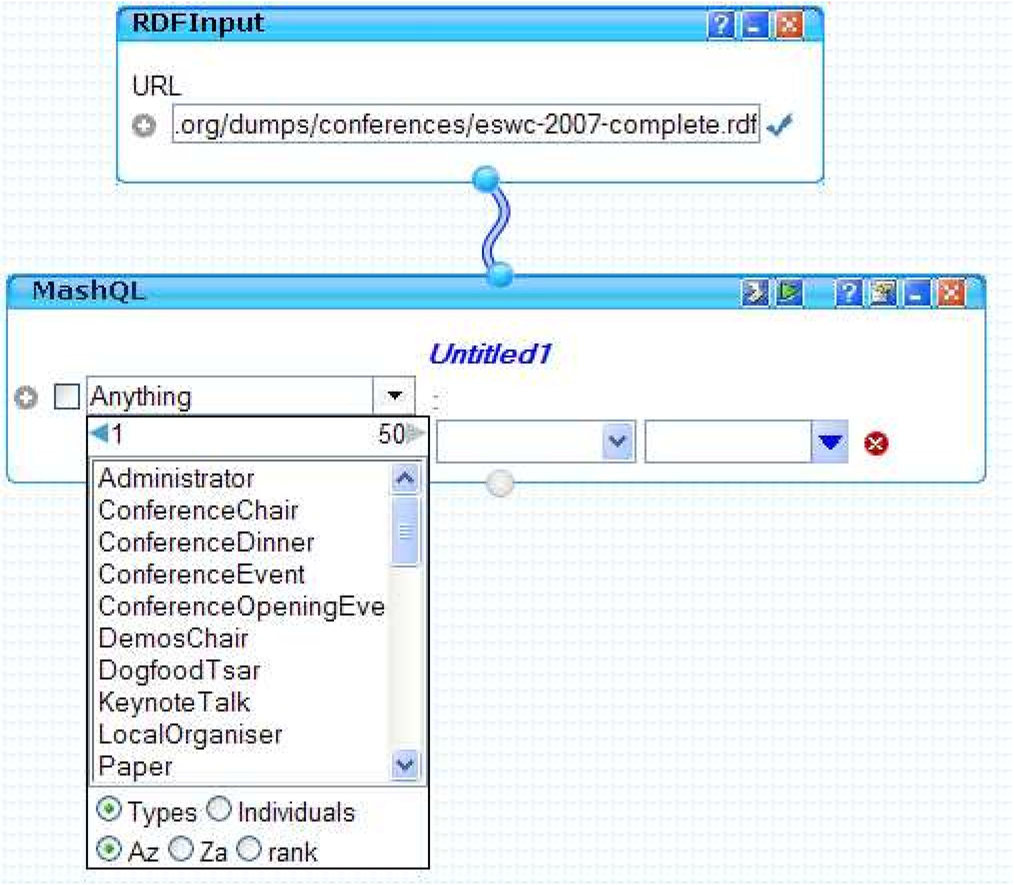
Also the query module has two terminals that can be connected (5):

- The one that is placed at the top of the module represents the inputs of the query modules. The inputs can be:
  - **RDFInput**.
  - other **MashQL** modules.
- The one that is placed at the bottom of the module represents the outputs of the query modules and inputs to other modules. Could be connected to:
  - other **MashQL** modules.
  - **Pipe Output** module.

## Prerequisites to formulate a Query

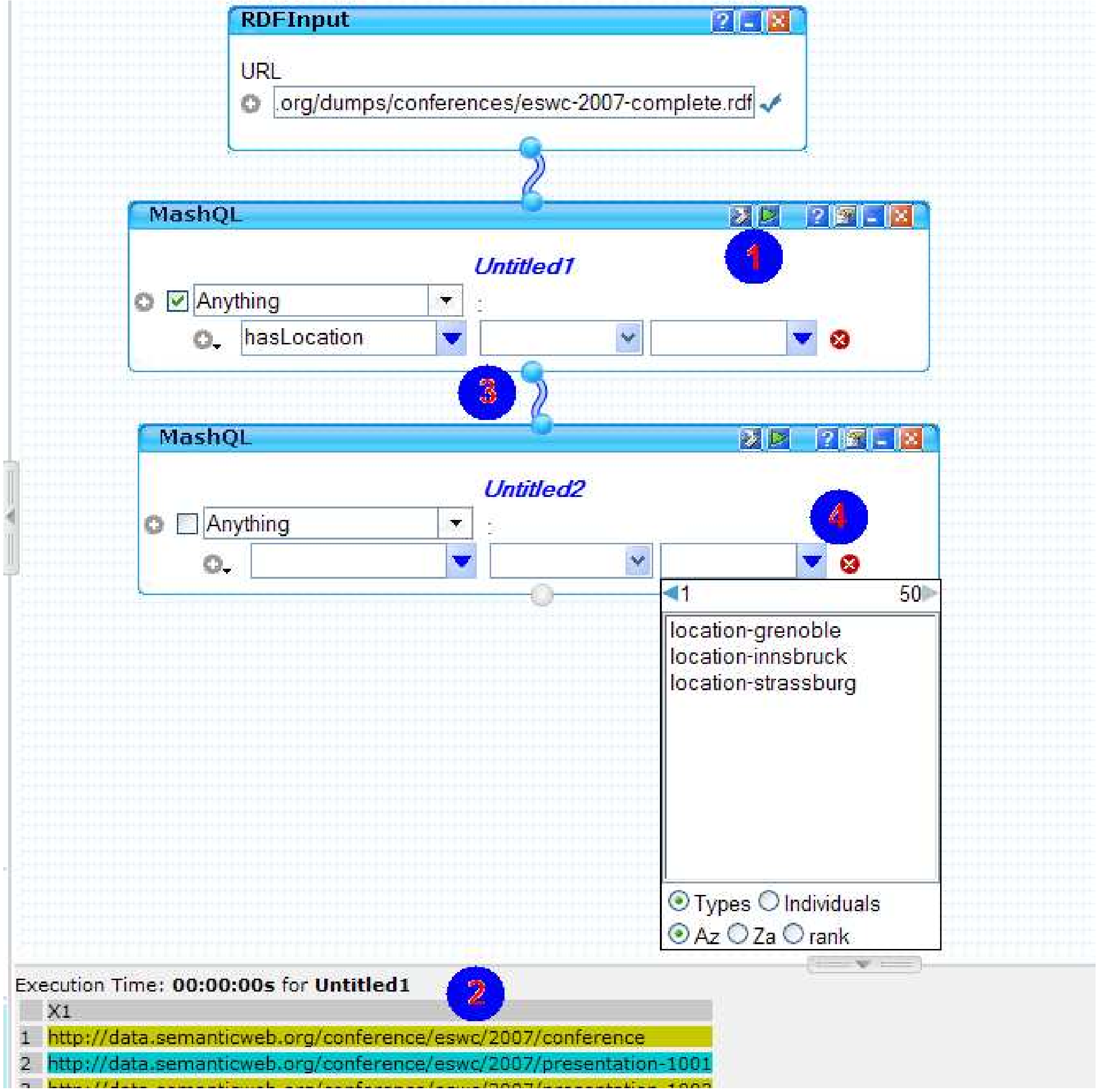
To formulate a query there are some things you have to do:

