Birkbeck

University of London

Department of Computer Science and Information Systems

Wonts - Web Ontology Searcher

Project Proposal: MSc Advanced Computing Technologies

Salvatore Rapisarda

April 2015

This proposal is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service.

Table of Contents

Proposal

In today's fast moving world of accessible information at the touch of the finger, it is more and more prevalent that the speed of data accessibility and information retrieval is increasing. A global demand for data on the World Wide Web[1] (WWW) from users of mobile & tablet devices, laptops and desktop computers have created the need for intelligent forms of retrieving these data using a maturing technology known as Semantic Web[2].

Semantic Web contains a framework aimed at standardising common data, formats and exchange protocols on the Web. This standardisation is governed by World Wide Web Consortium (W3C)[3], who aims to promote the sharing of data and exchange protocols on the Web by helping to build a technology architecture to support a web of data. Semantic Web[2] refers to W3C's vision of the Web of linked data.

This project is based on the Semantic Web technologies for linking data and index web information. One of the most important parts of the Semantic Web is the vocabulary or ontology[3]. The ontology is used to define the concepts and relationships.

Below is an example of a triple, which uses the basic representation of a concept: Subject – Predicate - Object

The concept of Subject – Predicate – Object can be simplified in a sentence, for example "John is a man". "John" is the subject, "is a" is the predicate (property or rule) and "man" is the object. In this case "man" is the class that "John" belongs to. Therefore "John" is an individual and each individual instances of this class is a "man". The ontology and set of individual instances of classes is a knowledge base or TBox[4].

By using the triple we can easily represent a simple ontology of a "Family" that is shown in the drawing below (Figure 1).

[1] http://en.wikipedia.org/wiki/World_Wide_Web
[2] http://en.wikipedia.org/wiki/Semantic_Web
[3] http://www.w3.org/standards/semanticweb/ontology
[4] http://en.wikipedia.org/wiki/Tbox

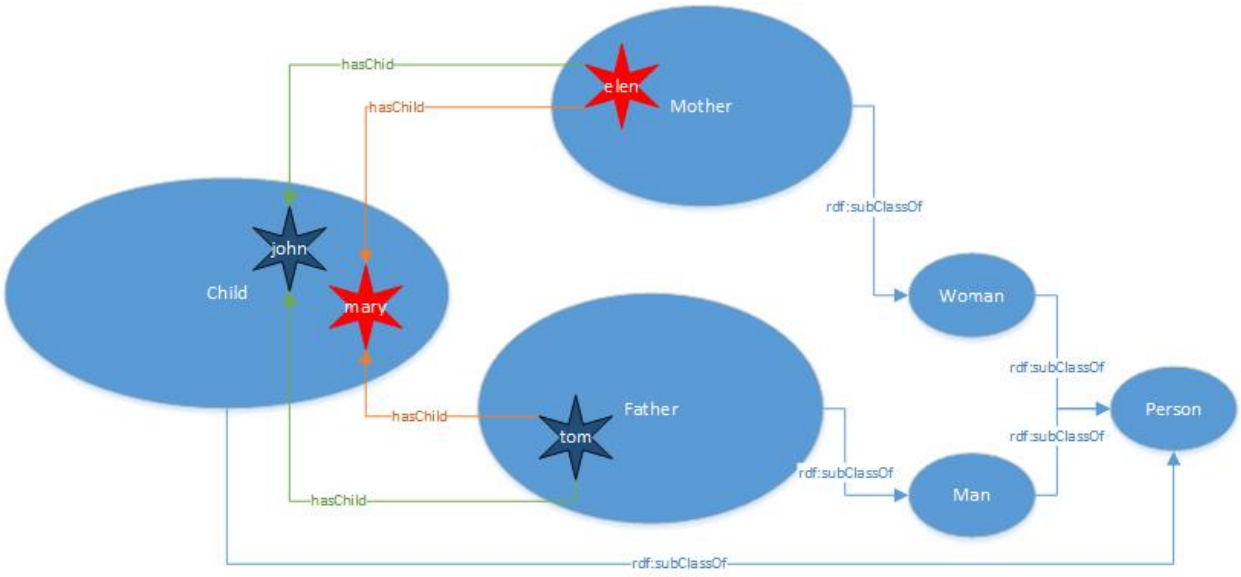


Figure 1

From this drawing in Figure 1 we can distinguish

- Classes: Person, Woman, Man, Father, Mother and Child
- Property: rdf:subClassOf and hasChild
- Individuals: *john*, *mary*, tom and *elen*

We have also defined the following concepts:

- A woman is a subclass of person
- A man is a subclass of person
- A mother is a subclass of woman and has a child
- A father is a subclass of Man and has a child
- A child is subclass of a person

We can easily extend the ontology by adding another concept, for example: a sibling has the same mother or father. Therefore we can logically conclude and infer that *john* and *mary* are siblings. It is possible for the ontology to use a "Semantic Reasoner"[5] for inferring all the logical consequences.

In the ontology "Family" example mentioned above we have added the individuals to specific classes that are part of the ontology. We have categorised and chosen the ontology class that best represents its individuals. It would be better if this process of individual categorisation are done automatically, especially if the individuals (URL[6] resources) are hundreds or thousands or greater.

For example we can define another ontology for the financial products by using the following concepts:

- Portfolio and Fund are sub classes of Financial

[5] http://en.wikipedia.org/wiki/Semantic_reasoner
[6] http://en.wikipedia.org/wiki/Uniform_resource_locator

- Account is sub class of Portfolio
- ISA and SIPP are sub classes of Account

The graph above is a practical example of an ontology model. We can note that all the classes are interconnected to each other by property. For example Account “rdf:subClassOf” of Portfolio and Portfolio “rdf:subClassOf” Financial class.

