

PROYECTO FIN DE CARRERA

Título: Diseño e implementación de un sistema de administración y búsqueda para un metadirectorio de servicios telco

Título (inglés): Design and implementation of an administration system and search engine for a telco services metadirectory

Autor: Pablo Moncada Isla

Tutor: Carlos A. Iglesias Fernández

Departamento: Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente: Gregorio Fernández Fernández

Vocal: Mercedes Garijo Ayestarán

Secretario: Carlos Ángel Iglesias Fernández

Suplente: Tomás Robles Valladares

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes



PROYECTO FIN DE CARRERA

**DESIGN AND IMPLEMENTATION OF AN
ADMINISTRATION AND SEARCH SYSTEM
FOR A TELCO SERVICES
METADIRECTORY**

Pablo Moncada Isla

Marzo de 2014

Resumen

Esta memoria es el resultado de un proyecto cuyo objetivo ha sido realizar un repositorio semántico de servicios y widgets.

Dicho repositorio tiene la facultad de poder ser rellenado de contenido de forma automática gracias a técnicas de descubrimiento automático en Internet, haciendo uso de arañas o scrappers que recogen el contenido y posteriormente lo convierten a formato estructurado o semántico.

Para que dicho repositorio tenga contenido de utilidad se han presentado algoritmos que son capaces de calificar los servicios y además se ha desarrollado una herramienta que permite organizar y administrar todo el contenido extraído por las herramientas de descubrimiento automático.

Posteriormente se ha presentado otra herramienta que permite al usuario final explorar el repositorio de servicios y ayudarle a buscar y a encontrar el contenido deseado mediante técnicas de búsqueda semántica.

Por último, se han presentado las conclusiones extraídas del trabajo, las posibles líneas de continuación del proyecto, así como los siguientes pasos en cuanto a desarrollo y aprovechamiento de la plataforma.

Palabras clave: Tecnologías semánticas, Linked data, OpenRDF Sesame, Linked Media Framework, RDF, SPARQL, PHP, JavaScript, Java, Knowckout JS

Abstract

This thesis is the result of a project whose objective is to develop and deploy a semantic repository of services and widgets.

The repository has the faculty of been populated of content in a automated way thanks to Internet automated discovery techniques. Scrappers would fetch the content and after this it would be converted into structured or semantic data.

In order the repository to be useful we will present algorithms that would be able to rank services and widgets. We also present a tool that allow us to organize and administer the content extracted by the automated discovery tools.

To continue, another tool will be presented. It will allow the final user explore the service repository and will help the user to search and find the desired content through semantic search techniques.

Finally, we gather the extracted conclusions plus some lessons learnt, the possible line of work regarding the continuance of the platform as well as the next step regarding development and exploitation of the service.

Keywords: Semantic technologies, Linked data, OpenRDF Sesame, Linked Media Framework, RDF, SPARQL, PHP, JavaScript, Java, Knowckout JS

Agradecimientos

Quiero aprovechar estas líneas para agradecer a toda la gente que me ha acompañado durante este magnífico *viaje* que tanto he disfrutado.

A mi familia, por su apoyo incondicional y por sus intentos de entender siempre a lo que me dedicaba, a pesar de que raras veces comprendían de que hablaba.

A mis amigos y compañeros de la *Escuela*. A todos aquellos con los que me he cruzado en el camino y me han permitido aprender de ellos. En especial a mis amigos Adrián, Marcos, Juan Fernando, David, Gonzalo y con mención de honor a Beatriz.

Gracias también a mis compañeros de Delegación de Alumnos y de Eurielec, con ellos he aprendido un montón y vivido experiencias inolvidables.

A todos los del Grupo de Sistemas Inteligentes y en especial a Carlos, que siempre ha confiado en mí y me ha apoyado.

A cualquiera que leyendo estas líneas sabe que ha significado algo para mí.

Contents

Resumen	V
Abstract	VII
Agradecimientos	IX
Contents	XI
List of Figures	XVII
List of Tables	XXI
1 Introduction	1
1.1 Context	3
1.2 Master thesis description	3
1.3 Master thesis goals	5
1.4 Structure of this Master Thesis	5
2 Enabling Technologies	7
2.1 Overview	9
2.2 OMELETTE mash-up Registry	10
2.2.1 RDF model	10
2.3 Automated Discovery	13
2.3.1 Introduction	13
2.3.2 Discovery techniques	17

2.4	Conclusions	20
3	Requirement Analysis	21
3.1	Overview	23
3.2	Use cases	23
3.2.1	Actors dictionary	23
3.2.2	OMR composition and search use case	25
3.2.2.1	Keyword search	26
3.2.2.2	Mash-up browse by category	27
3.2.2.3	mash-up search by query	28
3.2.2.4	mash-up compose	29
3.2.2.5	Ask suggestion	30
3.2.3	OMR discovery and administration use case	31
3.2.3.1	Automatic mash-up feeding	32
3.2.3.2	HTML Form discovery and description	33
3.2.3.3	mash-up registry integration	34
3.2.3.4	API based integration	35
3.2.3.5	Scraping based integration	36
3.2.3.6	Manual mash-up management	37
3.2.3.7	Browse pending mash-ups and services	38
3.2.3.8	Validate a mash-up / service	39
3.2.3.9	Reject a mash-up / service	40
3.2.4	Web interface use case	41
3.2.4.1	Request available mash-ups	42
3.2.4.2	Search mash-ups by query	43
3.2.5	Conclusions	43

4	Architecture	45
4.1	Introduction	47
4.2	Automated discovery	48
4.3	Semantic repository	50
4.4	Ranking module	51
4.5	OMR Admin Interface	52
4.5.1	Main component	54
4.5.2	Functions library	54
4.5.2.1	Facet boxes	55
4.5.2.2	Result lists	56
4.5.2.3	Widget or service	56
4.5.2.4	Generate charts	56
4.5.2.5	Admin authentication	57
4.5.2.6	Cache and performance	57
4.5.2.7	Repository wrapper	60
4.5.2.8	Sparql Library	60
4.5.2.9	Validating resource	62
4.5.2.10	Rejecting resource	63
4.5.2.11	Wadl generation	64
4.5.2.12	LMF integration	66
4.6	OMR Client Browser	67
4.6.1	Back-end	67
4.6.2	Front-end	69
4.7	Conclusions	71
5	Prototype and example usage	73
5.1	Introduction	75

5.2	Automatic service discovery	77
5.3	Ranking algorithm	81
5.4	OMR administrative interface	82
5.4.1	Available general actions	83
5.4.1.1	Filtering	83
5.4.1.2	Searching	83
5.4.1.3	Obtaining help	84
5.4.1.4	Statistics	84
5.4.2	Reject a resource	84
5.4.3	Validating resource example	85
5.5	Web developer interface	86
5.6	Conclusions	89
6	Conclusions and future lines	91
6.1	Conclusions	93
6.2	Achieved goals	93
6.3	Future work	94
A	Installing and configuring Scrappy	97
A.1	Installation	97
A.1.1	Requirements	97
A.1.2	Installation steps	98
A.2	User manual	99
A.2.1	Command line interface	100
A.2.2	Web admin interface	100
A.2.3	Web service interface	101
A.2.4	Ruby interface	102

A.2.5	Integration with Sesame	103
A.2.6	Extractors	104
B	OMELETTE Mashup Registry (OMR)	113
B.1	Installation of Sesame with uSeekM (+PostgreSQL +PostGIS)	113
B.2	Installation of the DataGridService	116
B.3	User manual	117
C	OMR Administrative interface	121
C.1	Installation and configuration	121
D	OMELETTE Ranking System	123
	Bibliography	125

List of Figures

1.1	Omelette general picture	4
2.1	Omelette mash-up Registry RDF model	11
2.2	ProgrammableWeb	14
2.3	Yahoo Pipes	15
2.4	Opera Widgets	16
2.5	Mapping example for data extraction	17
2.6	Mapping example for data extraction	18
2.7	Execution page of a Yahoo Pipe's mash-up	19
3.1	Composition and Search use case	25
3.2	OMR discovery and administration use case	31
3.3	Web interface use case	41
4.1	General Architecture	48
4.2	Scrappy sequence diagram	49
4.3	Sequence diagram for ranking index generation	52
4.4	Simile Exhibit example interface	53
4.5	OMR Admin Interface architecture	53
4.6	OMR Admin Interface	54
4.7	Generating Facet Boxes	55
4.8	Pie chart	57
4.9	Sequence diagram for chart generation	58

4.10	Sequence diagram if query is not in cache	59
4.11	Sequence diagram if query cached	59
4.12	OMR admin top menu	62
4.13	OMR admin validate and reject buttons	62
4.14	Sequence diagram for validating RDF resource	63
4.15	OMR admin search box	63
4.16	Sequence diagram for rejecting RDF resource	64
4.17	Sequence diagram for WADL file generation	66
4.18	Scrappy sequence diagram	68
4.19	OMR client interface	68
4.20	Search fields in OMR client interface	70
5.1	Original site and corresponding LiMOn mapping	79
5.2	Main view OMR Administrator	82
5.3	OMR admin top menu	82
5.4	Filtering by LiMOn properties	83
5.5	OMR admin search box	83
5.6	OMR admin help	84
5.7	OMR see statistics	84
5.8	Bad charset encoding in widget description	85
5.9	Google maps api service in admin interface	85
5.10	OMR Developer interface main	86
5.11	Search service button	87
5.12	Select saved search or create a new one	87
5.13	Select filters for the search	87
5.14	Show results	88
5.15	Found services by the semantic module	88

5.16	Extended service info	89
A.1	Scrappy command line interface	101
A.2	Web admin interface of Scrappy	102
A.3	Extractors Admin Interface	102

List of Tables

3.1	Actors list	24
4.1	Runnable components in OMR	50
4.2	Execution query time comparison	58
4.3	Sparqllib output example	61
5.1	Actors list	75
5.2	Execution enviroment	76
5.3	Scrappy execution statistics	77
5.4	OMR components summary	77
5.5	Ranking algorithm execution statistics	81

Introduction

This chapters provides an introduction to the problem which will be approached in this project. It provides an overview of the benefits of mash-ups and linked data technologies. Furthermore, a deeper description of the project and its environment is also given.

1.1 Context

The convergence of Telecom, IT and content services drives new emerging service markets based on an open Internet of Services. mash-ups have gained big success in the so-called Web 2.0. The success of the Web 2.0 services has encouraged Telcos to expose their services as Telco mash-ups, in order to provide third parties with facilities to build their business. Moreover, the exposure of network infrastructure as services is facilitating the entry of new API-driven telco agents that bring traditional telco services (telephony, messaging, IP location, etc.) to the Web.

Yet, the technologies underlying each of the different mash-up types are heterogeneous, which makes integration challenging. Also, mash-ups do not offer a universal composition model either, since mash-up development is not vendor independent. A mash-up developed within a specific technology has to be re-coded in order to be deployed in another engine.

This master thesis is developed as part of OMELETTE [1] project which in turn is part of a FP7¹ project that aims at researching on the development, management, governance, execution and conception of converged services with a specific focus on the Telco domain. OMELETTE will create a sound model of mash-ups that follows the REST architectural style (also supported by standard widget technology), as well as a standard specification of a mash-up-containing platform that may guarantee portability and interoperability among different vendors and versions. These concepts will be based on a solid theoretical model of mash-up foundations and the specific requirements gained from the telco domain. OMELETTE will foster as well the reuse of existing components and mash-ups, thanks to its automated service discovery functionalities. Project OMELETTE aims at developing an open platform for building convergent mash-ups for the telco domain to be used within several industry-driven use cases.

1.2 Master thesis description

OMELETTE project provides end users an environment for developing and running widgets and mash-ups. Users can create mash-ups with a mash-up editor and then run these mash-ups in the Live OMELETTE Environment as depicted in figure 1.1.

The Live OMELETTE Environment interacts with the Service mash-up Environment (for direct deployment of new mash-ups), with the web, and with the OMELETTE Infor-

¹FP7-ICT-2009-5

mation Store (semantic storage of existing widgets and services).

For this master thesis **we are going to focus on how to discover and manage widgets and services** to develop new mash-ups (OMELETTE Information Store). We will integrate discovering techniques with the OMELETTE mash-up register (OMR) which is already developed and deployed.

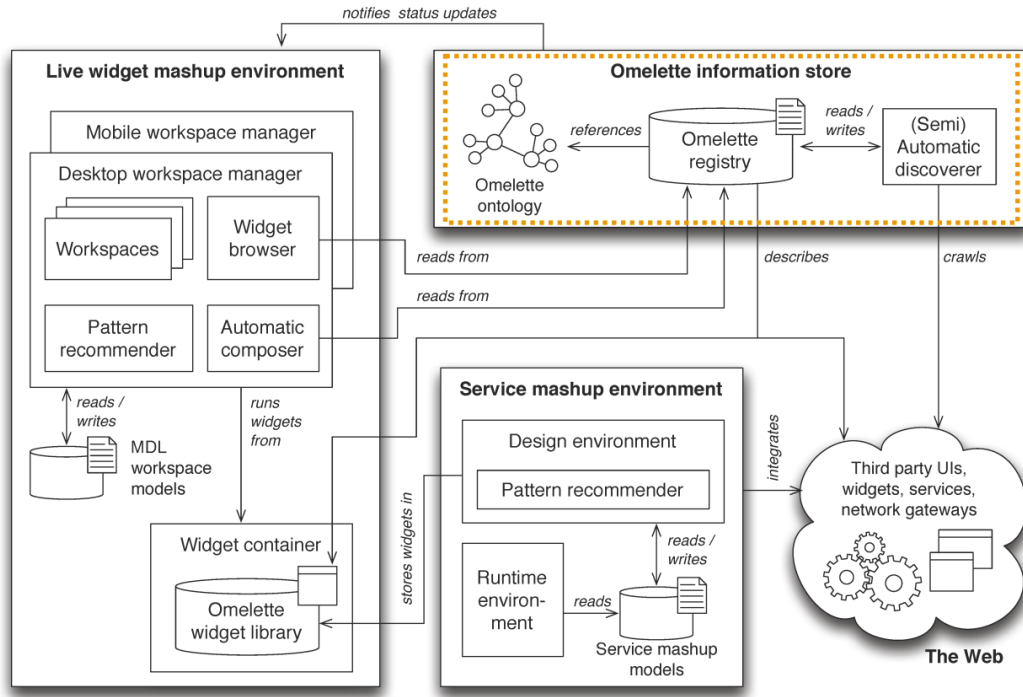


Figure 1.1: Omelette general picture

Inside the OMELETTE information store we distinguish the following modules:

OMELETTE mash-up Registry is the element that registers components for their usage by the rest of elements in the OMELETTE platform. A component can be a mash-up, a service, or a widget. Component descriptions and accompanying binary content will be stored in the OMR so that other elements from the OMELETTE architecture can query the OMR for them and use them. To describe components, a unified RDF component model has been defined. This unified component model provides a unified way to query components and identify, e.g., appropriate widgets for composition, interesting new services to be used when creating a widget, or relevant mash-ups of a particular domain.