- If the input of the query is an **RDFInput**, then you have to add an RDFInput module in the editor, some valid URLs into it and also must be downloaded successfully. Then you have to connect the RDFInput module to the MashQL module and you can start formulating the query. An example is shown in the next figure.



URL: .org/dumps/conferences/eswc-2007-complete.rdf

If the input of the query is another **MashQL** module, then you must **execute** or **debug** the input MashQL query and then if is executed/debuged succeddfully then connect it to the new MashQL module. The next image shows an example of it.



URL: .org/dumps/conferences/eswc-2007-complete.rdf

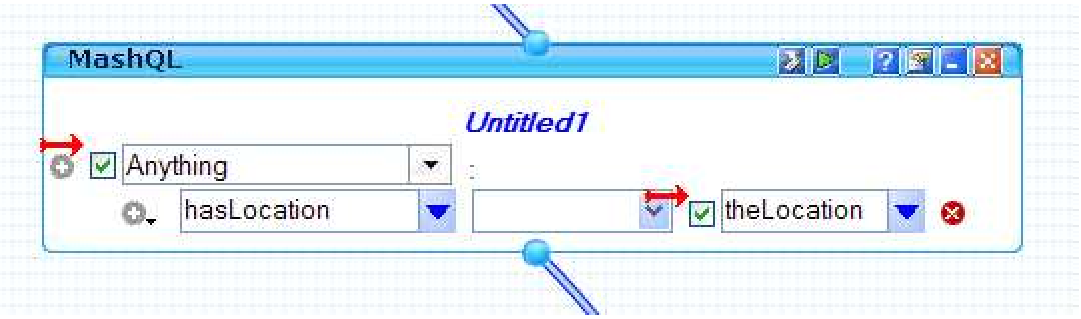
location-grenoble
location-innsbruck
location-strassburg

```
Execution Time: 00:00:00s for Untitled1
  X1
1 http://data.semanticweb.org/conference/eswc/2007/conference
2 http://data.semanticweb.org/conference/eswc/2007/presentation-1001
```

As the above figure shows, in step (1) you execute or debug the existing query. You can seen the results in the debbuger as shown in point (2). Then you connect the executed query to the new query (3). Then you start formulating your new query (4). The new query play the role of the **filtering**. In other words the puprose of it is to filter the results of the previous query.

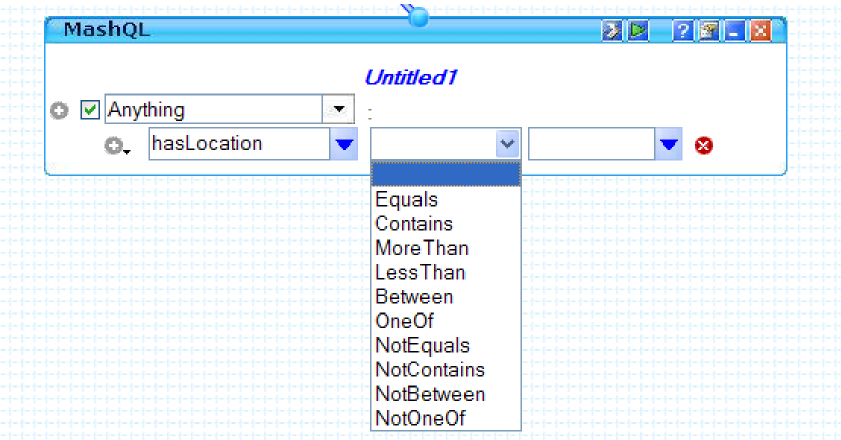
## Restictions to formulate a Query

To execute a query successfully, you have to select **at least one return entity**. In other words you have to check one of the check boxes that are placed in front of the three text areas. Those check boxes represent the return values of your query. An example is the following.



This query is translated to: *Select Anything and theLocation from the {soutce} where "Anything" has a property named hasLocation and this has a value "theLocation".* Note that if you select a value from the list, then you cannot select it to be a return value and you cannot see a check box in front of it

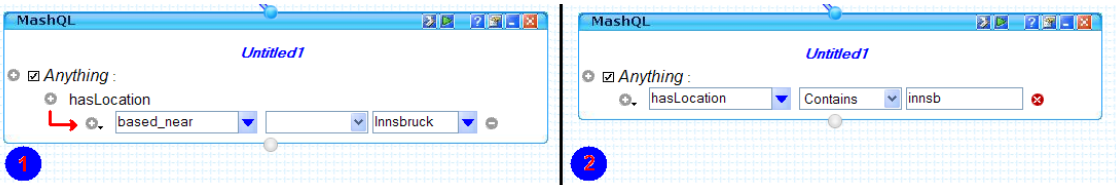
You can restrict you queries more by using the **functions** that are available from you, as we show in the next figure.