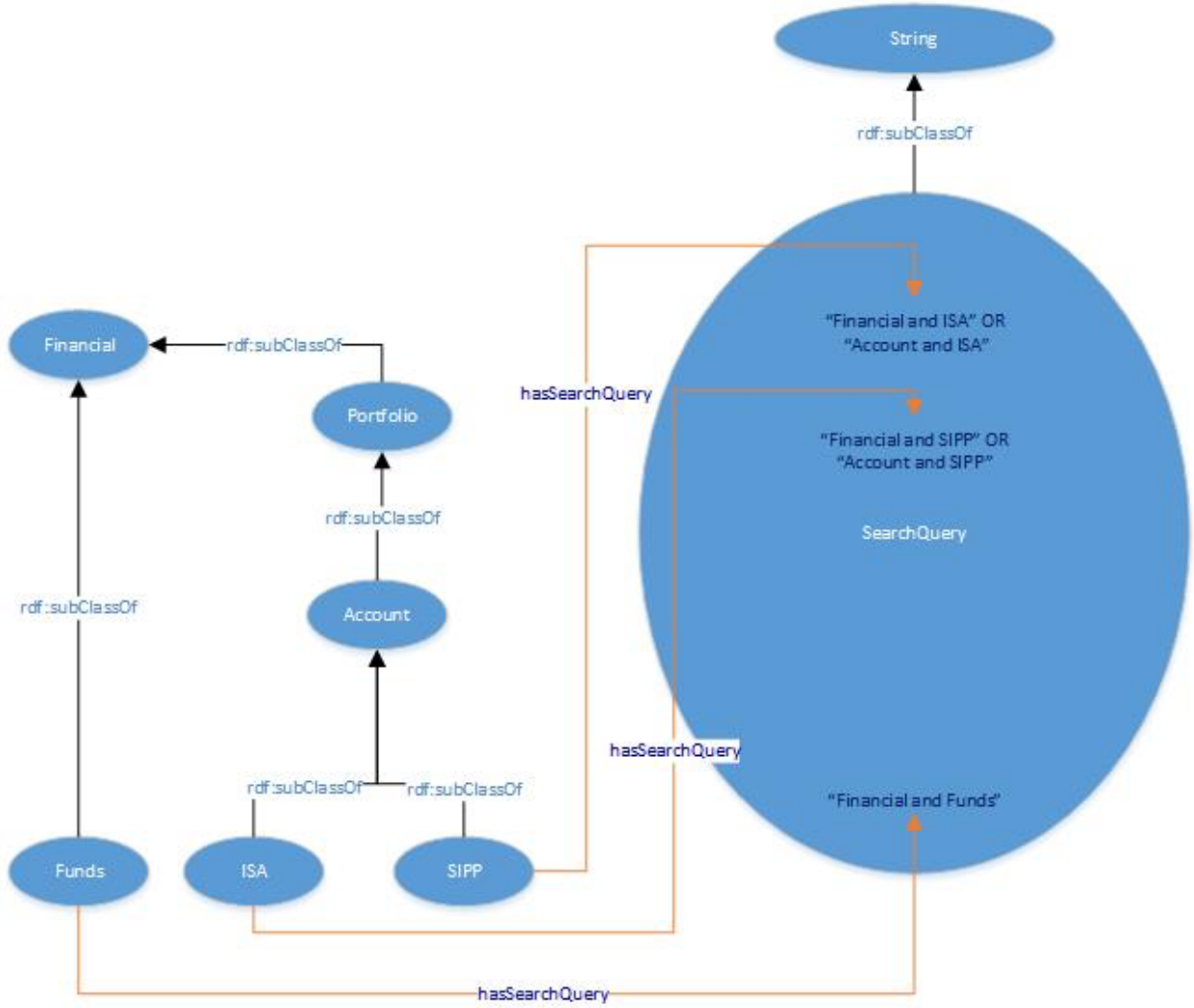


Figure 2

If we want to include in the ontology all the individuals in the classes Fund, ISA and SIPP, we need to navigate the web and search for all the documents that are relevant to the topic of our research. For example all the pages where the financial company offers to open an account that can be an ISA or SIPP or maybe all the financial sites where is possible to buy funds. Searching for all the individual resources to add to our ontology is time consuming.

The aim of this project is to create a system that is able to add individuals automatically to ontology[7] class models.

For instance it would be very useful to instruct a search engine that automatically searches and adds the individual resources to the target ontology classes.

To do this, it is necessary to develop a search engine that when an ontology is inputted, it will return the same ontology with classes that contain the individual elements we are searching for.

These are the most important components to create the proposed search engine:

- Web crawler to store in a database triples
- OWL[8] data manager and provider
- Document indexer and data analyser
- Web Application
- Search Engine

Web Crawler

The web crawler is used to read all the links contained in a web page and adds them, as resources, to an OWL ontology model. A crawler is used to extracts and represents as matrix the interconnections between pages in the web. The matrix result is used to calculate the page ranking of each web resource. For example the above can express a simple web schema links between pages:

- "A" links out "B" and "C", so that "B" and "C" have a link in from "A";
- "B" links out "C", so that "C" has a link in from "B"
- "C" links out "A", so that "A" has a link in from "C"

	A	B	C
A	0	0	1
B	1	0	0
C	1	1	0

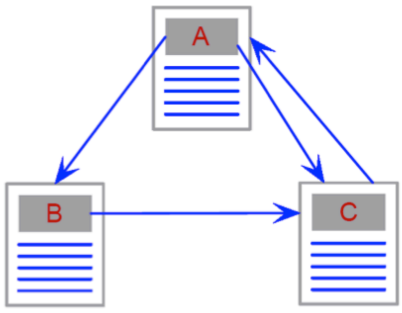


Figure 3

The Figure 3 above is showing how the crawler normally can represent a simple web resources network.