Automatic Discoverer is the system responsible for populating the OMR with up-to-date components. In the current Web plenty of services and widgets are released every day,

and developers need to be aware of these services in order to build state-of-the-art mash-ups. To achieve this, a module that crawls service and widget repositories registers these components into the OMR. This automatic discoverer produces semantic descriptions of the components found in the Web out of the unstructured HTML documents they are contained into.

OMR admin interface will allow to organize the components fetched by the automatic discoverer. Some of the components inserted into the OMR might be undesirable, and the administrator user within the admin interface will be responsible filter them.

Web developer interface will be used by final developer users allowing them to query by needs and recommending other services by using semantic search technologies.

1.3 Master thesis goals

The main purpose of this master thesis is to have a **repository** of widgets and services to build new mash-ups. First we need to **feed the repository** and we have to do it automatically using discovery techniques. The content fetched from the internet must be **structured** before it is inserted into the repository and therefore the repository has to be able to store the data with the same structure, this is done using **semantic repositories**.

All the information stored in the repository has to be **managed** manually by an administrator. This is the main goal of this master thesis, create an administration interface that permits an administrator user chose which widgets and services automatically fetched and stored are useful. The administrator interface will provide several tools to the administrator user and will use **ranking** algorithms to help him decide which ones are usefull.

Finally, there is a web interface for final users that will allow them find services and services using semantic search technologies.

1.4 Structure of this Master Thesis

In this section we will provide a brief overview of all the chapters of this Master Thesis. It has been structured as follows:

Chapter 1 provides an introduction to the problem which will be approached in this project. It provides an overview of the benefits of mash-ups and linked data technologies. Furthermore, a deeper description of the project and its environment is also given.

Chapter 2 contains an overview of the existing technologies on which the development of the project will rely.

Chapter 3 describes one of the most important stages in software development: the requirement analysis using different scenarios. For this, a detailed analysis of the possible use cases is made using the Unified Modeling Language (UML). This language allows us to specify, build and document a system using graphic language. The result of this evaluation will be a complete specification of the requirements, which will be matched by each module in the design stage. This helps us also to focus on key aspects and take apart other less important functionalities that could be implemented in future works.

Chapter 4 describes the architecture of the system, dividing it into 3 groups and differentiating front-end and back-end modules.

Chapter 5 describes a selected use case. It is going to be explained the running of all the tools involved and its purpose. It is based on how to crawl the web to find new mash-ups, then feed the repository, do the validation and rejections of the mash-ups, and finally the developer will be able to use the discovered services.

Chapter 6 sums up the findings and conclusions found throughout the document and gives a hint about future development to continue the work done for this master thesis.

Finally, the appendix provide useful related information, especially covering the installation and configuration of the tools used in this thesis.

Enabling Technologies

This chapter introduces which technologies have made possible this project. First of all there must be a place (repository) to store all the mash-ups, this is achieved by the OMELETTE mash-up register, explained in section 2.2. Second, the technology that has made possible feeding the repository in section 2.3. Finally, the technologies that have been used to develop the web interfaces that enable browsing the mash-ups.

2.1 Overview

In the current Web, developers enjoy the availability of plenty of services and widgets that can be reused to build new web applications. This ecosystem of reusable web components comprises elements such as data feeds of various domains, telco services or desktop and mobile widgets. Additionally, there is a growing set of tools for the creation of mash-ups such as MyCocktail [2] or mashArt [3] that facilitate developers in the combination of services into new applications. Also, Programmable Web, Yahoo Pipes or Opera widgets are examples of registries that reference services and widgets of many different kinds. Users can query them in order to search useful applications and services that they can reuse for mash-up composition.

However, developers face some difficulties when working on development of mash-ups.

First, it is not easy for a developer to find the most appropriate services for a mash-up she is building, as although many of them are available but the information might be scattered across various repositories in the web in different formats on multiple levels of granularity.

Second, services are annotated using different description standards and semantics, thus requiring deep study of the documentation by the developer.

Third, due to this lack of consistent standardized descriptions, services need to be adapted in order to be used in a mash-up platform.

This master thesis describes the creation of a searchable repository populated with services and widgets that will serve in which a developer user will be able to find those components that he needs to create new mash-ups.

The main goal of this project is to create an interface that permits the developer fulfil his requisites finding the suitable services or widgets in the repository. To create this interface first there must be a repository and this repository must have widgets and services. The structure and the functionalities that this repository must have are described in section 2.2 and we call it the OMELETTE mash-up Registry (OMR).

The repository is fed automatically using automated discovery techniques. These techniques are described in section 2.3 and exposed those existing websites on the Internet where the automatic algorithms fetch the data from.

2.2 OMELETTE mash-up Registry

The Omelette mash-up Registry (OMR) has a component model that aggregates necessary information for querying web components and searching the most appropriate ones. Additionally, this model reuses other underlying standards, such as WSMO, WSDL, WADL or W3C widgets, as low-level grounding standard description languages that allow components to be readily executable by referencing to them whenever available. These descriptions are built automatically, when possible, in a discovery phase that allows populating the registry with new reusable software artefacts from the Web.

Components stored in the OMR use the Linked mash-ups Ontology (LiMOn) RDF model [4].

2.2.1 RDF model

The objective of the OMELETTE mash-up Registry is to provide an integrated centralized reference of web components to facilitate querying and selection of relevant ones when building new mash-ups. To achieve this, an RDF model based format is employed to describe the components. It is defined in this section along with the interface that supports querying and selection of components from the registry.

The registry integrates heterogeneous components that can be potentially used in various web applications. More specifically, mash-up applications and services from the Web are the ones under consideration. mash-up is treated as a first-class object that is comprised of any web applications. Examples of mash-ups and services can be found in repositories such as Yahoo Pipes or Programmable Web.

In order to make these components available for developers, the registry stores relevant metadata that can be used by the developers for selecting components. Additionally, these metadata should be available in the web in order to make it possible to automate the population of the registry with real components. Usually, web component repositories contain metadata such as a component's name, textual description, tags or categorization. Other specific properties that depend on the nature of the component can also be found, such as inputs, endpoints, web service dependencies, or underlying formal descriptions like WSMO or WSDL.

With these considerations in mind, the OMELETTE team has defined the model presented in Figure 2.1: LiMOn (Linked mash-up Ontology). It is a model that integrates the properties and fields that are provided by current component repositories in the web.

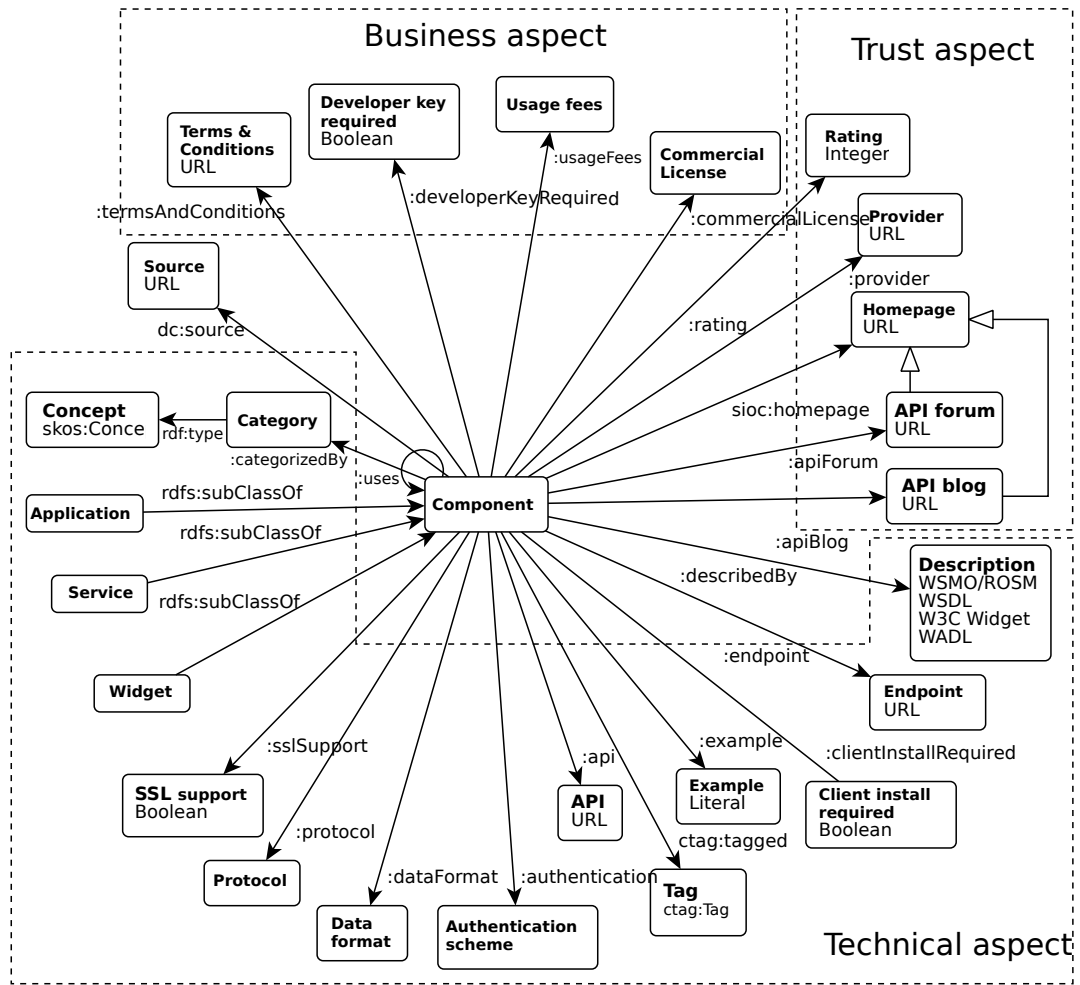


Figure 2.1: Omelette mash-up Registry RDF model

Its name comes for its approach of bringing Linked Data to mash-up-Driven Development. It allows describing mash-ups and their components for integrating and sharing mash-up information such as categorization or dependencies.

This model covers aspects such as general categorization metadata, licensing or usage, and basic aspects of component execution. It reuses Simple Knowledge Organization System (SKOS¹), Friend of a Friend (FOAF²), Dublin Core (DC³) and Common Tag (CTag⁴) ontologies in order to follow the guiding principle of Semantic Web, which manifest reusability as one of the main postulates.

The OMELETTE schema also makes use of work done in SOA4All [5] FP7 project

¹<http://www.w3.org/2009/08/skos-reference/skos.html>

²<http://xmlns.com/foaf/spec/>

³<http://dublincore.org/>

⁴<http://commontag.org/Home>

on RESTful services with ROSM/WSMO service descriptions. Every component might reference an additional description, such as WSDL, WSMO or W3C widget. As shown in the discovery section, ROSM will be used as the basis for describing REST services.

A registry with component descriptions according to the presented model can be queried using a SPARQL query as the one below:

Listing 2.1: Example SPARQL

```
PREFIX om: <http://www.ict-omelette.eu/schema.rdf#>
PREFIX ctag: <http://commontag.org/ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?service
WHERE
{
  ?service rdf:type om:Service;
  om:categorizedBy om:Telco;
  ctag:tagged [ rdfs:label "video" ];
  ctag:tagged [ rdfs:label "conference" ];
  om:developerKeyRequired "false".
}
```

This query retrieves telco services with video conferencing functionality. The next SPARQL query asks the registry for services able to search a picture by keywords. It also retrieves the actual endpoint or URL that needs to be accessed to run the service:

Listing 2.2: Example SPARQL keywords

```
PREFIX om: <http://www.ict-omelette.eu/schema.rdf#>
PREFIX ctag: <http://commontag.org/ns#>
PREFIX rosm: <http://www.wsmo.org/ns/rosm/0.1#>
PREFIX hrests: <http://www.wsmo.org/ns/hrests#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?service ?endpoint
WHERE
{
  ?service rdf:type om:Service;
  ctag:tagged
  [ rdfs:label "photos" ];
  om:describedBy [ rdf:type
  rosm:Service;
  rosm:requestURIParemeter [ ctag:tagged [ rdfs:label "keywords" ] ];
  hrests:hasAddress
  ?endpoint ].
}
```

2.3 Automated Discovery

Automated discovery is one of the tasks present in the scope of this project. Its objective is to enable OMELETTE users to access a wide amount of web components (i.e. both widgets and services) inside the OMR. Thanks to the automated discovery capabilities OMELETTE users will be able to use services and widgets as soon as they are published on external repositories.

2.3.1 Introduction

Three repositories were mined for services, widgets and mash-ups at this stage of the project, namely Programmable Web⁵, Opera Widgets⁶ and Yahoo Pipes⁷.

ProgrammableWeb

Programmable Web, shown in 2.2, is the most popular registry of APIs and mash-ups on the Web, and allows developers to include their APIs or mash-ups for other developers. It currently contains more than 3,000 APIs and more than 5,000 mash-ups. Information about which APIs are used by mash-ups, licensing issues, or categorization information can be found in Programmable Web too.

Yahoo Pipes

Yahoo Pipes, shown in 2.3, is a mash-up environment developed by Yahoo, where developers can build data feeds that make use of other feeds by visually dragging and dropping operators and sources. The resulting so-called "pipes" can be run as any other feed, also accepting input parameters and providing a standardized RSS output. The pipes, or mash-ups, are categorized by tags, data format, sources, and also include short textual descriptions.

Opera Widgets

Opera Widgets, shown in 2.4, is a repository of mainly W3C widgets that are shared among the community of users of Opera Web Browser. These widgets can be used in OMELETTE because they follow W3C Widget standard. The repository provides a categorized collection of widgets, along with short textual descriptions of the widget's functionality.