The restictions contains the following functions:

- Equals: A predicate equals to something.
- Container: A predicate contains a word like something.
- MoreThan: A predicate is bigger than a value.
- LessThan: A predicate is smaller than a value.
- Between: A predicate is between two values.
- OneOf: A predicate exist in a number of variables.
- The negation of all the above: Acts exactly the opposite.

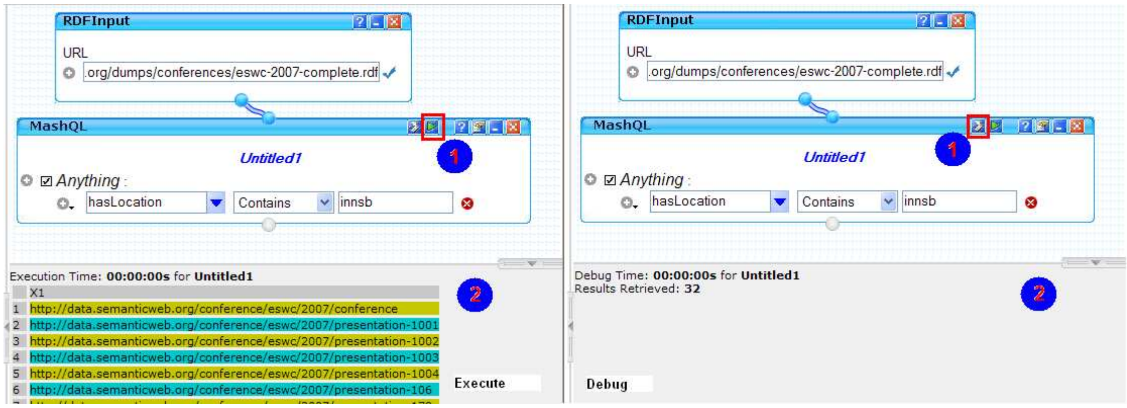
Note that if you add a function to a restriction then you **cannot** expand this restriction more. In other words you cannot press the ⊕button and create a branch of the restriction. The next example shows the different when you have a restriction and when you don't.



In the first part we expand the first restiction to a second one. However in the second, we cannot exand it because we added a fuction to it.

## Execute/Debug

To execute the query you just have to press the execute button at the top right of the MashQL module when you finish the query formulation. The debug is the same as the execute, but with the difference that it shows you statistics rather that results. It shows the time taken to execute the query and the rows retrieved from the database.



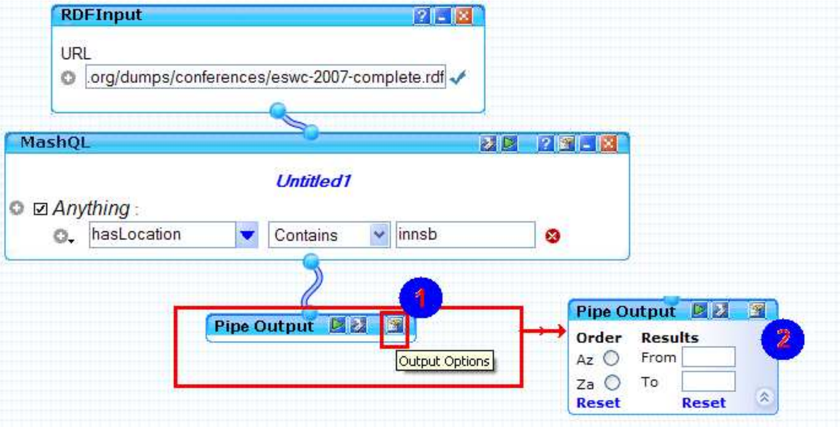
Execution Time: **00:00:00s** for **Untitled1**

| | X1 |
|---|---|
| 1 | http://data.semanticweb.org/conference/eswc/2007/conference |
| 2 | http://data.semanticweb.org/conference/eswc/2007/presentation-1001 |
| 3 | http://data.semanticweb.org/conference/eswc/2007/presentation-1002 |
| 4 | http://data.semanticweb.org/conference/eswc/2007/presentation-1003 |
| 5 | http://data.semanticweb.org/conference/eswc/2007/presentation-1004 |
| 6 | http://data.semanticweb.org/conference/eswc/2007/presentation-106 |

Debug Time: **00:00:00s** for **Untitled1**
Results Retrieved: **32**

The results are shown in the debugger as you can see in the above figure.

You can also execute and debug the whole pipe though the **Pipe Output** module. By connecting the last in hierarchy of the MashQL modules to the output module, you have the functionallity to limit the results or sort them, as you can see in the next figure.



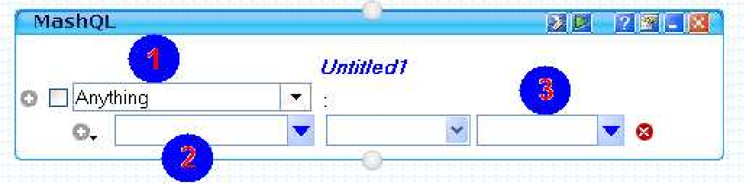
If you click on the output options (1) the module is expanded, giving you optons to sort and limit the results (2).
If you fill those options and you press the ⊠or the ▶button on the top right of the Pipe Output module, then it acts like when executing a query module, but stand for the whole pipe.