Instead of using a matrix, it is possible to use an ontology model to represent the example above, where each URL will be linked one to the other using simple RDF triples.

[7] http://semanticweb.org/wiki/Ontology
[8] http://www.w3.org/2001/sw/wiki/OWL

The common triple can be represented by the following concept, where the interpretation of the domain is:

$\mathrm{I} = \left(\Delta^{\mathrm{I}}, \cdot^{\mathrm{I}}\right), \quad \mathrm{KB} = \mathcal{T}\mathrm{box} + \mathcal{A}\mathrm{Box}, \quad \mathrm{I} \vDash \mathrm{KB}$

$\mathcal{T}$box:

$\top \sqsubseteq \mathrm{Document}$

$\top \sqsubseteq \mathrm{Mediatype}$

$\mathrm{Document} \sqcup \mathrm{Mediatype} \sqsubseteq \mathrm{WebResource}$

$\mathrm{Html} \sqcup \mathrm{Pdf} \sqcup \mathrm{Doc} \sqcup \mathrm{Docx} \sqcup \mathrm{Ppd} \sqcup \mathrm{zip} \sqsubseteq \mathrm{WebResource}$

$\exists \mathrm{linksOut} . \top \sqsubseteq \mathrm{WebResource}$

$\top \sqsubseteq \forall \mathrm{linksOut} . \mathrm{WebResource}$

$\exists \mathrm{hasLinkIn} . \top \sqsubseteq \mathrm{WebResource}$

$\top \sqsubseteq \forall \mathrm{hasLinkIn} . \mathrm{WebResource}$

$\mathrm{linksOut}^{-} \equiv \mathrm{hasLinkIn}$

$\mathcal{A}$Box: All the assertions and individual resources will be populated during the web crawling.

In the ontology described above all the individuals contained in the class *WebResource* are all the URLs that have been crawled.

For example, in the Figure 3 above, "URL-A -> linksOut -> URL-B" therefore "URL-B -> hasLinkIn -> URL-A". In this case "***linksOut***" and "***haslinkIn***" are two properties where the domain and the range are equal and these are represented by the class *WebResource*. It is also possible to note that the ***haslinkIn*** is the inverse function of **linksOut**.

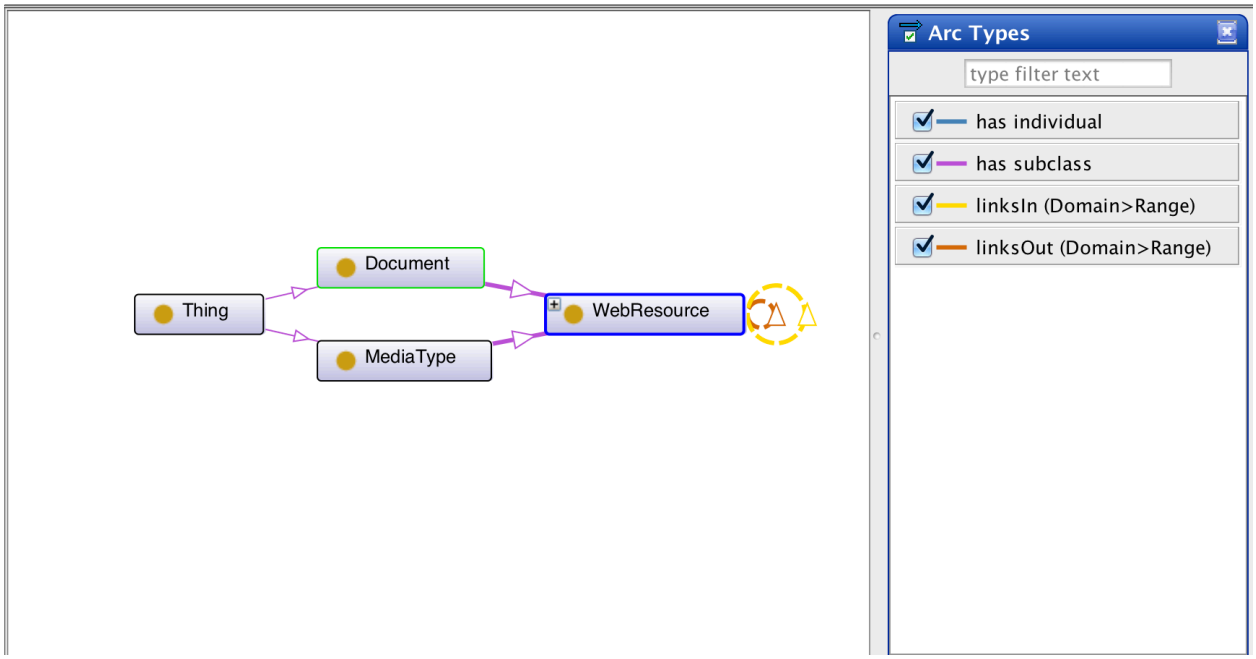


Figure 4

Figure 4 is an example of how this ontology makes it possible to navigate and query, by using SPARQL[9], the individual web resources data that are represented by the entire URL page crawled.