⁵<http://www.programmableweb.com/>

⁶<http://widgets.opera.com/>

⁷<http://pipes.yahoo.com/>

The screenshot shows the ProgrammableWeb website homepage. At the top left is the 'programmableweb' logo. To its right is a banner for 'Rovi Cloud Services APIs' with a list of services: Advertising Service, Search Service, Recommendations Service, Video Analytics Service, and Data Service. Below this is a navigation bar with links for 'Hot APIs', 'Twitter', 'YouTube', 'Facebook', 'Google Maps', 'Flickr', 'LinkedIn', and 'More'. The main navigation bar includes 'Home', 'API News', 'API Directory', 'Mashups', 'Community', 'How-to', and 'Subscribe'. A search bar is present with the text 'Find APIs, mashups, code and developers'. Below the search bar, it says 'Popular searches: photo google flash mapping enterprise sms'. The main content area is divided into three columns. The left column is titled 'New APIs' and lists 'PhotoXpress', 'PerfectForms', 'Hyves Data', 'Sabre', 'Amadeus', and 'Milo'. The middle column is titled 'Mashup of the Day' and features an image of sandals. The right column is titled 'New Mashups' and lists 'Zappos Peer Pressure', 'Instant Camera', 'Watch.io', 'tweedly', 'Cheap-Flights.To', and 'Let's Make a Puzzzle'. On the right side of the page, there are several advertisements: 'MASHERY The Premier API Management Solution', 'HOOVER'S IDEAS & APPS CONTEST', 'Payment Processing Simplified e-Similate', and 'brewmp. How the other half apps.'.

programmableweb

Rovi Cloud Services APIs
Powerful web services that let you build clever entertainment apps and sites

explore now
... and try our APIs for free

rovi

Hot APIs » Twitter YouTube Facebook Google Maps Flickr LinkedIn More »

Latest news Long a Data Provider, Financial Research Firm Becomes AP... »

Home API News API Directory Mashups Community How-to Subscribe: RSS Email Twitter Facebook

Keeping you up to date with APIs, mashups and the Web as platform. Learn more »

Find APIs, mashups, code and developers Search

Popular searches: photo google flash mapping enterprise sms

3497 APIs

5920 Mashups

New APIs

- PhotoXpress
- PerfectForms
- Hyves Data
- Sabre
- Amadeus
- Milo

See more APIs

Mashup of the Day

See previous winners

New Mashups

- Zappos Peer Pressure
- Instant Camera
- Watch.io
- tweedly
- Cheap-Flights.To
- Let's Make a Puzzzle

See more mashups

Long a Data Provider, Financial Research Firm Becomes API Provider

Zacks, a well known and experienced financial research firm, is opening its data up to developers with the new [Zacks Financial API](#). What's interesting is that the [Xignite APIs](#), which we called [maitre d' of financial data](#), lists Zacks as a data source. So, it seems that Zacks has decided to see just how well it can play the API game on its own. And why not? The financial sector is fairly late in adopting APIs, so there is room for both data providers.

Posted July 18, 2011. Continue reading

ZACKS INVESTMENT RESEARCH

MASHERY
The Premier API Management Solution

HOOVER'S IDEAS & APPS CONTEST
Win \$10,000! Enter by August 25th

Payment Processing Simplified
e-Similate view APIs now

Pat McKenzie is a DOer

brewmp.
How the other half apps.

He is a one man software factory...

LEARN MORE

Figure 2.2: ProgrammableWeb

The screenshot shows the Yahoo Pipes website. At the top, there's a navigation bar with links: Home, My Pipes, Browse, Discuss, Documentation, and a prominent 'Create a pipe' button. A search bar is also present with the text 'Search for Pipes...'. Below the navigation bar, a grey banner states: 'In the first week of August, all Pipes will be upgraded to the V2 engine. Report V2 engine issues here ...'.

The main content area is divided into two columns. The left column is titled 'Current Search:' and contains a search bar. Below it, a section 'Refine your search' lists various filters with counts:

- Formats (58)** (more): csv (1542), georss (6914), media (15709), media-application (797), media-audio (590), media-image (3554), media-video (426), mediaapplication (838), mediaaudio (420), mediaimage (3526).
- Tags (99)** (more): blog (825), feed (511), flickr (634), google (539), jobs (650), music (550), news (2241), rss (974), search (638), twitter (792).
- Sources (99)** (more): api.flickr.com (9343), blogspot.com (7867), feedburner.com (16764).

The right column is titled 'Browse Pipes' and lists several pipes with their titles, authors, descriptions, and clone counts:

- Title Mangler** ★ by Ramblurr: Lets you add text before and after the title in a feed. Published on 04/04/08 | 442 clones.
- Add Feed Label to Each Item Title** ★ by Randy (rockman): If you are aggregating multiple feeds, sometimes its nice to be able to tell which source an item is coming from. I like to add a tag or description of the blog or feed name to each item's title. This pipe is useful when inserted into another pipe and you can hard code the feed URL and the label... Published on 10/30/07 | 1499 clones.
- Title Mangler - improved** ★ by Christopher Peter: I just took out some mistakes and exchanged Prefix/Postfix to make it more handy. Lets you add text before and after the title in a feed. Use for blanks between Prefix or Postfix and URL. Tags: feed feeds prefix editing improve +3... Published on 05/29/09 | 192 clones.
- Update Maker for Lifestreams** ★ by Kingsley: If you would like to post lifestream updates like "Kingsley listened to 'Like You Used To' by J.J. Cale on Last.fm", then this is the pipe for you! Give it a feed url. Then give it a prefix (like "Kingsley listened to ") - it will be added in front of the title of each... Sources: twitter.com Published on 04/28/08 | 253 clones.
- YouTube tags to RSS** ★ by Eric: A building block for the more advanced YouTube filter Pipe. Tags: rss youtube subpipe Sources: youtube.com Published on 05/11/07 | 1242 clones.
- RSS 2 Geo** ★ by Premasagar: Uses the geonames webservice to add location information to a standard RSS feed, creating GeoRSS. The result can be displayed on a map. Modified from: http://pipes.yahoo.com/pipes/pipe.info?id=gGThvN_62xG2JH50ZoQMOQ

Figure 2.3: Yahoo Pipes

The image shows the ProgrammableWeb website interface. At the top, there's a navigation bar with links like Home, API News, API Directory, Mashups, Community, and How-to. Below this, there's a search bar and a section for "New APIs" listing items like PhotoXpress, PerfectForms, Hyves Data, Sabre, Amadeus, and Milo. A "Mashup of the Day" section features a pair of sandals. To the right, there's a "New Mashups" section listing items like Zappos Peer Pressure, Instant Camera, Watch.io, tweedly, Cheap-Flights.To, and Let's Make a Puzzzzle. The bottom section features a news article titled "Long a Data Provider, Financial Research Firm Becomes API Provider" with a sub-headline about Zacks. The article text mentions Zacks Financial API and Xignite APIs. There are also several promotional banners for Rovi Cloud Services APIs, Mashery, Hoover's IDEAS & APPS CONTEST, e-Similate, Pat McKenzie, and brewmp.

programmableweb

Rovi Cloud Services APIs
Powerful web services that let you build clever entertainment apps and sites

explore now
... and try our APIs for free

rovi

Hot APIs » Twitter YouTube Facebook Google Maps Flickr LinkedIn More » **Latest news** Long a Data Provider, Financial Research Firm Becomes AP... »

Home API News API Directory Mashups Community How-to Subscribe: RSS Email Twitter Facebook

Keeping you up to date with APIs, mashups and the Web as platform. **Learn more »**

Find APIs, mashups, code and developers **Search**

Popular searches: [photo](#) [google](#) [flash](#) [mapping](#) [enterprise](#) [sms](#)

3497 APIs **5920** Mashups

New APIs

- PhotoXpress
- PerfectForms
- Hyves Data
- Sabre
- Amadeus
- Milo

[See more APIs](#)

Mashup of the Day

[See previous winners](#)

New Mashups

- Zappos Peer Pressure
- Instant Camera
- Watch.io
- tweedly
- Cheap-Flights.To
- Let's Make a Puzzzzle

[See more mashups](#)

RSS Latest News: July 18, 2011 [Read more news](#)

Long a Data Provider, Financial Research Firm Becomes API Provider

[Zacks](#), a well known and experienced financial research firm, is opening its data up to developers with the new [Zacks Financial API](#). What's™ interesting is that the [Xignite APIs](#), which we called [maitre d' of financial data](#), lists Zacks as a data source. So, it seems that Zacks has decided to see just how well it can play the API game on its own. And why not? The financial sector is fairly late in adopting APIs, so there is room for both data providers.

Posted July 18, 2011. [Continue reading](#)

ZACKS INVESTMENT RESEARCH

MASHERY
The Premier API Management Solution

HOOVER'S IDEAS & APPS CONTEST
Win \$10,000! Enter by August 25th

Payment Processing Simplified
e-Similate™ [view APIs now](#)

Pat McKenzie is a DOer

brewmp.
How the other half apps.

LEARN MORE

He is a one man software factory...

Figure 2.4: Opera Widgets

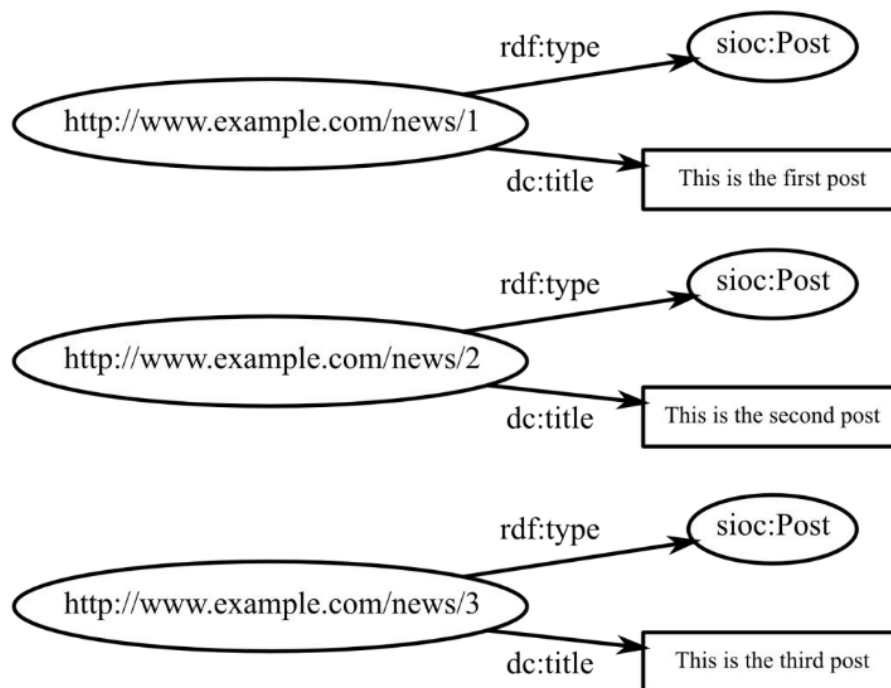


Figure 2.6: Mapping example for data extraction

crawl the sites and build an RDF knowledge base that is dumped into the OMR.

In the case of Programmable Web, each web resource either represents a mash-up or an API. For each of them, the fields shown are mapped into an element of the ontology, covering the components' metadata such as categorization or tagging.

Similarly, for Opera Widgets each web resource represents a widget, so the information about the widget is mapped to the terms from the ontology. Also, its widget package (which uses WGT extension⁸) is mapped as well as the widget's endpoint.

In Yahoo Pipes, more advanced scraping has been performed, thanks to an implicit service description that is available as an HTML form. For each pipe, a form for its execution is available in the mash-up's webpage, as shown in Figure 2.7.

These HTML forms are mapped to a Resource-Oriented Service Model (ROSM) or a Web Service Modeling Ontology (WSMO) description in order to get detailed information of the service's interface.

⁸<http://www.w3.org/TR/2011/REC-widgets-20110927/>

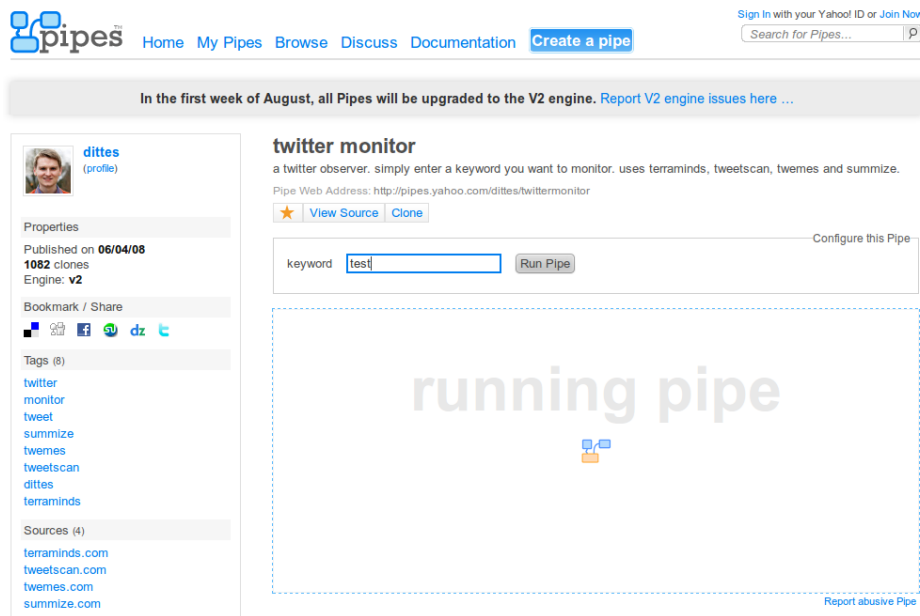


Figure 2.7: Execution page of a Yahoo Pipe's mash-up

An example of ROSM description, extracted from the pipe shown in the Figure 2.5, which accepts a set of textual keywords on a URL, is shown next:

Listing 2.3: Example ROSM definition

```
<rosm:Service>
  <rosm:supportsOperation>
    <rosm:Operation>
      <hrests:hasAddress
        rdf:resource="http://pipes.yahoo.com/pipes/pipe.info?_id=c32fa09"/>
      <rosm:requestURIParameeter>
        <rdf:Description>
          <ctag:tagged>
            <rdf:Description>
              <rdfs:label>text</rdfs:label>
            </rdf:Description>
          </ctag:tagged>
          <rdfs:label>keyword</rdfs:label>
        </rdf:Description>
      </rosm:requestURIParameeter>
    </rosm:Operation>
  </rosm:supportsOperation>
</rosm:Service>
```

2.4 Conclusions

In this chapter we have introduces some of the technologies wich are part of the OMELETTE project and conform the base for this master thesis.