# The triples

The way RDF works is by using the **triples** pattern. The triples pattern is consisting of **(subject) (predicate/property) (object)**.
That is, a subject has a property, and the property has a value. For example:

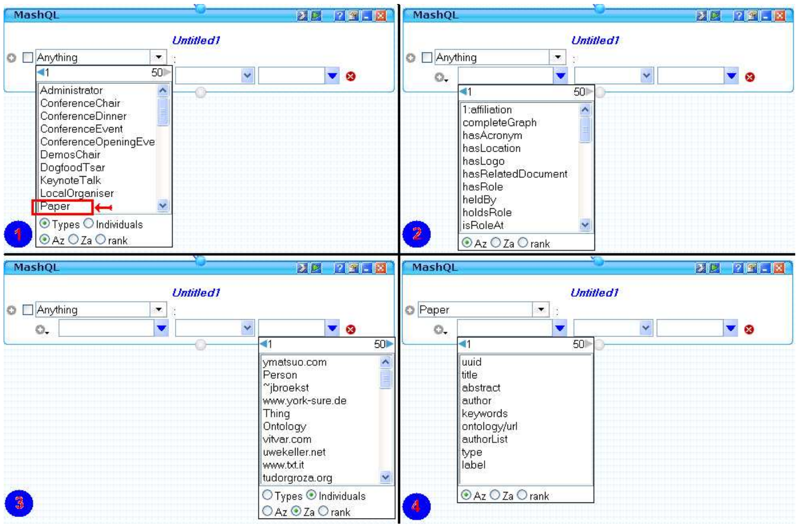
*(names) (gender) (male)*
If this was in a query the meaning is that: Give me all names that have a property gender, and the gender is male.

To represent this way that **RDF** works, the Query Module (MashQL) is used.



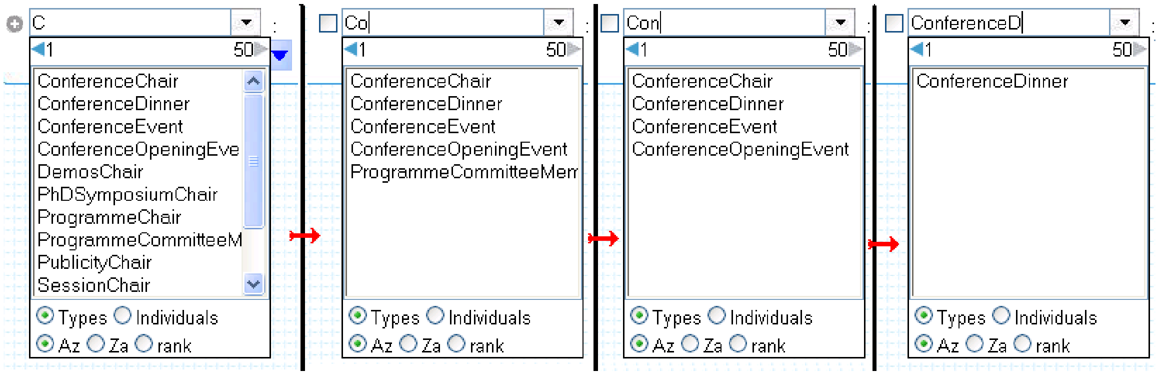
- (1) The subject.
- (2) The predicate.
- (3) The object.

Every of these entities has a check box □ which commands the query to return this entity in the query results.
The ▾ button show the subject list according to the sources predifined. The ▾ button when pressed shows the properties and objects according to the selected subject if any. You can see an example in the next figure.



In (1) the list shows the subjects of a particular RDFInput sourccs.
In (2) there are all the predicates according to RDFInput source, since no subject is selected to limit the results in the list
In (3) all the objects are shown. In (4) all the objects are shown. The difference is that the results are limited down to satisfy the pattern (Paper} {Predicate1} {Object1}, since the subject "Paper" already selected.

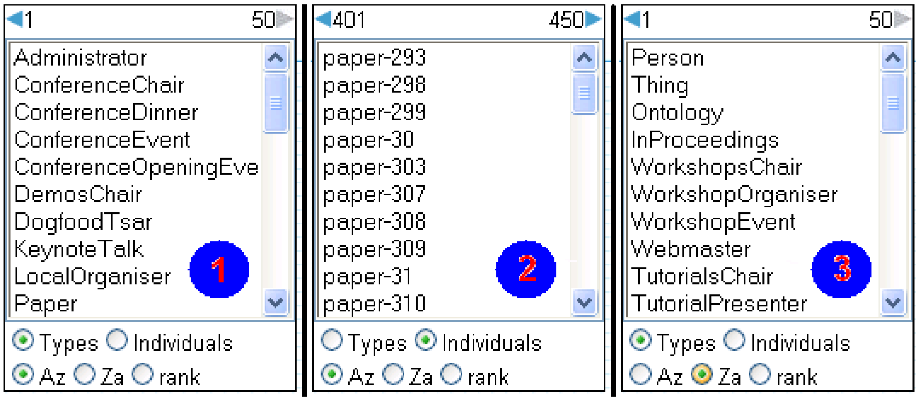
In addition for each of the entities you can limit down the results you want by typing in the text area of each one the word you want like the next example.



● Types ○ Individuals
● Az ○ Za ○ rank

## Types/Ordering

In the lists for each of the entity of a triple, you can choose the type of results to fetch and also the ordering of them.



1: ● Types ○ Individuals; ● Az ○ Za ○ rank

2: ○ Types ● Individuals; ● Az ○ Za ○ rank

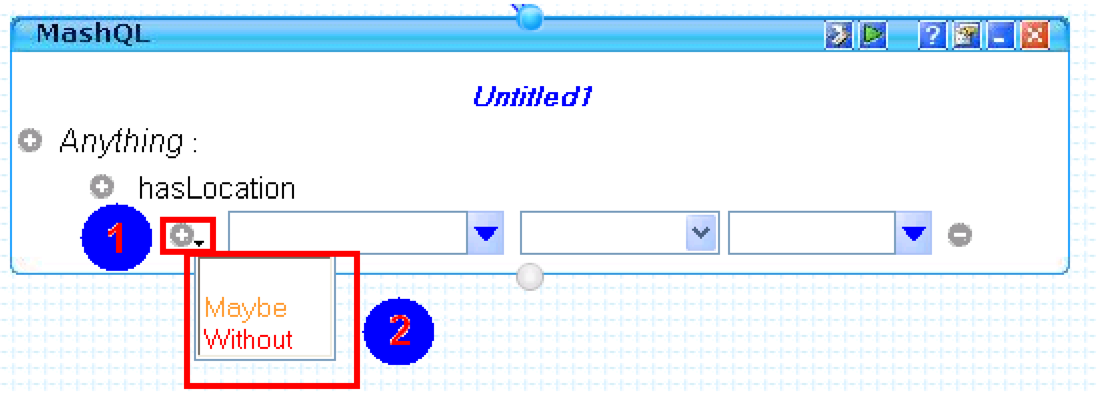
3: ● Types ○ Individuals; ○ Az ● Za ○ rank