RDF data manager and provider

The RDF data manager is an important component in the project. By using it, it is possible to create the model and add this into all the resources that the crawler has found during its research. The second and most important phase is to store the ontology in a database or in a NoSQL[10] container. The resources should be updatable as well. For instance, the crawling of a particular URL should be periodical and the period of update is decided by the variation of its contents in the time. The content of a Newspaper is updated continuously, whereas a recipe book on the web is not updated so often. Therefore, in base of this principle some URL will be re-crawled and updated more often than others.

Below there is a possible schema database configuration based on nodes and triples.

Tables	Description
nodes	it contains all the resources nodes
triples	it contains all the triples represented subject (s), predicate (p), object(o)

In this case a single node represents a resource and it is an element that composes a triple. Therefore, it can be one of the elements in the list below:

- subject

[9] http://en.wikipedia.org/wiki/SPARQL
[10] http://en.wikipedia.org/wiki/NoSQL

- predicate
- object

Document Indexer and data analyser

Each resource node previously processed by the crawler and stored in the table will be indexed to create a common structure to use as information retrieval. The individual RDF triple resources (object or subject), during this step will be index to create a quick information retrieval. This is an important part of all the system because it will make it possible to analyse and create matching data research. It is also important that the document will be deeply analysed to find any kind of information on it. This module should be also able to find similarities between different resources to create common groups of them. This part of the project will use a stable and solid software indexer as Apache Solr[11] that has been already created for this purposes.

Web Application

The web application will be the interface between the user and the search engine. To use the application, it will be necessary to have an account. Each user will be able to create different projects. Each project will contain URLs to use as data retriever and different ontology model to use as a main query of research. It will be possible through the web application to manage different projects by adding or deleting URLs or selecting URLs that has been already crawled.

The picture below (Figure 5) is a proposed conceptual web site schema:

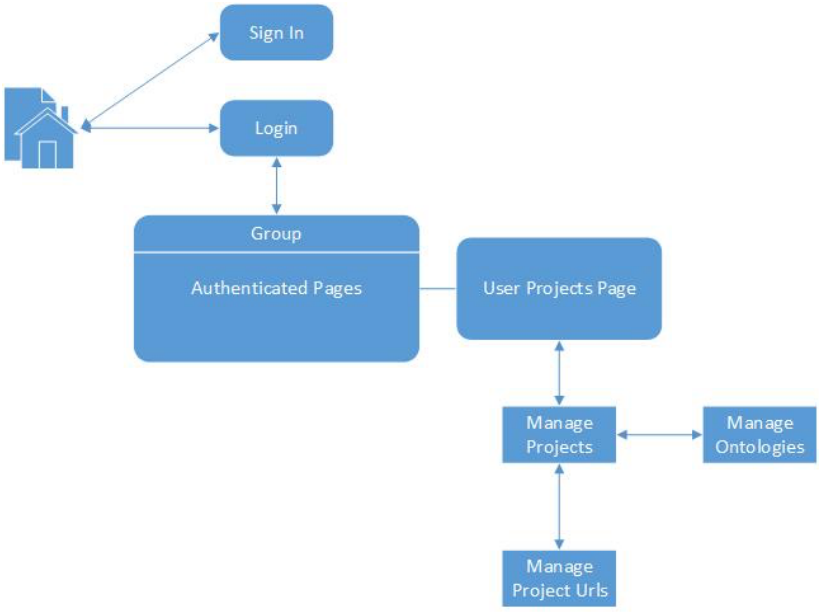


Figure 5

The uploaded ontology will be the base model for a web research. The web page will be the user interface for managing the web ontology searcher. All the requests to the server will be asynchronously

[11] http://lucene.apache.org/solr/

processed. When the web search is completed, the ontology model will be ready to download and use. It will be possible to see in a grid all the process requested and download the completed.

Search engine

The search engine is the last and most important part of the system. This, instead of returning a sequence of rows as result, it will return the ontology fulfil with individuals.

A question to ask ourselves may be: how to use an ontology model as a query?

This can be achieved by using in the ontology model the property "**hasSearchQuery**". This property will be used as a predicate between the class and the "specific search query":

∃hasSearchQuery.{"query to search"} .

With this the search engine we will be able to create and fill our ontology classes.

The advantage to adopt an ontology model as a query of research is that the ontology can contain a Terminology (TBox) and Data Assertion (ABox). In our case, the TBox will be created by the user and the ABox will be created by the search engine to complete the knowledge base requested.

Another option may be that when the ontology is completed with TBox and ABox, it will be possible to use this ontology as a training set for other ontologies where it is not defined in the property **∃hasSearchQuery.** This option will automatize the process of Abox creation.

Current Researchers

There are many researches focused on Semantic Web because this is a new technology that needs to be fully explored. The Semantic Web technologies are not widely used and therefore are not very popular. Many researches have been carried out in this field, especially on the relationship between resources on the World Wild Web (www).

Many companies and institutions have started to study this new technology, such as important web realities like Google[12], Microsoft[13], Wikipedia[14]. They have started to use it in large scale. For instance Wikipedia has created DBpedia[15]. This is a huge ontology that represents Wikipedia itself and it is possible to query all the information that it contains using SPARQL query language.

There are many researchers in this field. Most of them are focused on linking the data together as part of this project is doing.

Where to apply it

It is possible to apply this approach of creating search ontologies in many contexts.

For instance, suppose that we have a web site that sells pizza online and we want to improve our web site by researching other web competitors.

[12] https://www.google.co.uk/
[13] http://www.microsoft.com
[14] https://www.wikipedia.org
[15] http://dbpedia.org/About

First of all we need to create a specific ontology for our research about pizzas. For example the ontology can specify the concept of pizza, with ingredients and different types of pizza. It is important that the property *hasSearchQuery* is defined in all the classes that we need to be populated with individuals by the search engine.