It is necessary to understand first what a repository is for and how the automated discovery is done. We have also introduced the websites scrapped and this helps to understand what kind of services and widgets populate the repository.

Requirement Analysis

This chapter describes one of the most important stages in software development: the requirement analysis using different scenarios. For this, a detailed analysis of the possible use cases is made using the Unified Modeling Language (UML). This language allows us to specify, build and document a system using graphic language.

3.1 Overview

The result of this chapter will be a complete specification of the requirements, which will be matched by each module in the design stage. This helps us also to focus on key aspects and take apart other less important functionalities that could be implemented in future works.

3.2 Use cases

These sections identify the use cases of the system. This helps us to obtain a complete specification of the uses of the system, and therefore define the complete list of requisites to match. First, we will present a list of the actors in the system and a UML diagram representing all the actors participating in the different use cases. This representation allows, apart from specifying the actors that interact in the system, the relationships between them.

These use cases will be described the next sections, including each one a table with their complete specification. Using these tables, we will be able to define the requirements to be established.

3.2.1 Actors dictionary

The list of primary and secondary actors is presented in table 3.1. These actors participate in the different use cases, which are presented later.

Actor identifier	Role	Description
ACT-1	User	End user that uses Omelette mash-up Editor to find a mash-up based on her goals, which are expressed using keywords
ACT-2	Developer	Technical developer which uses the OMR.
ACT-4	Admin	Administrator of the OMR, in charge of tasks such as inserting, deleting mash-ups, as well as including new available mash-up repositories..
ACT-5	MDP	mash-up Delivery Platform, component of the "Live Omelette Environment" that executes mash-ups..
ACT-6	External service	External services for executing actions.

Table 3.1: Actors list

3.2.2 OMR composition and search use case

This use case package collects the search functionalities of OMR, as shown in 3.1.

The use cases presented in this section are as shown in the Figure 3.1:

- *keyword search* detailed in sub-section 3.2.2.1.
- *mashup browse by category* detailed in sub-section 3.2.2.2.
- *ask question* detailed in sub-section 3.2.2.5.
- *mashup search by query* detailed in sub-section 3.2.2.3.
- *mashup compose* detailed in sub-section 3.2.2.4.

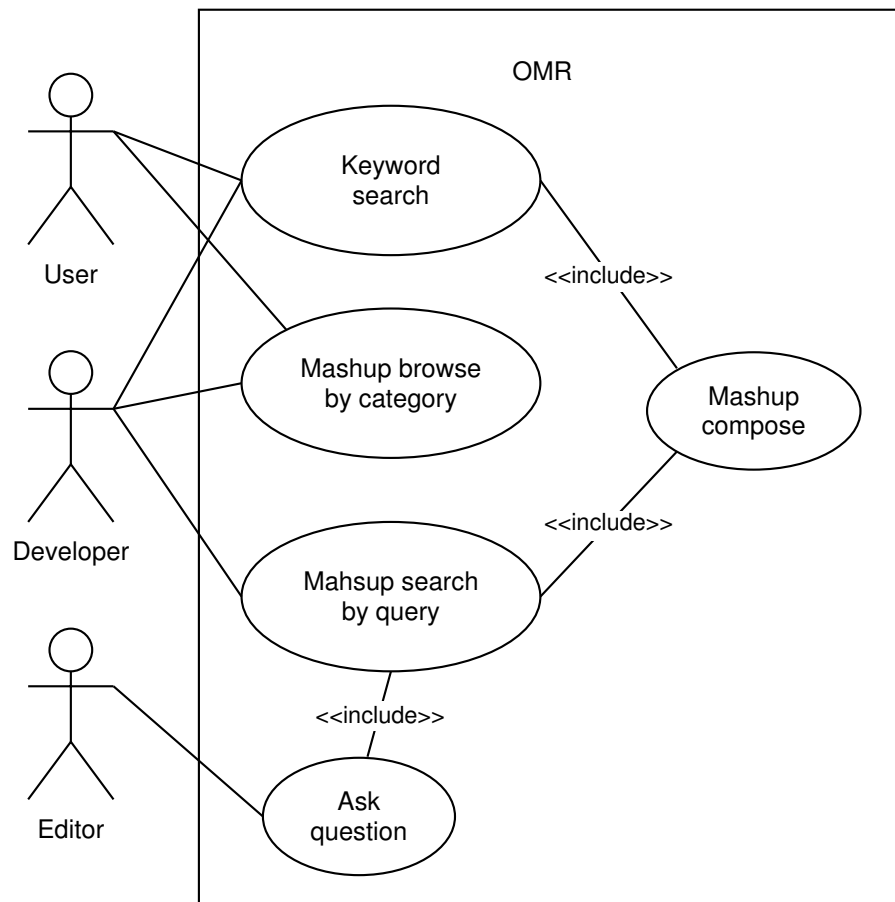


Figure 3.1: Composition and Search use case

3.2.2.1 Keyword search

Use Case Name	keyword search		
Use Case ID	UC1.1		
Pre-Condition	OMR has been fed with mash-ups and individual services		
Post-Condition	Optionally, the developer completes the MDL definition to make a mash-up executable		
Flow of Events		Actor Input	System Response
	1	The user / developer expresses her goals by using textual keywords	Ranked result list of mash-ups and individual services. The mash-up can be an existing mash-up or a new dynamically composed mash-up, which can be executable or not. In the same way, the service can be automatically available in the editor or could require human intervention.
	2a	The user / developer access service / mash-up metadata and selects one service / mash-up in order to use it	System shows details of the metadata service / mash-up.
	2b	The developer has obtained a non executable mash-up and opens the editor to complete it	The system opens the mash-up editor, which allows the user to finish the mash-up composition (giving API details, etc.), completing the MDL, and finally, publishing the resulting mash-up in the OMR.

3.2.2.2 Mash-up browse by category

Use Case Name	mash-up browse by category		
Use Case ID	UC1.2		
Primary Actor	User, Developer		
Pre-Condition	OMR has been fed with mash-ups and services and has categorized the service / mash-ups according to one or more categories. For example, users can follow a taxonomy based on their needs and types of applications, such as Appstore, while developers can follow a taxonomy based on the technology, integration needs, etc.		
Post-Condition	-		
Flow of Events		Actor Input	System Response
	1	The user / Developer selects a category	Ranked list of service / mash-ups belonging to that category
	2	The user / developer refines the search with filtering options	Filtered result set based on filtering options

3.2.2.3 mash-up search by query

Use Case Name	mash-up search by query		
Use Case ID	UC1.3		
Primary Actor	Developer		
Pre-Condition	OMR has been fed with service / mash-ups		
Post-Condition	-		
Flow of Events		Actor Input	System Response
	1	The developer executes a query using a query language (for example, SPARQL)	Result set of matching service / mash-ups

3.2.2.4 mash-up compose

Use Case Name	mash-up compose		
Use Case ID	UC1.4		
Primary Actor	User, developer		
Pre-Condition	OMR has been fed with individual services and mash-ups		
Post-Condition	Optionally, the developer completes the MDL definition to make a mash-up executable		
Flow of Events		Actor Input	System Response
	1	The User / Developer expresses her goals using textual keywords	Since there is no service mash-up that fulfills the user goals, the system analyses potential combinations of available mash-ups in OMR and provides a composed mash-up or a template of a potential composition

3.2.2.5 Ask suggestion

Use Case Name	Ask suggestion		
Use Case ID	UC1.5		
Primary Actor	Editor		
Pre-Condition	OMR has been fed with individual services and mash-ups		
Post-Condition			
Flow of Events		Actor Input	System Response
	1	The mash-up editor requests a list of available service / mash-ups to suggest them to the user	Ranked result list of service / mash-ups expressed in MDL. The mash-up can be an existing mash-up or a new dynamically composed mash-up, which can be executable or not

3.2.3 OMR discovery and administration use case

This use case package collects the main administration use cases of the OMR, as shown in 3.2

- *automated mash-up feeding* detailed in subsection 3.2.3.1
- *manual mash-up feeding* detailed in subsection 3.2.3.6
- *form discovery and description* detailed in subsection 3.2.3.2
- *mash-up registry integration* detailed in subsection 3.2.3.3
- *mash-up registry API based integration* detailed in subsection 3.2.3.4
- *mash-up registry Scrappy based integration* detailed in subsection 3.2.3.5
- *manual mash-up management* detailed in subsection 3.2.3.6
- *browse pending* detailed in subsection 3.2.3.7
- *validate service* detailed in subsection 3.2.3.8
- *reject service* detailed in subsection 3.2.3.9

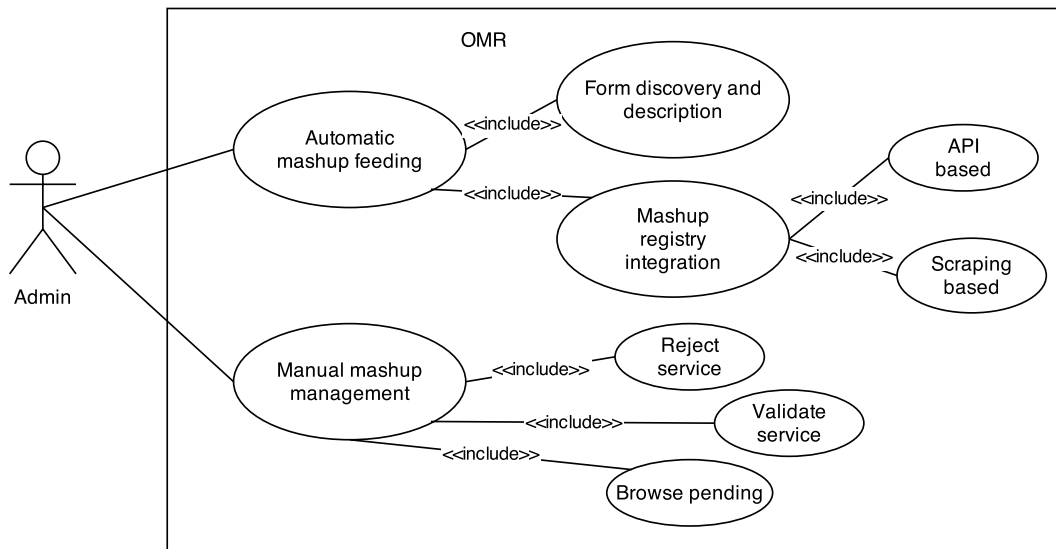


Figure 3.2: OMR discovery and administration use case

3.2.3.1 Automatic mash-up feeding

Use Case Name	Automatic mash-up feeding		
Use Case ID	UC2.1		
Primary Actor	Admin		
Pre-Condition	Availability of external service mash-up repositories or web sites		
Post-Condition	Service / mash-ups are added to OMR		
Flow of Events		Actor Input	System Response
	1	The admin selects a mash-up source	The system connects to the service mash-up source and obtains potential mash-ups, their metadata, and catalogues them
	2		he system generates automatically the MDL (partial MDL or executable MDL).
	3a	(Optional) the admin reviews results and approves them for its inclusion in the registry.	
	3b	The discovered mash-ups are automatically integrated in the registry.	

3.2.3.2 HTML Form discovery and description

Use Case Name	Automatic mash-up feeding		
Use Case ID	UC2.2		
Primary Actor	Admin		
Pre-Condition	Availability of external web sites offering services		
Post-Condition	Service / mash-ups are added to OMR		
Flow of Events		Actor Input	System Response
	1	The admin selects a interesting domain and a description schema for that domain (mash-up metadata). Optionally, the admin selects a set of potential interesting web domain as well as a black list	The system crawls the web looking for web sites which match the domain. The system identifies interesting HTML forms, characterizes these HTML forms as well as their results. Then, it classifies the identified mash-up, generates its MDL and adds it to the OMR.
	2a	(Optional) the admin reviews the results and approves them for its inclusion in the registry.	
	2b		The discovered service / mash-ups are automatically integrated in the registry.

3.2.3.3 mash-up registry integration

Use Case Name	mash-up registry integration		
Use Case ID	UC2.3		
Primary Actor	Admin		
Pre-Condition	Availability of external mash-up repositories		
Post-Condition	mash-ups are added to OMR		
Flow of Events		Actor Input	System Response
	1	The admin selects a mash-up source as well as the polling conditions.	The system connects to the mash-up source and obtains mash-up descriptions and catalogues them.
	2a	(Optional) the admin reviews the results and approves them for its inclusion in the registry.	
	2b		The discovered service / mash-ups are automatically integrated in the registry.

3.2.3.4 API based integration

Use Case Name	API based integration		
Use Case ID	UC2.4		
Primary Actor	Admin		
Pre-Condition	Availability of external mash-up registries offering an API		
Post-Condition	mash-ups are added to OMR		
Flow of Events		Actor Input	System Response
	1	The admin selects a mash-up source and the polling options.	The system connects to mash-up source and obtains potential mash-ups, their metadata, and catalogues them
	2a	(Optional) the admin reviews the results and approves them for its inclusion in the registry.	
	2b		The discovered service / mash-ups are automatically integrated in the registry.

3.2.3.5 Scraping based integration

Use Case Name	Scraping based integration		
Use Case ID	UC2.5		
Primary Actor	Admin		
Pre-Condition	Availability of external mash-up registries not offering an available API or a restrictive API		
Post-Condition	mash-ups / Services are added to OMR		
Flow of Events		Actor Input	System Response
	1	The admin selects a mash-up source and defines a description schema	The system scrapes mash-up registry, obtaining potential mash-ups / services, and catalogues them
	2a	(Optional) the admin reviews the results and approves them for its inclusion in the registry.	
	2b		The discovered service / mash-ups are automatically integrated in the registry.