- Types Conditions:
  - (1) Bring all the values that has a property of type.
  - (2) Bring all the values including and those that have property of type.
- Sorting:
  - (1) Sort the values in an ascending order (Az).
  - (3) Sort the values in an descendinf order (Za).

## Maybe/Without Restrictions

In each restriction in a wuery module you can notice a small arrow exactly next to the ⊕ button.

By clicking on that arrow a list appears, and you can select the restiction type of the restiction. You can see an example on the next figure.

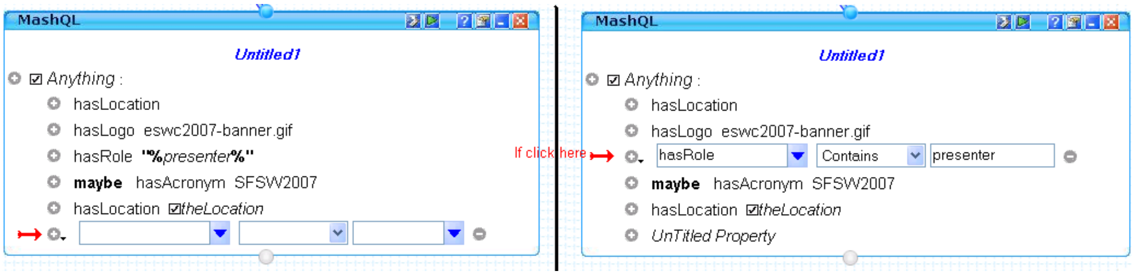


There are three posibilities:

- Empty: No addition restriction. Indicated with black.
- Maybe: The restriction may be exist in the resultset. Indicated with orange.
- Without: The restriction wiil not exist in the resultset. Indicated with red.

## Verbalization

The restriction that you can change is the one that you click on it when is verbalized.



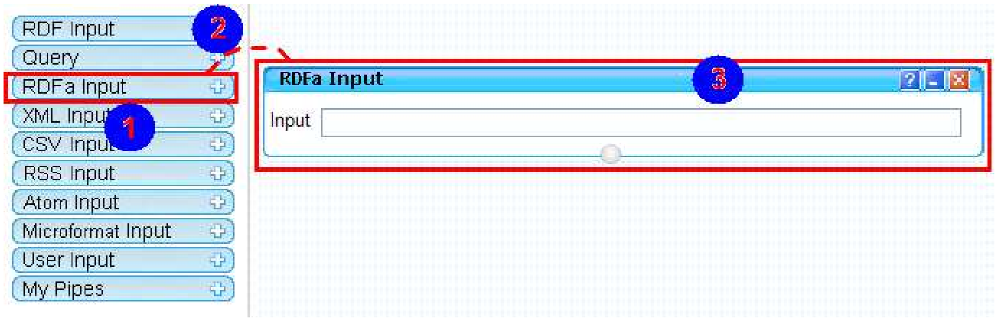
As you can see in the above figure, when you click on a particular restriction you can update it or delete it. The others are hidden and only their description is shown.

# RDFa Module

RDFa (or Resource Description Framework - in - attributes) is a set of extensions to XHTML that is now a W3C Recommendation. RDFa uses attributes from XHTML's meta and link elements, and generalises them so that they are usable on all elements. This allows annotating XHTML markup with semantics. A simple mapping is defined so that RDF triples may be extracted.The W3C RDF in XHTML Taskforce is also working on an implementation for non-XML versions of HTML. The primary issue for the non-XML implementation is how to handle the lack of XML namespaces.

The **RDFa Module** is used to add URLs with those kind of inputs.

The way that you can add this module is to drag it from the menu area to the editor area as you see in the next figure. It works the same way as the RDFInput module explained in the previous section.



There are also three helpfull buttons at the top right RDFa Input module bar.

- The ?button shows module information the information area.
- The ▬button minimizes the module.
- The ☒button removes the module from the editor area.

This input module is connected to a query module though its terminal so you can query the resources just like the RDFInput module.
However this module is still under construction so you cannot query any of these sources yet.

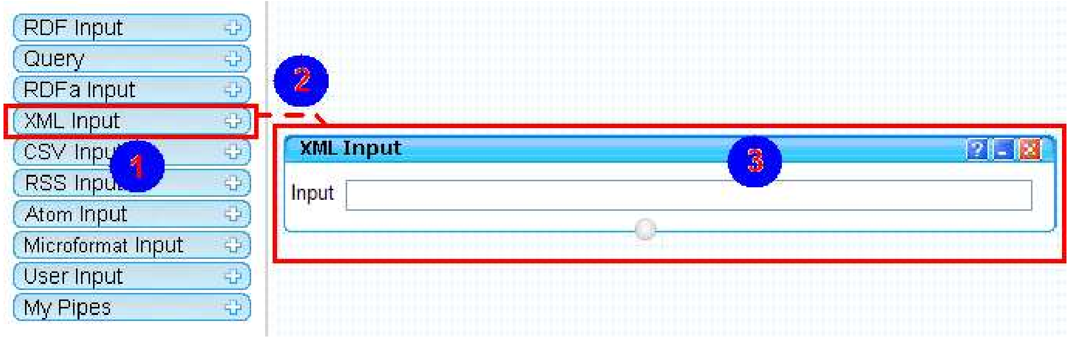
# XML Input Module

XML (Extensible Markup Language) is a set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification produced by the W3C and several other related specifications; all are fee-free open standards.