Secondly, on the web application we need to create a new project and add all the links that need to be crawled. These steps are necessary to define the domain of research. This will enable us to extend our research on URLs/domains which are already present on the database and that have been crawled.

Finally, in the project that has been created, we will upload our ontology of research. When the service engine has finished crawling and indexing the documents it will return an ontology that will contain all the results that are matching the query present in the ontology.

At this point the ontology will be full of individuals ($\mathcal{A}$Box defined) so it will be possible to execute different SPARQL queries to search through the data and use a semantic reasoner[16] to resolve all the logical consequences[17]($\mathcal{L}$) generated in the ontology model.

A possible ontology can be the following:

Ontology schema

Self-standing	Relations	Definable	Modifier

• Offers
  ○ Latest
  ○ Deal
• Product
  ○ Pizza
  ▪ PizzaSlice
  ○ Drink
• PostCode
• Location
  ○ ShopAddress
• Order
  ○ Collected
  ○ Delivered
• Money
  ○ Pounds
• Query
	• hasSearchQuery
	• identifies
	• hasSlices
	• isPartOf
	• costs
	• searches
	• pays
		• PizzaSize
		  ○ PizzaSmallSize
		  ○ PizzaMediumSize
		  ○ PizzaLargeSize
		  ○ PizzaExtraLargeSize
		• Price
		• Menu
		  ○ MenuItem
			• PizzaCategory
			  ○ PizzaVegetarian
			  ○ PizzaBiological
			  ○ PizzaGlutenFree

Relations

Property Name	Domain	Range	Type
hasSearchQuery	T	Query	not functional not symmetric not transitive
identifies	PostCode	Location	not functional Symmetric not transitive

[16] http://en.wikipedia.org/wiki/Semantic_reasoner
[17] http://en.wikipedia.org/wiki/Logical_consequence

hasSlices	Pizza	PizzaSlice	not functional not transitive not symmetric
isPartOf	PizzaSlice	Pizza	inverse of hasSlices
hasPrice	Product	Price	functional not symmetric not transitive
isValueOf	Price	Product	functional not symmetric not transitive
searches	Query	T	not functional not symmetric not transitive
pays	Money	Order	not symmetric not functional not transitive
isPaidBy	Order	Money	inverse of pays

Definable

$PizzaSize \sqsubseteq PizzaSmallSize \sqcup PizzaMediumSize \sqcup PizzaLargeSize \sqcup PizzaExtraLarge$Size

$SmallSize \equiv Pizza \sqcap = 4hasSlices.PizzaSlice$

$MediumSize \equiv Pizza \sqcap = 6hasSlices.PizzaSlice$

$LargeSize \equiv Pizza \sqcap = 8hasSlices.PizzaSlice$

$ExtraLargeSize \equiv Pizza \sqcap = 12hasSlices.PizzaSlice)$

$Price \equiv \exists isValueOf.Product$

$Menu \sqsubseteq MenuItem \sqcup \exists hasPrice$

Specify the relation hasSearchQuery

$Offer \sqsubseteq \exists hasSearchQuery.\{"Pizza\ and\ online\ and\ offer"\}$

$Latest \sqsubseteq Offer \sqcap \exists hasSearchQuery.\{"Latest","new"\}$

$Deal \sqsubseteq Offer \sqcap \exists hasSearchQuery.\{"Deal"\}$

$Product \sqsubseteq Offer \sqcap \exists hasSearchQuery.\{"Pizza\ and\ online"\}$

$Menu \sqsubseteq Pizza \sqcap \exists hasSearchQuery.\{$"Menu"$\}$

$Price \sqsubseteq Pizza \sqcap \exists hasSearchQuery.\{$"Price"$\}$

$Pizza \sqsubseteq Product \sqcap \exists hasSearchQuery.\{"Pizza"\}$

$Pizza \sqsubseteq Product \sqcap \exists hasSearchQuery.\{"Drink"\}$

$Location \sqsubseteq \exists hasSearchQuery.\{"Pizza\ and\ online\ and\ (contact\ or\ address)",\}$

$Order \sqsubseteq \exists hasSearchQuery.\{"Pizza\ and\ online\ and\ order",\}$

$Collected \sqsubseteq Order \sqcap \exists hasSearchQuery.\{$"collect*", "take away"$\}$

$Delivered \sqsubseteq Order \sqcap \exists hasSearchQuery.\{$"deliver*"$\}$

$SmallSize \sqsubseteq Pizza \sqcap \exists hasSearchQuery.\{$"4 and slice*, "4 and portions"$\}$

$MediumSize \sqsubseteq Pizza \sqcap \exists hasSearchQuery.\{$"6 and slice*", "6 and portions"$\}$

$LargeSize \sqsubseteq Pizza \sqcap \exists hasSearchQuery.\{$"8 and slice*", "8 and portions"$\}$

$ExtraLargeSize \sqsubseteq Pizza \sqcap \exists hasSearchQuery.\{$"12 and slice*", "12 and portions"$\}$

$Vegetarian \sqsubseteq Pizza \sqcap \exists hasSearchQuery.\{$"Vegetar*"$\}$

$Biological \sqsubseteq Pizza \sqcap \exists hasSearchQuery.\{$"Bio*"$\}$

$Gluten \sqsubseteq Pizza \sqcap \exists hasSearchQuery.\{$"gluten*"$\}$

How it works

As explained in the first chapter we need to have five different modules to make the search engine able to populate the model ontology of individual web resources such as:

- Web crawler
- OWL data manager and provider
- Document indexer and data analyser
- Web Application
- Search Engine

The first on the list above is the crawler module and it will be a java application. This application will read a table in the database that contains the URLs to crawler.