3.2.3.6 Manual mash-up management

Use Case Name	Manual mash-up management		
Use Case ID	UC2.6		
Primary Actor	Admin		
Pre-Condition	-		
Post-Condition	A mash-up is added, deleted to OMR		
Flow of Events		Actor Input	System Response
	1a	The admin selects a mash-up / service and execute one CRUD operation	he system executes the operation and returns the result.
	1b	(Optional) the system uses filtering options to find the mash-up (by source, based on properties, etc.)	

3.2.3.7 Browse pending mash-ups and services

Use Case Name	Browse pending mash-ups and services		
Use Case ID	UC2.7		
Primary Actor	Admin		
Pre-Condition	OMR has been fed with service / mash-ups		
Post-Condition	-		
Flow of Events		Actor Input	System Response
	1	The admin filters using facets	The system executes the corresponding queries and shows the matching mash-ups and services
	2a	(Optional) The admin can request at any moment statistics about the filtering	The system shows the requested statistics using charts
	2b	(The admin requests information about a mash-up / service	The system gets the requested information

3.2.3.8 Validate a mash-up / service

Use Case Name	Validate a mash-up / service		
Use Case ID	UC2.8		
Primary Actor	Admin		
Pre-Condition	OMR has been fed with service / mash-ups		
Post-Condition	Services / mash-ups validated into OMR		
Flow of Events		Actor Input	System Response
	1	The admin filters using facets	The system executes the corresponding queries and shows the matching mash-ups and services
	2a	The admin checks the services / mash-ups wanted to be validated	The system validates the services / mash-ups into OMR

3.2.3.9 Reject a mash-up / service

Use Case Name	Reject a mash-up / service		
Use Case ID	UC2.9		
Primary Actor	Admin		
Pre-Condition	OMR has been fed with service / mash-ups		
Post-Condition	Services / mash-ups rejected into OMR		
Flow of Events		Actor Input	System Response
	1	The admin filters using facets	The system executes the corresponding queries and shows the matching mash-ups and services
	2a	The admin checks the services / mash-ups wanted to be rejected	The system rejects the services / mash-ups into OMR

3.2.4 Web interface use case

This section contains a web interface use case. The client web interface interacts with the OMR in order to obtain consult the registry, which could involve service composition. This use case includes some use cases described previously, as shown in 3.3.

- *request available mash-ups* detailed in 3.2.4.1
- *search mash-ups by query* in 3.2.4.2
- *compose* is not detailed as it is part from the OMELETTE project but not part of this master thesis.

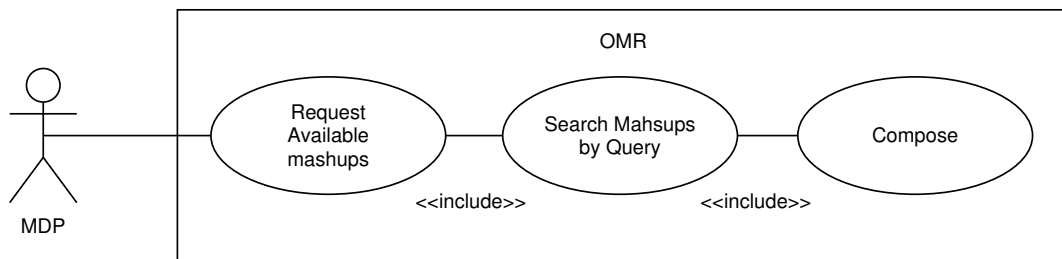


Figure 3.3: Web interface use case

3.2.4.1 Request available mash-ups

Use Case Name	Request available mash-ups		
Use Case ID	UC3.1		
Primary Actor	Web interface		
Pre-Condition	Web interface requests dynamic mash-up binding		
Post-Condition	Web Interface receives a mash-up list		
Flow of Events		Actor Input	System Response
	1a	Web Interface requests a service / mash-up.	The system provides a feed with the ranking of available mash-ups / services.
	1b	(Optional) The system does not find a matching service / mash-up and composes a new one on the fly and returns it	

3.2.4.2 Search mash-ups by query

Use Case Name	Search mash-ups by query		
Use Case ID	UC3.2		
Primary Actor	Web interface		
Pre-Condition	Web interface requests dynamic mash-up binding		
Post-Condition	Web Interface receives a mash-up list		
Flow of Events		Actor Input	System Response
	1a	Web Interface requests a service / mash-up.	The system provides a feed with the ranking of available mash-ups / services.
	1b	(Optional) The system does not find a matching service / mash-up and composes a new one on the fly and returns it	

3.2.5 Conclusions

With the use cases described we have introduced the basic functionalities that have been implemented in this project. They help us to understand the different actors that can interact. They can serve as a base for further development and different use cases that can come to mind. They do not cover mash-up composition, this could be considered for future work lines.

CHAPTER 4

Architecture

This chapter describes in depth how the system is structured in different modules and how the users interact with them and also how the modules interact with other modules by themselves.

4.1 Introduction

The main purpose of this master thesis is to have a **repository** of widgets and services to build new mash-ups. First we need to **feed the repository** and we have to do it automatically using discovery techniques. The content fetched from the internet must be **structured** before it is inserted into the repository and therefore the repository has to be able to store the data with the same structure, this is done using **semantic repositories**.

All the information stored in the repository has to be **managed** manually by an administrator. This is the main goal of this master thesis, create an administration interface that permits an administrator user chose which widgets and services automatically fetched and stored are useful. The administrator interface will provide several tools to the administrator user and will use **ranking** algorithms to help him decide which ones are usefull.

Finally, there is a web interface for final users that will allow them find services and services using semantic search technologies.

We will fill a semantic repository in a automated way as introduced in section 2.3. To achieve this we use Scrappy [6] explained in section 4.2. The former module inserts the semantic data into one of the semantic repositories detailed in section 4.3. This repository needs to be managed by an administrator user through the administrator interface explained in section 4.5. The final developer user will access the semantic repository trough the developer or client interface detailed in 4.6.1.

A diagram of the architecture is shown in Figure 4.1. Each module is detailed in the following sections.

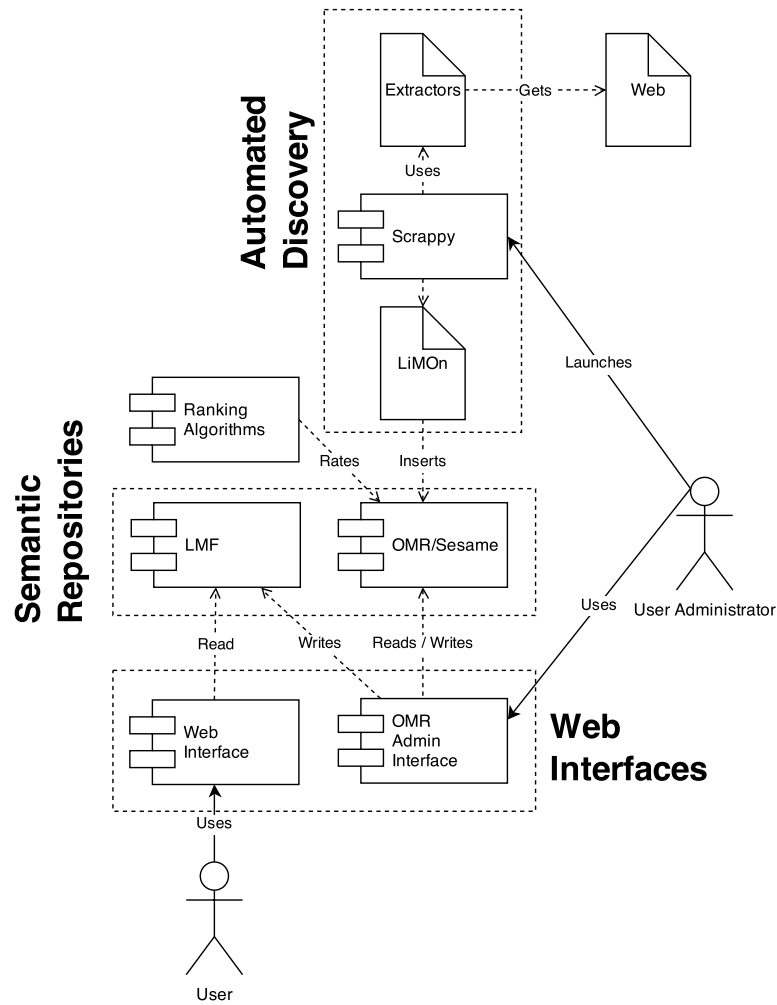


Figure 4.1: General Architecture

4.2 Automated discovery

The automated discovery module's finality is to fetch several services data from different websites. We fetch this data from *ProgrammableWeb*, *Yahoo! Pipes* and *Opera Widgets*. More information about the type of services and widgets this websites contain is described in section 2.3.

To achieve this we will use Scrappy [6], an opensource tool developed by GSI. It can be downloaded from the Github repository of the main contributor of the project¹.

It allows us to extract information from web pages and producing RDF data. Within the OMELETTE project, it has been used for developing the Discovery Module and mining

¹<https://github.com/josei/scrappy>

several repositories of mashups, widgets and services in order to feed the Omelette Mashup Registry with these components.

Scrappy is developed using Ruby and it uses the Webkit library for visual processing of web resources.

Scrappy access a determined website, gets the corresponding HTML, converts the data into RDF using the defined ontology and inserts the result into the OMR (Figure 4.2).

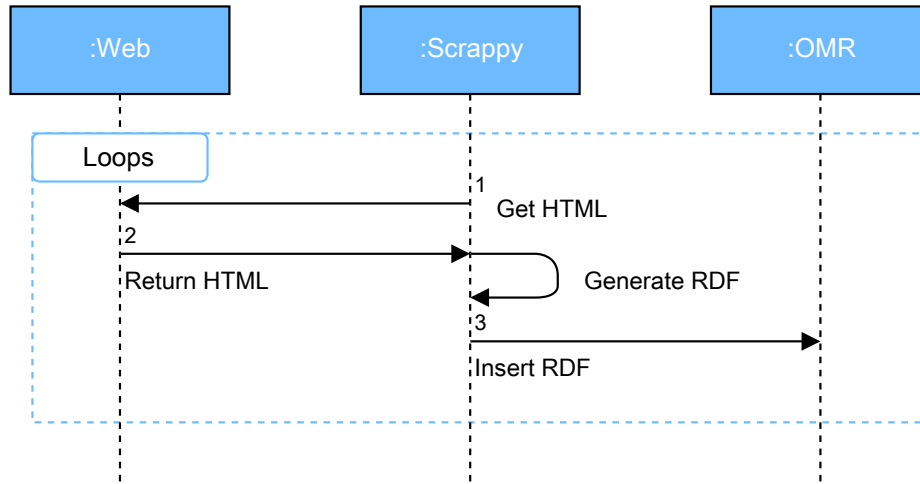


Figure 4.2: Scrappy sequence diagram

In order to fetch the desired information from a website, Scrappy needs to know how is the structure of it. This is done by defining a mapping file written in YARF² which reflects the structure of the html file using CSS selectors.

To create the mapping file we need first to analyze manually the HTML. Significant changes in the HTML of the website to be fetched could break the mapping and re-analyzing would be required.

An example of mapping is shown in Listing A.1, which allows extracting all titles from Yahoo! Pipes.

Scrappy will use the Linked Mashup Ontology (LiMOn) when mapping the resources as seen in 2.1.

The total number of components scrapped from *Yahoo! Pipes* and *Opera Widgets* can be consulted in table 5.4.

²format supported by LightRDF gem, as well as RDFXML, JSON, NTriples formats, which can also be used to define the mappings

Table 4.1: Runnable components in OMR

Number of runnable services	1542
Number of runnable WSDL ³ services	296
Number of runnable REST ⁴ services	1246
Number of runnable widgets	1804
Total number of runnable components	3346

4.3 Semantic repository

All the services and widgets fetched by the automated discovery module described in the former section 4.2 have to be stored somewhere. This place is what we call the *OMELETTE Mashup Registry* (OMR).

The objective of the OMELETTE Mashup Registry is to provide an integrated centralized reference of web components to facilitate querying and selection of relevant ones when building new mashups. To achieve this, an RDF model based format is employed to describe the components (see section 2.2.1).

The registry or repository integrates heterogeneous components that can be potentially used in various web applications. More specifically, mashup applications and services from the Web are the ones under consideration. Mashup is treated as a first-class object that is comprised of any web applications. Examples of mashups and services can be found in repositories such as Yahoo Pipes or Programmable Web.

In order to make these components available for developers, the registry stores relevant metadata that can be used by the developers for selecting components. Additionally, these metadata should be available in the web in order to make it possible to automate the population of the registry with real components. Usually, web component repositories contain metadata such as a component's name, textual description, tags or categorization.

For developing purposes Sesame is used instead of the OMR to help us to solve some problems we were experiencing with different versions of SPARQL. SPARQL is in continuous development and some of the functionalities needed were not implemented in the first 1.0 version but they were in 1.1 version. We chose Sesame between various alternatives as it is

one of the most popular.

Both Sesame and OMR complies the same REST API interface.

Sesame is a de-facto standard framework for processing RDF data. This includes parsing, storing, inferencing and querying of/over such data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions.

Sesame has been designed with flexibility in mind. It can be deployed on top of a variety of storage systems (relational databases, in-memory, file systems, keyword indexers, etc.), and offers a large scale of tools to developers to leverage the power of RDF and related standards. Sesame fully supports the SPARQL query language for expressive querying and offers transparent access to remote RDF repositories using the exact same API as for local access. Finally, Sesame supports all main stream RDF file formats, including RDF/XML, Turtle, N-Triples, TriG and TriX.

4.4 Ranking module

There are thousands of services and widgets fetched and stored in the semantic repository as shown in Table 4.1. The administrator user will have to search for useful components in this repository. By default there is no way to distinguish between relevant or not relevant ones. To solve this, we apply the ranking algorithms described by Tilo Zemke [9].

This module is implemented as a separate component written in Java which runs independently. It connects to the semantic repository and executes the algorithms to calculate the indicators in each component. It is scheduled to run periodically and re-calculate the indexes when new mashups or services are incorporated to the meta-directory.