XML's design goals emphasize simplicity, generality, and usability over the Internet. It is a textual data format, with strong support via Unicode for the languages of the world. Although XML's design focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.

The **XML Input module** accepts those kind of inputs.

The way you can add this module so you can query them, is to do exactly as we explain in the previous two modules. You just drag the XML module to the editor area and add the XML URLs in the area next to Input label.
You can see the example in the next figure.



There are also three helpfull buttons at the top right XML Input module bar.

- The ?button shows module information the information area.

- The −button minimizes the module.
- The ☒button removes the module from the editor area.

This input module is connected to a query module though its terminal so you can query the resources just like the RDFInput module.
However this module is still under construction so you cannot query any of these sources yet.

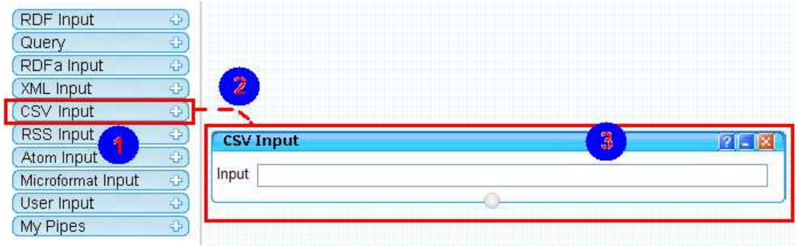
# CSV Input Module

A Comma separated values (CSV) file is used for the digital storage of data structured in a table of lists form, where each associated item (member) in a group is in association with others also separated by the commas of its set. Each line in the CSV file corresponds to a row in the table.

Within a line, fields are separated by commas, each field belonging to one table column. Since it is a common and simple file format, CSV files are often used for moving tabular data between two different computer programs, for example between a database program and a spreadsheet program.

The are two known formats of it. The .csv and .txt formats.

The **CSV Input module** accepts those kind of inputs. You can add it to the editor by dragging it from the control panel area as you can see in the next example.



There are also three helpfull buttons at the top right CSV Input module bar.
- The ?button shows module information the information area.
- The −button minimizes the module.
- The ☒button removes the module from the editor area.

This input module is connected to a query module though its terminal so you can query the resources just like the RDFInput module.
However this module is still under construction so you cannot query any of these sources yet.

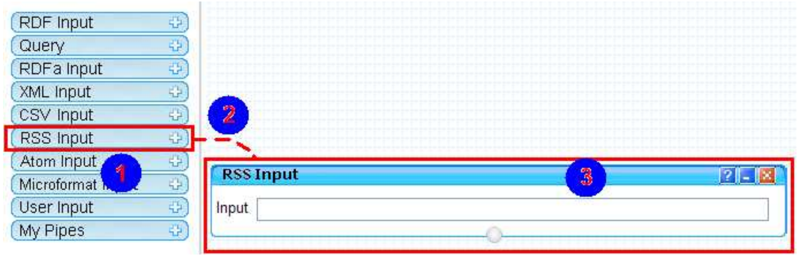
# RSS Input Module

RSS (most commonly translated as "Really Simple Syndication" but sometimes "Rich Site Summary") is a family of web feed formats used to publish frequently updated works—such as blog entries, news headlines, audio, and video—in a standardized format.

An RSS document (which is called a "feed", "web feed", or "channel") includes full or summarized text, plus metadata such as publishing dates and authorship. Web feeds benefit publishers by letting them syndicate content automatically.

The two known formats of the RSS are the .rss and .xml formats.

The **RSS Input module** could be found in the control panel, and is resposible to accept those kind of inputs. You can see an example in the next figure.



There are also three helpfull buttons at the top right RSS Input module bar.

- The ?button shows module information the information area.
- The -button minimizes the module.
- The ☒button removes the module from the editor area.

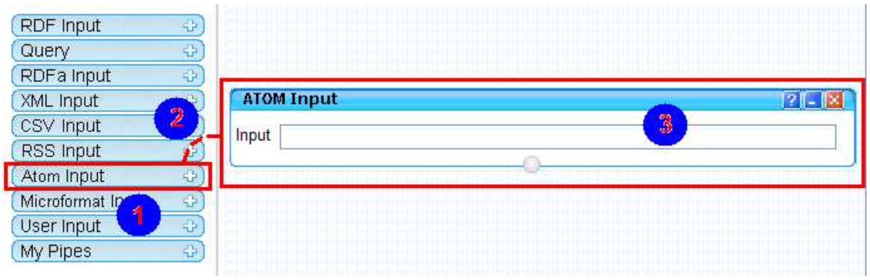
This input module is connected to a query module though its terminal so you can query the resources just like the RDFInput module.
However this module is still under construction so you cannot query any of these sources yet.

# ATOM Input Module

The name Atom applies to a pair of related standards. The Atom Syndication Format is an XML language used for web feeds, while the Atom Publishing Protocol (AtomPub or APP) is a simple HTTP-based protocol for creating and updating web resources.

Web feeds allow software programs to check for updates published on a web site. To provide a web feed, a site owner may use specialized software (such as a content management system) that publishes a list (or "feed") of recent articles or content in a standardized, machine-readable format. The feed can then be downloaded by web sites that syndicate content from the feed, or by feed reader programs that allow Internet users to subscribe to feeds and view their content.

The ATOM Input module which is responsible for these kind of data could be found in the control panel as you can see in the next example. You can drag it from there into the editor area to input the URLs related to this format.



There are also three helpfull buttons at the top right ATOM Input module bar.

- The ?button shows module information the information area.
- The ▬button minimizes the module.
- The ☒button removes the module from the editor area.