The grid below specifies the data set structure for the table “UrlToCraw”:

Name	Type	Length	Default	Note
IdUrlToCrawl	NUMERIC	11		Primary Key
UrlToCrawlVal	VARCHAR	2083		Not Null
IsToCrawl	Bit	1	1	Not Null
IsToPersist	Bit	1	1	Not Null
IsToIndex	Bit	1	1	Not Null
OwlFileName	VARCHAR	255		Null
IsActive	Bit	1	0	Not Null
Created	DateTime	8		Not Null
Modified	DateTime	8		Not Null

In the web application, during the creation of a project, it will be possible to insert the link to crawl in the database. The crawling of the links does not start immediately because each one of this links will be

inserted in a queue[18] process. Each document found during the crawling will be indexed using Apache Solr[19]. Apache Sorl creates an inverted document index, for all the information that each document page contains. This module will be an important part of the system because it will make it possible to submit a query to Apache Solr and receive all the relevant documents related to the submitted query as a response. Each of the relevant documents received will be inserted as an individual in the uploaded ontology model.

Methodology and work plan

The methodology for developing this project will be AGILE[20]/SCRUM[21]. The entire project will be split into different stories and each story will have many tasks and subtasks. This will make it easier to plan time spent on the project and checking its progress. This also allows the creation of clear specifications that the software needs to be developed. The specifications will be used to define the tasks of all the story project. This will be useful during all the implementation of the Test Driven Development[22] (TDD) used to build the entire project application.

The project's SCRUM stories will be represented by the main module components that need to be developed. These modules are:

- Web crawler
- OWL data manager and provider
- Document indexer and data analyser
- Web Application
- Search Engine

Now we will create the story for each module.

Web crawler

To develop this part it is necessary to split this SCRUM story in three tasks: Service Crawler Engine, Crawler library, End-to-End tests[23].

The Service will be continuously looking up the database table UrlToCraw, to find entries that have not been crawled (IsToCrawl=1). To do that it sends a message to a Crawler actor[24], which crawls the URL and creates the OWL file.

The Crawler library will crawl and create the linking data (Abox) as ontology representation of any URLs. Apache Jena[25] API will be used to generate the OWL file. This file will be stored in a specific server folder. After that it will be possible to update the table UrlToCraw by setting these fields:

- IsToCrawl equal to 0 and
- OwlFileName equal to the name of how the file has been saved.

[18] http://en.wikipedia.org/wiki/Queue_(abstract_data_type)
[19] http://lucene.apache.org/solr/
[20] http://en.wikipedia.org/wiki/Agile_software_development
[21] http://en.wikipedia.org/wiki/Scrum_(software_development)
[22] http://en.wikipedia.org/wiki/Test-driven_development
[23] https://msdn.microsoft.com/en-gb/library/cc194885.aspx
[24] http://doc.akka.io/docs/akka/snapshot/scala/actors.html
[25] https://jena.apache.org

The End-to-End test will test and verify that the story follows the specifications requested.

Following the story's tasks and subtask:

- Service Crawler Engine
  - Analysis of requirements
  - Service crawler implementation using TDD
  - Documentation and Report integration
- Crawler library (task)
  - Analysis of the requirements
  - Crawler Software implementation using TDD
  - Documentation and Report integration
  - Functional tests
- End to end tests

Triple data manager and provider

The OWL data manager provider should be split in three tasks as well: service provider for ontology files, triple data provider, End-to-End Test.

The service provider for ontology files looks up the new ontology files that have been created. To do that, the service looks up the table UrlToCraw and checks all the records that has been crawled (IsToCrawl = 0). For each of these records the *service provider for ontology* should send a message to the *triple data provider* actor to process and save on database the relative file name (OwlFileName).

The *triple data provider* library should use Apache Jena for saving the triple in the database (TDB[26]). Every time it receive a message it does the following:

1. It loads the OWL ontology file contained in the message
2. It stores the ontology in the database as triple definitions.

Following the SCRUM story's tasks and subtasks:

- Triple data provider
  - Analysis of how to use TDB
  - Implementation using TDD
  - Documentation and Report integration
- Service provider for ontology files
  - Analysis requirements
  - Implementations
  - Documentation and report integration
- End-to-End Test

Document indexer and data analyser

In this SCRUM story it is necessary to use and integrate Apache Solr as indexer engine. In this case it will be necessary to do a very deep analysis of:

[26] https://jena.apache.org

- how to set up Apache Solr,
- how to install and use Apache Solr,
- how to make Apache Solr able to index the necessary WebResources being crawled.

To create a setup for the service, it will be necessary to add a new *core*[27] to the running instance of Solr. This core it is used as a configuration for the new documents index.

Another important phase of this story will be how to send a message to Solr to index a file. The best approach for doing this is to fetch and download the resource that we need to index. When the resource becomes available we will be able to send a message to Solr to index it. To do this we need to read from the database the record-triples that we want to index. These triples will be part of the URLs that has been crawled.