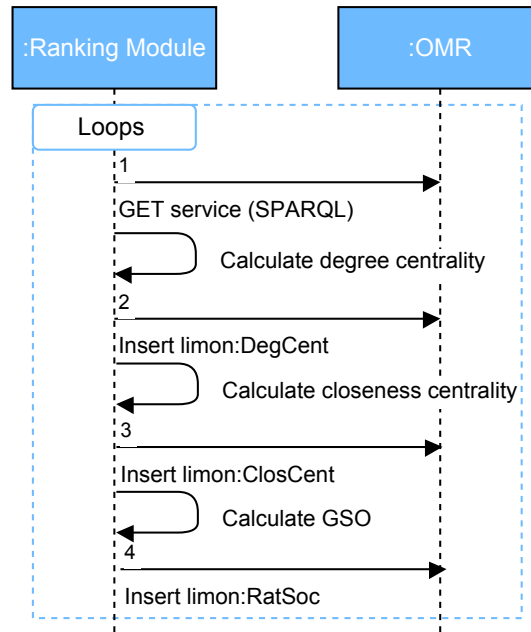


Figure 4.3: Sequence diagram for ranking index generation

4.5 OMR Admin Interface

The automated discovery module fetches thousands of services and widgets and store them in the semantic repository, therefore they are ranked to make the best matches show up when quering the repository. After this an administrator user has to seleect the appropriate one manually. This must be done within a interface that helps that person to do the job, helping him by browsing trough the different categories and properties defined in the RDF model (section subsec:rdfmodel).

On a first instance we decided to build this interface using Simile Exhibit⁵. The description of Exhibit says:

Exhibit lets you easily create web pages with advanced text search and filtering functionalities, with interactive maps, timelines, and other visualizations.

That description seemed exactly what we looked for, but some problems occured, for example high load and poor perfomance or unexpected bugs related to non mature technology. We decided to create our own administrator interface based on the principles of Exhibit with faceted search.

Out OMR administrator interface is built using PHP⁶, HTML, CSS and Javascript.

⁵<http://simile.mit.edu/wiki/Exhibit>

⁶<http://www.php.net>

Recent United States Senate Bills

A continually evolving look at what legislation is passing through Senate committees and thus which states' constituents end up having a preliminary say. Yesterday's new legislation is presented, unless the Senate was in recess. This demo uses the HTML5 Exhibit 3.0 syntax. Powered by [Exhibit 3.0](#).

100 Senators

TABLE • MAP • TILES

name	party	state	member of	sponsored	cosponsored
Daniel K. Inouye	D	HI	Committee on Indian Affairs, Committee on Appropriations, Committee on Commerce, Science, and Transportation, y Committee on Rules and Administration	S.CON.RES.28 y S.RES.256	S.1487 y S.1504
John Hoeven	R	ND	Committee on Indian Affairs, Committee on Agriculture, Nutrition, and Forestry, Committee on	S.RES.253	

Sponsoring
60 (missing this field)
1 S.1467
1 S.1468
1 S.1469
1 S.1470

Co-Sponsoring
54 (missing this field)
2 S.1467
1 S.1468

On Bill Committee
1 (missing this field)
22 S.1467
24 S.1468

Figure 4.4: Simile Exhibit example interface

A general view of the class diagram is shown in 4.5. All the components are executed from the `index.php` file, which we will consider as the main file.

One of the big benefits of Exhibit was the easiness for developers to change the main view. The OMR admin interface is done the same way, using the functions library detailed in subsection 4.5.2 the developer can change the behavior and functionalities of the web interface.

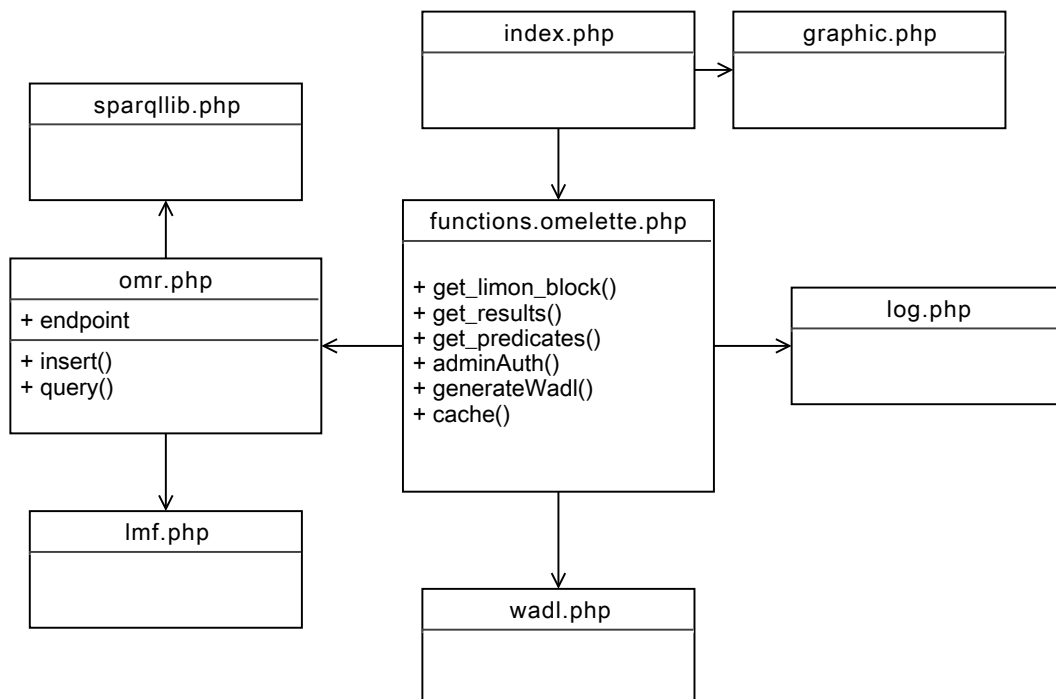


Figure 4.5: OMR Admin Interface architecture

4.5.1 Main component

The main file of the OMR Admin interface is the *index.php*. It is the execution point. The rest of components and functions are called from this execution point.

In the *index.php* the developer can decide how to visualize everything and just adding few lines of code it will generate the facet boxes and the results container.

It is mostly HTML with CSS and Javascript. It also uses the javascript framework jQuery⁷ and svglizer⁸ to render the result of SPARQL SELECT queries into charts.

The functions available that will generate the HTML automatically are described in *functions.omelette.php* and will be detailed in the sub-section 4.5.2.

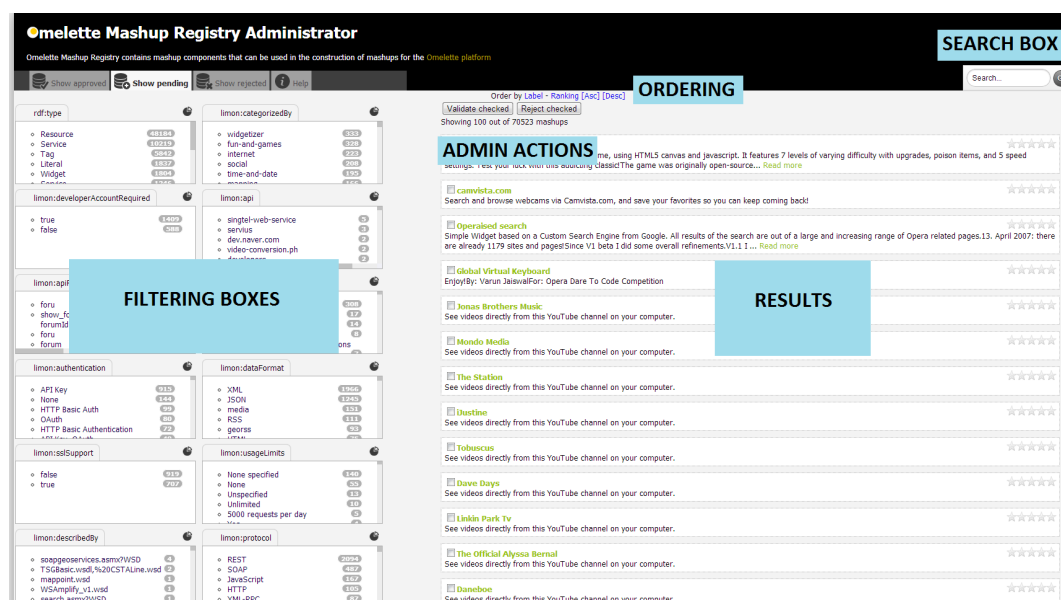


Figure 4.6: OMR Admin Interface

4.5.2 Functions library

The file *functions.omelette.php* is a library of functions used by the rest of the classes. These functions will help to generate the HTML code in the interface.

⁷<http://jquery.com>

⁸<http://sgvizler.googlecode.com/>

4.5.2.1 Facet boxes

This is one of the functions of the library. By creating facet boxes we can do facet searching. We can create one facet box for each of the properties defined in the RDF model (section 2.2.1).

We can create facet boxes using the following function in the main file of our interface. Every time we call the *get_limon_block()* function it will render a facet box.

Listing 4.1: "Creating facet box"

```
<?php get_limon_block("limon:categorizedBy",filter); ?>
```

In the Figure 4.7 we can see the code on the right that will render the facet boxes on the left.

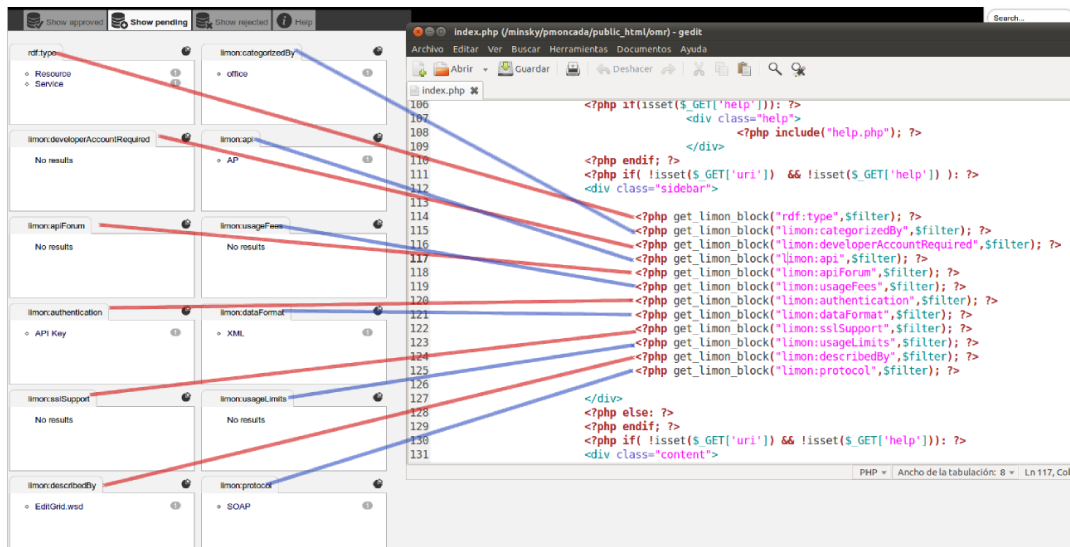


Figure 4.7: Generating Facet Boxes

4.5.2.2 Result lists

The results of the search are shown in the *result list*. We can create a result list container with the following PHP function of the library:

Listing 4.2: "Creating result container"

```
<?php get_results(filter,order,dir); ?>
```

4.5.2.3 Widget or service

If we want to see all the information regarding to a selected *widget or service* we can do it by using the code listed here. This information is showed when clicking on a hyperlink of the result list.

Listing 4.3: "Creating information view"

```
<?php elseif( isset($_GET['uri']) ): ?>
    <?php get_predicates($_GET['uri']); ?>
<?php endif; ?>
```

4.5.2.4 Generate charts

We can generate charts to see in a graphical way the information regarding to the repository as different services shown in different categories.

It uses an external module named *svgizler* to generate charts. The system will generate the proper SPARQL SELECT query and insert it into the module to generate the pie chart.

Listing 4.4: "Generating pie charts"

```
<div id="chart"
    data-svgvizler-endpoint="http://krusti.gsi.dit.upm.es
        :8080/LMF-3.0.0/sparql/select"
    data-svgvizler-query="<?php echo base64_decode($_GET['
        query']); ?>"
```

```

data-sgvizler-chart="gPieChart"
data-sgvizler-endpoint_output="xml"
style="width:800px; height:400px;"></div>
</body>

```

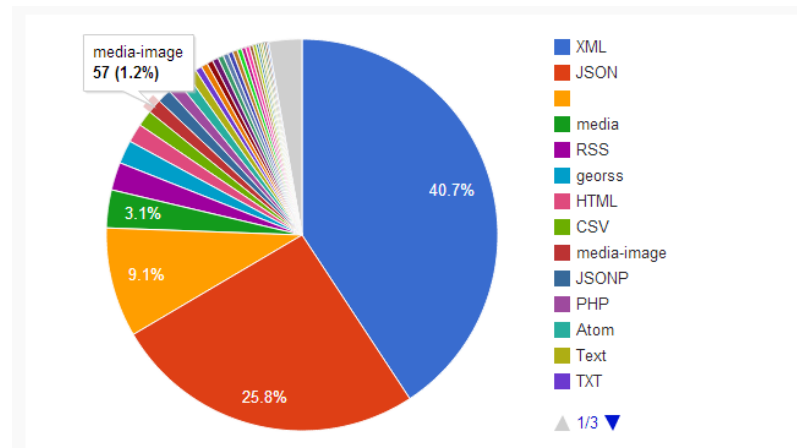


Figure 4.8: Pie chart

To generate a chart to visualize statistics, the user has to browse through the OMR administrator, selecting filters and making search queries using free text. When the user is satisfied with the results shown he can click on the button to generate the pertinent chart. The flow can be visualized in figure 4.9.

4.5.2.5 Admin authentication

The OMR Admin interface provides of basic authentication for write operations on the repository. This authentication is HTTP based⁹ using PHP.

4.5.2.6 Cache and performance

When working with thousands of components in the semantic repository some performance issues came up. To solve this we created some helpers written as functions inside the library that creates a file based cache.