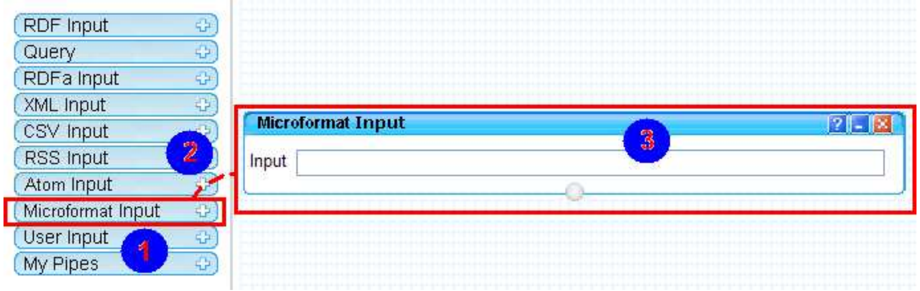
This input module is connected to a query module though its terminal so you can query the resources just like the RDFInput module.
However this module is still under construction so you cannot query any of these sources yet.

# Microformat Input Module

A microformat is a web-based approach to semantic markup that seeks to re-use existing XHTML and HTML tags to convey metadata[2] and other attributes.

This approach allows information intended for end-users (such as contact information, geographic coordinates, calendar events, and the like) to also be automatically processed by software.

The module is located in the control panel and could be dragged to the editor area to accept URLs with these kind of format as you can see in the next example



There are also three helpfull buttons at the top right MICROFORMAT Input module bar.

- The ?button shows module information the information area.
- The ▬button minimizes the module.
- The ☒button removes the module from the editor area.

This input module is connected to a query module though its terminal so you can query the resources just like the RDFInput module.
However this module is still under construction so you cannot query any of these sources yet.

# User Input Module

The User Input module, is used to accept any datatype with its values that the user want to add as a restriction in the query module. Supported datatype are the following:

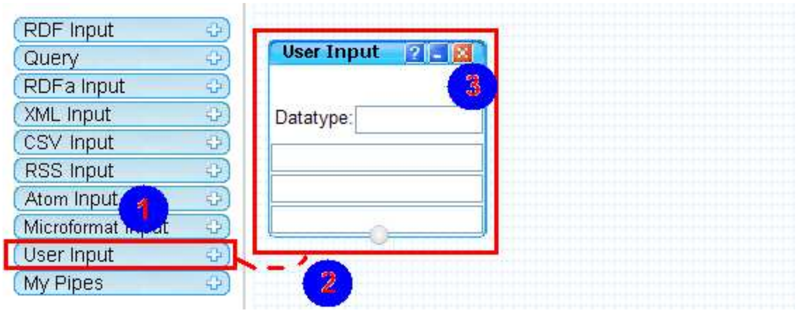
- integers,
- floating-point numbers (decimals), and
- alphanumeric strings.

It is used in the case that you choose the functions of **OneOf** and **NotOneOf** in a query restriction. In case you choose one of these functions you can connect the restriction to the User Input module and specify the datatype and the actual values that will be included in the restriction.

For example we may have the next case:

> *Select a Country which is one of something*
> The word something is replaced with User Input module like this.
> *Datatype:String. Value:Cyprus,Greece*
> So the restriction now is formulated as:
> *Select Country which is **OneOf** {'Cyprus','Greece'}*

The User Input module responsible to contribute to the queries' restrictions could be found in the control panel and can be dragged in the editor area as shown in the next figure.



There are also three helpfull buttons at the top right USER Input module bar.

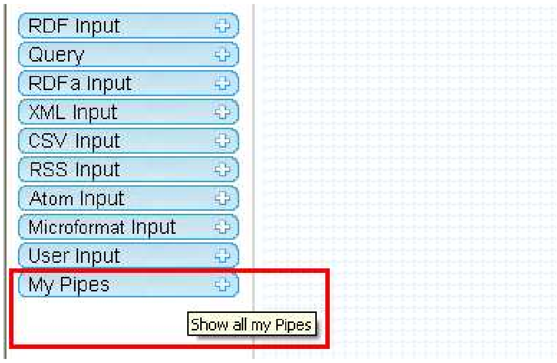
- The ?button shows module information the information area.
- The -button minimizes the module.
- The ☒button removes the module from the editor area.

However this module is still under construction so you cannot query any of these sources yet.

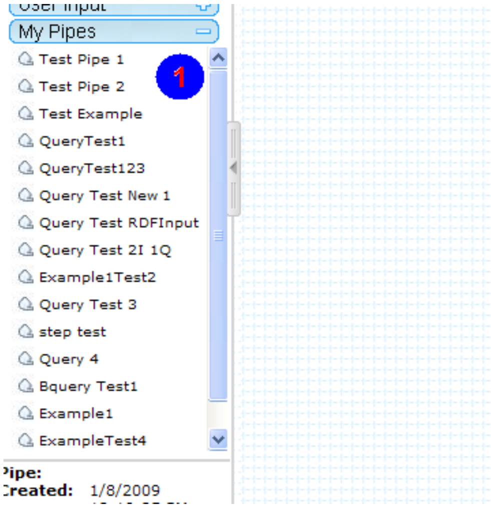
# How to view/load My Pipes

The last module in the control panel is **My Pipes** module. Here you can find all the pipes that you have create.

When you move your mouse over that module you can see a tip saying that when you click that module your pipes will appear, as shown in the next figure.



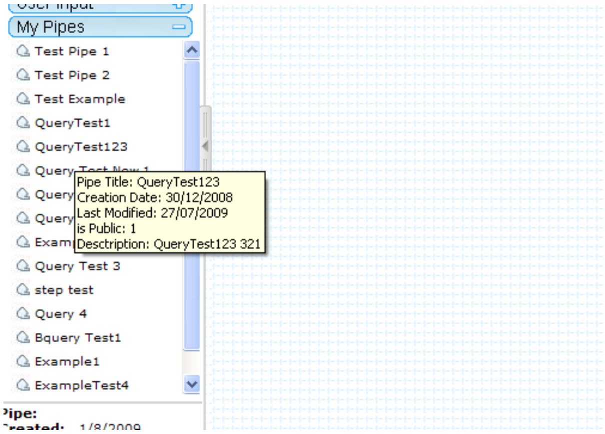
On clicking, below the module you see all your pipes, labeled with their names as they have been saved the last time, as you can see in the next figure.