At this point we need to use Apache Jena to execute a SPARQL query on the database and to know all the links that are descendants of the crawled URL. In this case it is possible to use the following query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX j.0: <http://titan.dcs.bbk.ac.uk/~srapis01#>
SELECT  ?o
WHERE { ?s ?p ?o .
        FILTER regex(str(?s), \"http://www.dcs.bbk.ac.uk/~michael\", \"i\")
}
LIMIT 10
```

The query above returns just the first 10 results, which are shown in the table below:

```
-------------------------------------------------------
| o                                                   |
=======================================================
| <http://www.dcs.bbk.ac.uk/~michael/index.body.html>    |
| <http://ceur-ws.org/Vol-1015>                         |
| <http://www.dcs.bbk.ac.uk/~michael/frocos09-final.pdf> |
| <http://www.dcs.bbk.ac.uk/~michael/ISWC-14-v2.pdf>     |
| <http://www.dcs.bbk.ac.uk/~michael/ER07.pdf>           |
| <http://www.dcs.bbk.ac.uk/~michael/dl03final2.ps>      |
| <http://www.illc.uva.nl/tacl09/#URI=%3Fpage%3D_>       |
| <http://www.dcs.bbk.ac.uk/~michael/AiML-08.pdf>        |
| <http://www.dcs.bbk.ac.uk/~michael/fulloct19.ps>       |
| <http://www.dcs.bbk.ac.uk/~michael/fulloct19.ps>       |
-------------------------------------------------------
```

After it will be necessary to fetch the web page resource and after index the page using Solr for each of the resources above. At this point, the process of indexing all the resources will be finished and the table UrlToCraw will be updated by setting the field IsToIndex to 0 (zero) for each of the web resources previously indexed.

The following below are the SCRUM story's tasks and subtasks:

- Solr Analysis of requirements
    - Read documentation
    - How to install an instance of Solr

[27] https://wiki.apache.org/solr/CoreAdmin

    - How to create core configuration.
    - How to send the document to index
    - Documentation and report integration
- Solr configuration
    - Installation and configuration
    - Core creation
    - Testing of all the prerequisites
    - Documentation and report integration
- Service indexer using TDD
    - Fetch and download resources
    - Send message command to Solr
    - Update DB
    - Documentation and report integration
- End-to-End test

Web Application

The web application as explained before will be a simple web user interface (UI) with a login page and another page where it will be possible to manage the user projects. To do this we need that the database contains at least the following tables:

User

Name	Type	Length	Default	Note
UserId	NUMERIC	11		Primary Key
UserName	VARCHAR	20		Not Null
UserEmail	VARCHAR	255		Not Null
IsActive	Bit	1	1	Not Null
Created	DateTime	8		Not Null
Modified	DateTime	8		Not Null

Project

Name	Type	Length	Default	Note
ProjectId	NUMERIC	11		Primary Key
ProjectName	VARCHAR	50		Not Null
ProjectDescription	VARCHAR	1024		Not Null
IsComplete	Bit	1	0	Not Null
IsActive	Bit	1	0	Not Null
Created	DateTime	8		Not Null
Modified	DateTime	8		Not Null

Ontology

Name	Type	Length	Default	Note
OntologyId	NUMERIC	11		Primary Key
OntologyName	VARCHAR	50		Not Null

OntologyDescription	VARCHAR	1024		Not Null
OntologyFileName	VARCHAR	255		Not Null
IsActive	Bit	1	0	Not Null
Created	DateTime	8		Not Null
Modified	DateTime	8		Not Null

Url

Name	Type	Length	Default	Note
UrlId	NUMERIC	11		Primary Key
UrlValue	VARCHAR	2083		Not Null
UrlDescription	VARCHAR	1024		Not Null
IsActive	Bit	1	0	Not Null
Created	DateTime	8		Not Null
Modified	DateTime	8		Not Null

ProjectUrl

Name	Type	Length	Default	Note
ProjectId	VARCHAR	11		Primary Key
UrlId	VARCHAR	11		Primary Key

ProjectOntology

Name	Type	Length	Default	Note
ProjectId	VARCHAR	11		Primary Key
OntologyId	VARCHAR	11		Primary Key

We also need that the web application exposes all the necessary http actions to manage the tables above. The actions will be called by using Ajax[28] requests to the web app. For the client interface the project will use common HTML[29], JavaScript[30], JQuery[31], *less*[32] style CSS[33] generator and AngularJS[34]. The Web App will be a Java application. The user interface will develop using BDD[35] and jasmine[36] library framework.

The web application will be also able to update the table UrlToCrawl by adding new URLs on it.

[28] http://en.wikipedia.org/wiki/Ajax_(programming)
[29] http://en.wikipedia.org/wiki/HTML
[30] http://en.wikipedia.org/wiki/JavaScript
[31] http://jquery.com
[32] http://lesscss.org/
[33] http://en.wikipedia.org/wiki/Cascading_Style_Sheets
[34] https://angularjs.org/
[35] http://en.wikipedia.org/wiki/Behavior-driven_development
[36] http://jasmine.github.io/

The web application will use a data provider to access and update the data tables in the database.

Following the SCRUM story's tasks and subtasks:

- Data Base
    - Schema and structures creation
    - Documentation and Report implementation
- Web Application
    - Analysis and implementation of Web API action
    - Analysis and implementation of Data Provider
    - Design and Implementation of HTLM pages using less CSS and angular JS
    - Ajax call implementation
    - Jasmine unit test
    - Documentation and Report implementation
- End-to-End test

Search Engine

The search engine is the last SCRUM story. For this module we need to have: model reader, query executor, project service. The ontology reader will be able to read the OWL model that has been uploaded in the project. Apache Jena API will be used from the search engine for creating the model and execute a SPARQL query like:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wonts: <http://titan.dcs.bbk.ac.uk/~srapis01#>
SELECT  ?sobject ?object
WHERE { ?sobject wonts:hasSearchQuery ?object . }
```

The SPARQL query above will return all the queries to execute (?object) in Solr and where the resource documents will be associated (?sobject).