The cache system works as follows. When a query to the OMR is executed (Figure 4.10) it stores the result shown in 4.6 using a no collision hash calculated using the SPARQL query. Every time a query is launched it will look for a stored result in the cache file list 4.5 and

⁹<http://php.net/manual/es/features.http-auth.php>

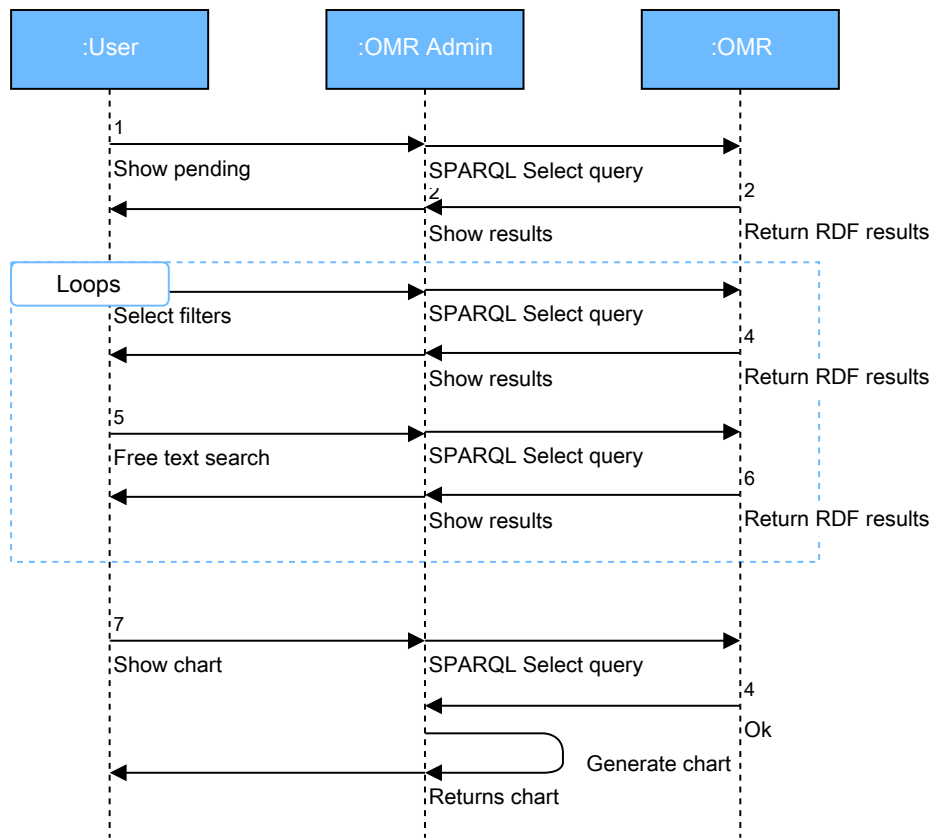


Figure 4.9: Sequence diagram for chart generation

it will be served in case there is a matching (Figure 4.11). The cached result is parsed the same way using sparqllib. This is quite important when working with a repository with thousands of services that will demand high processing time for all of them to be retrieved and filtered.

Table 4.2: Execution query time comparison

	OMR Quering	Local Sesame	Cached query
Maximun time	120 seconds	15 seconds	0 seconds
Minimum time	3 secods	1 second	0 seconds

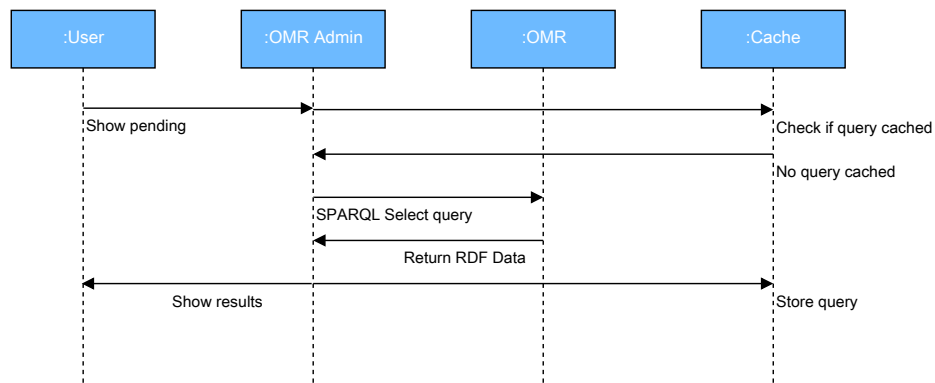


Figure 4.10: Sequence diagram if query is not in cache

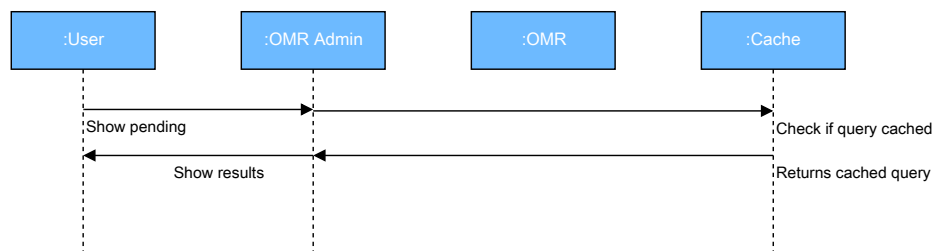


Figure 4.11: Sequence diagram if query cached

Listing 4.5: Cache file list example

```

pmoncada@lisa:/omr-admin/cache$ ls
07dd77d2e719486edf4c3fadd73bc57e.cache  607674268eb15fddb1f3f68966e60d8f.cache
afaef680863102e8717d8cf653607b00.cache
08e71b54c1ba078409a63b0514e05e5e.cache  63d0c0b4fddb2233ca11be1b05a90ff5.cache
b169c9805f3617f2bc54934dbca3f60f.cache
  
```

Listing 4.6: Cache file out example

```

pmoncada@lisa:/omr-admin/cache$ cat 07dd77d2e719486edf4c3fadd73bc57e.cache
<?xml version="1.0" encoding="utf-8"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="o" />
    <variable name="total" />
  </head>
  <results>
    <result>
      <binding name="o">
        <uri>http://www.ict-omelette.eu/omr/categories/shopping</uri>
      </binding>
    </result>
  </results>
</sparql>
  
```

```
<binding name="total">
  <literal datatype="http://www.w3.org/2001/XMLSchema#integer">37</literal>
</binding>
</result>
</results>
</sparql>
```

4.5.2.7 Repository wrapper

To connect to the semantic repository we use a wrapper that is in charge to do it using the *sparql library* (sub-section 4.5.2.8) and also using the cache system described in sub-section 4.5.2.6.

The class `omr.php` allows the system to connect to the different repositories available providing an API to it.

- `connect()`, connects to the repository endpoint.
- `query()`, executes a query in SPARQL language
- `insert()`, inserts a new element into the repository.

The possible repositories are the OMR, Sesame and LMF.¹⁰

4.5.2.8 Sparql Library

In order to handle RDF processing and SPARQL queries responses parsing, the system uses a third party library, `sparqllib.php`¹¹.

It has been modified to allow queries using POST method and HTTP authentication (which is a requisite for connecting to the OMR).

The library provides functions very similar to PHP `mysql_*` for comfort.

Listing 4.7: Sparqllib usage example

```
<?php
require_once( "sparqllib.php" );
$db = sparql_connect( "http://rdf.ecs.soton.ac.uk/sparql/" );
$db->ns( "foaf", "http://xmlns.com/foaf/0.1/" );
```

¹⁰Linked Media Framework

¹¹<http://graphite.ecs.soton.ac.uk/sparqllib/>

```

$sparql = "SELECT * WHERE { ?person a foaf:Person . ?person foaf:name ?name } LIMIT
5";
$result = $db->query( $sparql );
$fields = $result->field_array( $result );
print "<p>Number of rows: ".$result->num_rows( $result )." results.</p>";
print "<table class='example_table'><tr>";
foreach( $fields as $field ){ print "<th>$field</th>"; }
print "</tr>";
while( $row = $result->fetch_array() ){
    print "<tr>";
    foreach( $fields as $field ){
        print "<td>$row[$field]</td>";
    }
    print "</tr>";
}
print "</table>";
?>

```

Table 4.3: Sparqllib output example

Person	Name
bf4120a000000000	Bob
bf4120a12000000e	Alice

The output is shown in Table 4.3

4.5.2.9 Validating resource

The users browses into the repository trough the OMR admin interface going into the "Browse pending" section (Figure 4.12), selecting filters from the facet boxes (Figure 4.7) and using the free text search box (Figure 4.15) finally reaches a resource to be validated.

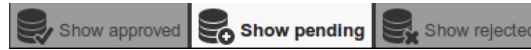


Figure 4.12: OMR admin top menu

When selecting the validate button (Figure 4.13), a SPARQL update query will be executed using the Sparql library to modify the OMR to indicate that the selected resource is already validated. Additionally, the RDF resource will be stored into the LMF to be indexed by SOLR.



Figure 4.13: OMR admin validate and reject buttons

Then, the validated services are shown in the *Show approved* section that can be accessed through the top menu as seen in Figure 4.12.

The whole sequence is shown in Figure 4.14.

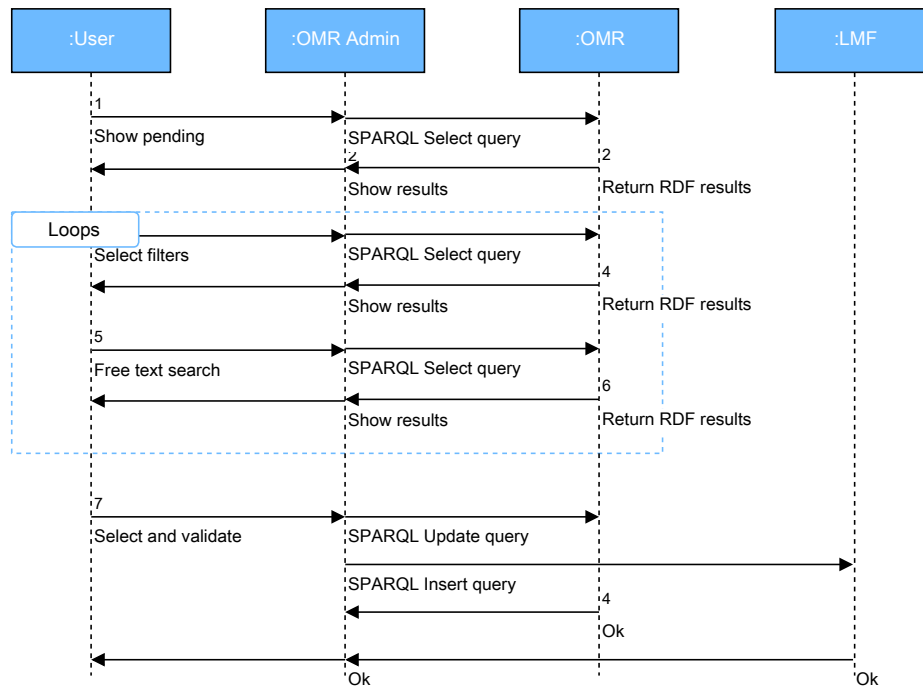


Figure 4.14: Sequence diagram for validating RDF resource

4.5.2.10 Rejecting resource

The users browses into the repository trough the OMR admin interface going into the "Browse pending" section (Figure 4.12), selecting filters from the facet boxes (Figure 4.7) and using the free text search box (Figure 4.15) finally reaches a resource to be validated. When selecting the reject button (Figure 4.13), a SPARQL update query will be executed using the Sparql library to modify the OMR to indicate that the selected resource has been rejected by the administrator.

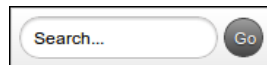


Figure 4.15: OMR admin search box

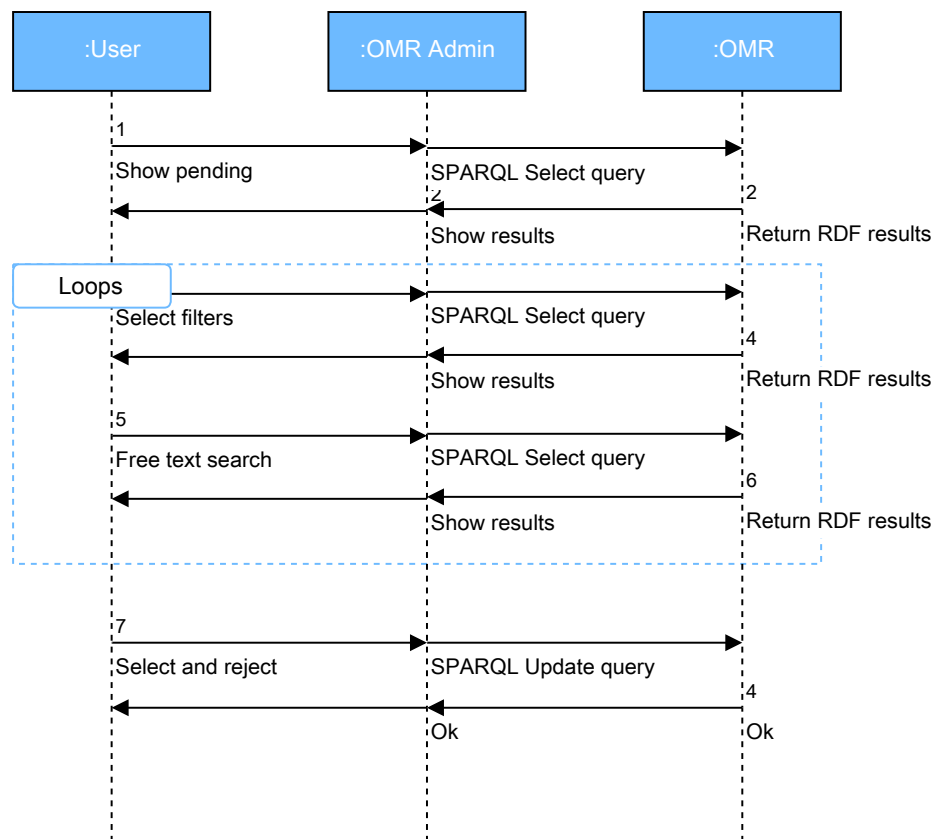


Figure 4.16: Sequence diagram for rejecting RDF resource

4.5.2.11 Wadl generation

The system can generate a WADL¹² file for those services with rosm¹³ description. **Important:** This is only available for services scrapped from Yahoo Pipes.

It returns a WADL file where it is specified the target URL and parameters for the REST service. This data is easily fetched executing SPARQL query as shown in 4.8.