If you move your mouse over a pipe you can see some information about it like:

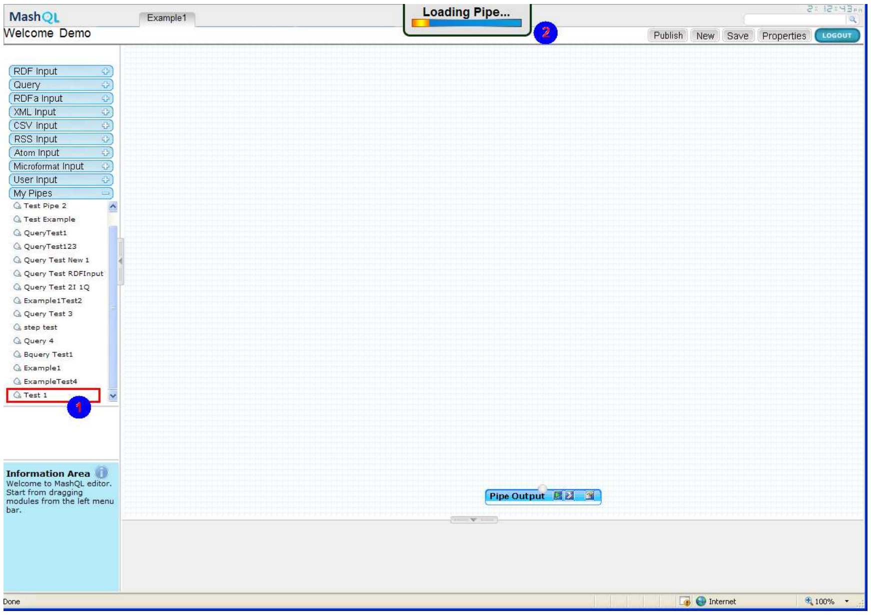
- Title
- Creation Date
- Last Modified
- if the pipe is Public to the other
- and the Description of the Pipe

You can see an example in the next figure of **Test1** pipe.



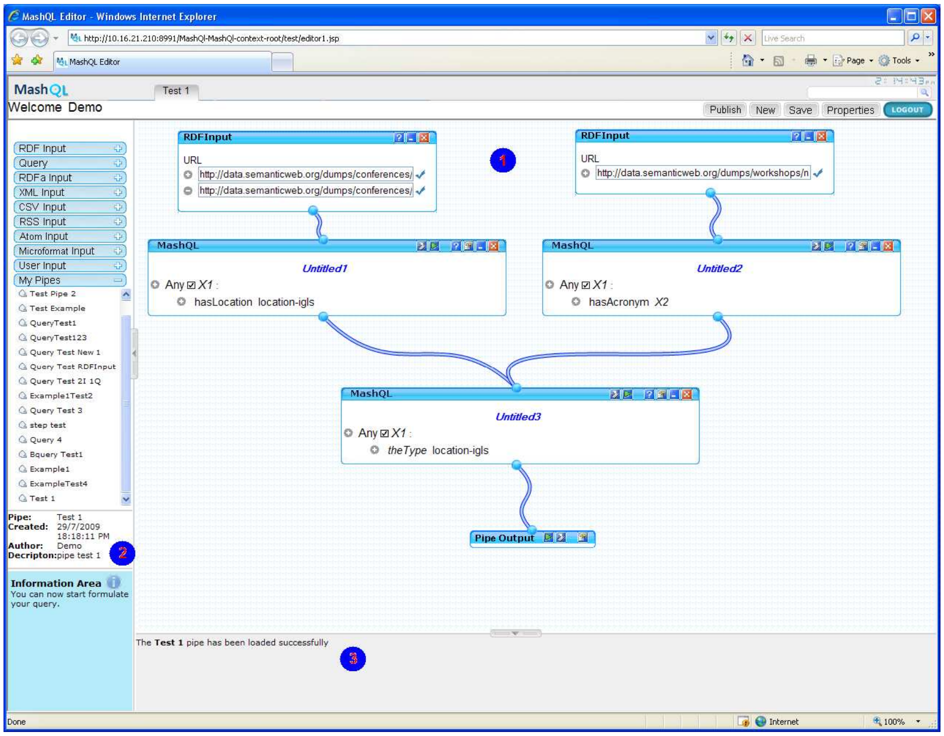
Pipe Title: QueryTest123
Creation Date: 30/12/2008
Last Modified: 27/07/2009
is Public: 1
Desctription: QueryTest123 321

If you want to load a particular pipe you just have to click to it.
Like in the next figure, if you first click on the pipe, you will wait for a while till the pipe is loaded. An indication (2) is shown to you while you wait.



Information Area
Welcome to MashQL editor. Start from dragging modules from the left menu bar.

When the loading finishes successfully you can see your pipe as was last saved. For example in the next figure, you can see the pipe in the editor area (1).
Also you can see the details of the pipe in the pipe information area (2).
A notification is also shown in the debugger, saying that the pipe was loaded successfully (3).



http://10.16.21.210:8991/MashQl-MashQl-context-root/test/editor1.jsp

RDFInput URL: http://data.semanticweb.org/dumps/conferences/
http://data.semanticweb.org/dumps/conferences/

RDFInput URL: http://data.semanticweb.org/dumps/workshops/n

Untitled1: Any ☑ X1 : hasLocation location-igls

Untitled2: Any ☑ X1 : hasAcronym X2

Untitled3: Any ☑ X1 : theType location-igls

Pipe: Test 1
Created: 29/7/2009 18:18:11 PM
Author: Demo
Decripton: pipe test 1

Information Area
You can now start formulate your query.

The **Test 1** pipe has been loaded successfully