It will be necessary to create a message provider to execute query in Solr. The Solr response will be associated in the model by adding individual resources to each mapped class. When the process has ended, a completed OWL ontology file will be saved and the ontology file will be updated.

Below are all the story's tasks and subtasks:

- Analysis of requirements
    - How to create a query to Solr
    - How to process the data
    - How to add the individual to the ontology
    - How to integrate the Db
    - Documentation and Report implementation
- Solr query message provider
    - Send query message to Solr
    - Process response
    - Documentation and Report implementation

    - Functional tests
- Ontology integrator
    - Add individual to ontology class
    - Save ontology
- End-to-End Test

Time Line

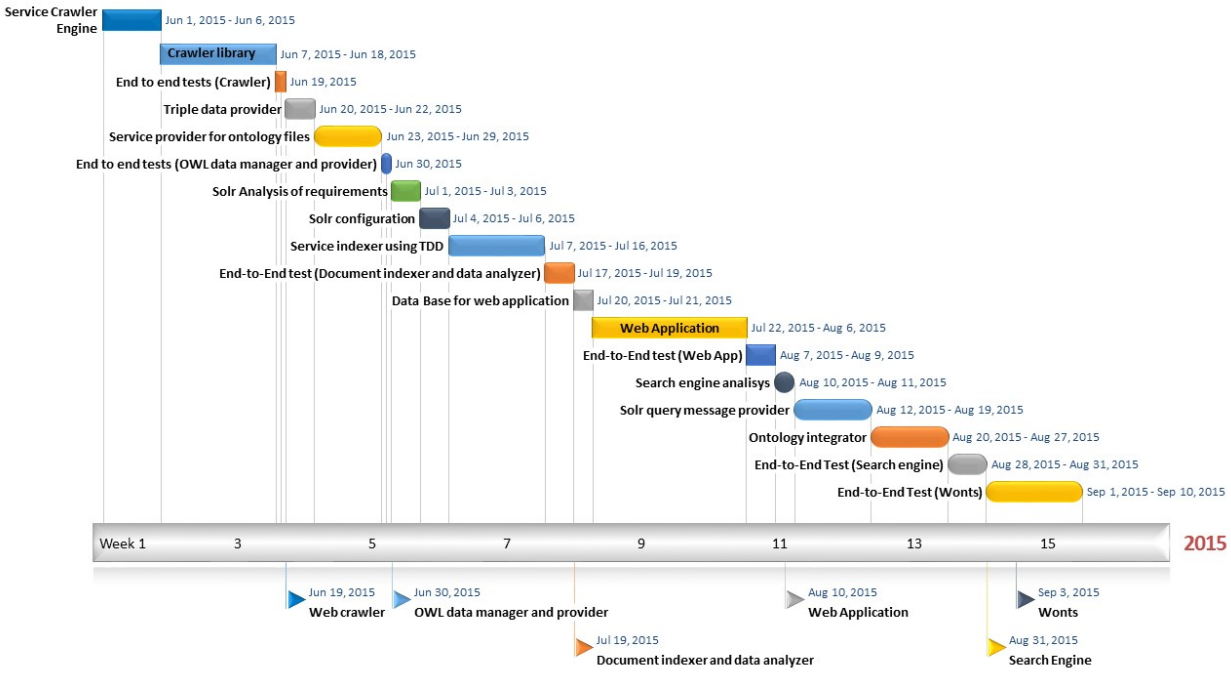
To be sure that the project is completed on time, it is necessary to schedule and plain its development. For each story it is necessary to schedule the time needed to implement software and report. At the end of each task it will be necessary to update the report and the documentation. The table below shows:

- Stories
- Tasks grouped by story
- The start and end date
- The relative working days (Saturday and Sunday excluded).

Story	Task	Start Date End Date	Working Day/s
Web Crawler	Service Crawler Engine	01/06/2015 07/06/2015	5
	Crawler library (task)	07/06/2015 18/06/2015	9
	End to end tests	19/06/2015 19/06/2015	1

OWL data manager and provider	Triple data provider	20/06/2015 22/06/2015	1
	Service provider for ontology files	23/06/2015 29/06/2015	5
	End-to-End Test	30/06/2015 30/06/2015	1
Document indexer and data analyser	Solr Analysis of requirements	01/07/2015 03/07/2015	3
	Solr configuration	04/07/2015 06/07/2015	1
	Service indexer using TDD	07/07/2015 16/07/2015	8
	End-to-End test	17/07/2015 19/07/2015	1
Web Application	Data Base	20/07/2015 21/07/2015	2
	Web Application	22/07/2015 06/08/2015	12
	End-to-End test	07/08/2015 09/08/2015	1
Search Engine	Analysis of requirements	10/08/2015 11/08/2015	2
	Solr query message provider	12/08/2015 19/08/2015	6
	Ontology integrator	20/08/2015 27/08/2015	6
	End-to-End Test	28/08/2015 31/08/2015	2
Wonts	End-to-End Test (wonts)	01/09/2015 10/09/2015	8

As noted in the table above it shows that the project starts 1st of June 2015 and it will end 10th September 2015. The drawing below represents the project time line.



Technologies

The technologies that will be used to develop all the project components are listed below:

- Java as developing languages
- Javascript, jquery, AngularJS, less, jasmine
- SQL and SPARQL
- Akka Actors for concurrencies process execution
- Apache Solar as document index and retrieval
- Apache Jena for ontology, TDB, RDF, OWL
- JIRA as SCRUM Methodology
- TDD and BDD for all the software to develop