¹² is a machine-readable XML description of HTTP-based web applications (typically REST web services)

¹³<http://www.wsmo.org/ns/rosm/0.1/>

Listing 4.8: Retrieving information for WADL with SPARQL

```

SELECT DISTINCT ?parameter ?getUrl WHERE{
  uriresource limon:describedBy ?a .
  ?a rosm:supportsOperation ?b .
  ?b hrests:hasAddress ?getUrl .
  ?b rosm:requestURIPParameter ?c .
  ?c rdfs:label ?d .
  ?d rdfs:label ?parameter
}

```

An example of WADL output is shown in 4.9.

Listing 4.9: WADL example

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
  <resources base="http://localhost:9998/storage/">
    <resource path="/containers">
      <method name="GET" id="getContainers">
        <response>
          <representation mediaType="application/xml"/>
        </response>
      </method>
    <resource path="{container}">
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema"
        type="xs:string" style="template" name="container"/>
      <method name="PUT" id="putContainer">
        <response>
          <representation mediaType="application/xml"/>
        </response>
      </method>
      <method name="DELETE" id="deleteContainer"/>
      <method name="GET" id="getContainer">
        <request>
          <param xmlns:xs="http://www.w3.org/2001/XMLSchema"
            type="xs:string" style="query" name="search"/>
        </request>
        <response>
          <representation mediaType="application/xml"/>
        </response>
      </method>
    </resource>
  </resources>
</application>

```

To generate WADL, the user has to browse through the OMR Admin and select a mashup

or service. Afterwards, if it is supported by the selected service, the user can request the OMR Admin to generate a WADL file. This sequence is described in figure 4.17.

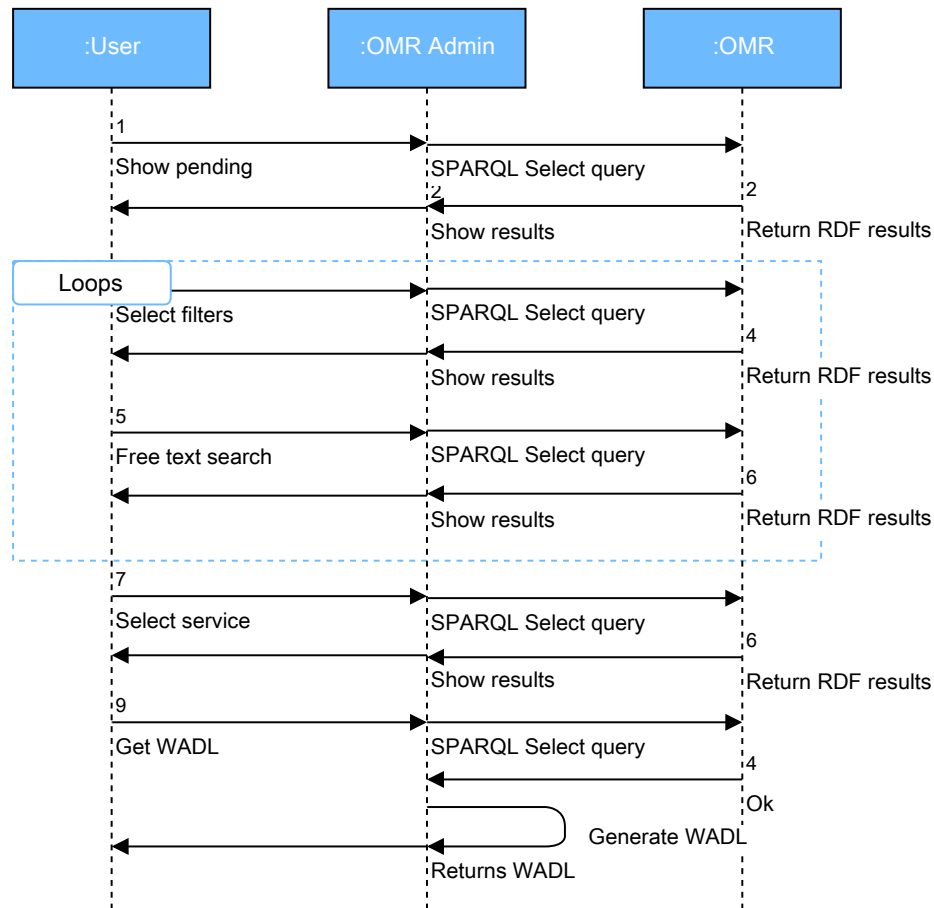


Figure 4.17: Sequence diagram for WADL file generation

4.5.2.12 LMF integration

There is an extra module¹⁴ that adapts the queries to insert selected services into an LFM repository. This enables SOLR indexing for the client service browser.

It connects to the sparql/update endpoint of LMF, in which SPARQL insert queries can be executed.

¹⁴lmf.php

4.6 OMR Client Browser

This interface is used by the final user when he wants to find suitable widgets or services to build new mash-ups. It connects to the repository and reads the components approved formerly by the administrator user via the OMR Admin interface.

The OMR Client Browser (4.19) consists in two parts, a back-end formed by LMF, and a front-end written using web technologies such as javascript and HTML. It is based on the Episteme¹⁵ platform.

The user can create a new search which will be automatically saved for posterity. The user will be shown a form with auto complete fields (which will retrieve the auto-complete data automatically (step 2 in Figure 4.18)). The user selects the desired filters (step 3 in Figure 4.18). A HTTP request to the LMF is made to retrieve the search results (step 4 in Figure 4.18). If any of the fields was semantic, an extra HTTP request will be done to the semantic module. The results are shown to the user (step 5 in Figure 4.18).

All the modules are described in the following points.

4.6.1 Back-end

For the back-end the system uses Linked Media Framework to build semantic search over the approved data selected with the OMR Admin described in 4.5. The only thing needed to configure the LMF is a semantic core that should follow the rules showed in 4.10. A search core represents a specific index configuration with fields filled from the linked data cloud. It consists of two mayor elements: a filter deciding which resources are added to the search index, and one or more fields defining the index fields of the search core and how the values of these fields are calculated. These filters are written following LD Path¹⁶, a simple path-based query language, similar to SPARQL Property Paths, that is particularly well-suited for querying and retrieving resources from the Linked Data Cloud by following RDF links between resources and servers.

¹⁵Episteme is a creator of opportunities designed to link the various partners and create consortia to suit a particular offer. With semantic search is able to find companies that are the best suited to an opportunity based on their characteristics.

¹⁶<http://code.google.com/p/ldpath/>

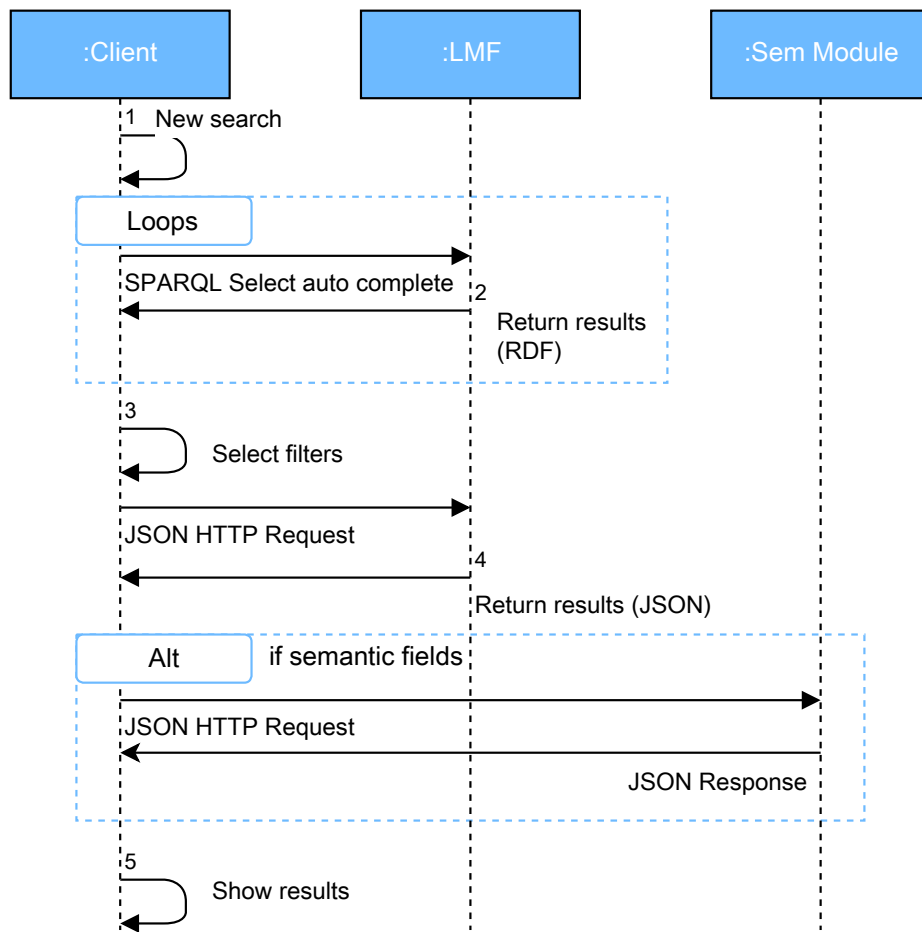


Figure 4.18: Scrappy sequence diagram

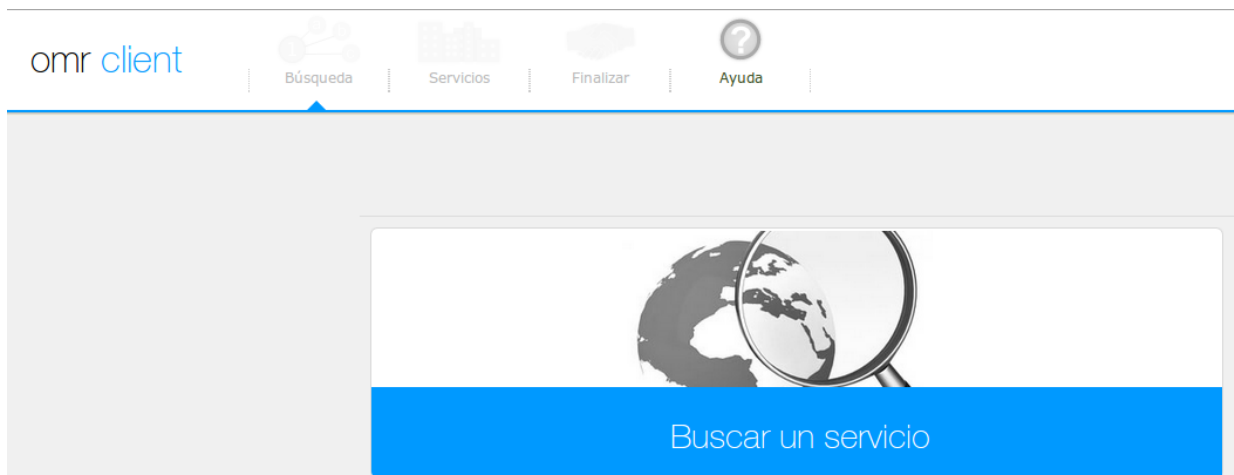


Figure 4.19: OMR client interface

Listing 4.10: Semantic core configuration for LMF using LD Path

```

@prefix limon : <http://www.ict-omelette.eu/schema.rdf#> ;
@prefix ctag : <http://commontag.org/ns#>;

Rdftype = rdf:type :: xsd:string ;
categorizedBy = limon:categorizedBy :: xsd:string ;
api = limon:api :: xsd:string ;
sslSupport = limon:sslSupport :: xsd:string ;
label = rdfs:label :: xsd:string ;
protocol = limon:protocol :: xsd:string ;
tagged = ctag:tagged :: xsd:string ;
authentication = limon:authentication :: xsd:string ;
dataFormat = limon:dataFormat :: xsd:string ;
description = dc:description :: xsd:string ;
Provenance = "gsi" :: xsd:string ;

```

4.6.2 Front-end

The Episteme front-end has been developed using various JavaScript frameworks, such as Knockout JS which simplifies the dynamic JavaScript UI with the Model-View-View Model (MVVM) pattern, and KendoUI¹⁷ to build beautiful interactive interface. Creating a search engine with Episteme has been done following few steps.

Search fields If we want to show new search fields as shown in figure 4.20 first of all we have to construct the SPARQL query that will fill the auto-complete. For example, one of them should look like described in 4.11. It is also necessary to add the data binding sequence with knockout as shown in 4.12.

¹⁷<http://www.kendoui.com>

Listing 4.11: SPARQL to retrieve all component types

```
SELECT DISTINCT ?o
  WHERE { ?a rdf:type ?o . }
  GROUP BY ?o ORDER BY DESC(?o)
}
```

The screenshot shows three search filter sections in the OMR client interface. Each section has a label, a dropdown menu, a '+' button, and a list of results. The first section is labeled 'Tipo de componente' and shows a result 'service x'. The second section is labeled 'Categorizado por' and shows a result 'social x'. The third section is labeled '¿Cuenta de desarrollador requerida?' and shows a result 'false x'.

Figure 4.20: Search fields in OMR client interface

Listing 4.12: Data binding with knockout

```
<!-- RDF TYPE -->
<span data-bind="if: data.field() == 'Rdftype'">
  <span class="filterName" data-bind="text: root.lang().rdftype"></span>
  <input class="solrInput" data-bind="kendoComboBox: { data: root.Rdftype,
    value: root.selectedRdftype}" />
  <a class="greenButton" data-bind="click: root.addSolrFilter.bind(data,
    data.values, root.selectedRdftype)">+</a>
  <a class="filterInfo blueButton" data-bind="click:
    root.addSolrFilter.bind(data, data.values, root.selectedType),attr: {
    'title': root.lang().rdftypeHelp}">?</a>
</span>
```

Multilingual support Episteme is prepared to enable multi-language, all the string values are stored in the *dictionary.js*.

Personalization manual is available with full detail at the Episteme Wiki.¹⁸

4.7 Conclusions

We have shown a architecture fully modular in which each component can be developed, maintained and deployed separately.

The automated discovery system could be upgraded to acquire new functionalities and adapt to new web structure.

As we demonstrated the OMR module could be substituted by an other framework to store the RDF data such as Sesame as we built the communication interface complying the standards.

Using modules built using the Model-View-View Model patters makes them reusable and extended for other purposes.

All modules are available as open source projects.

- Sesame: <http://www.openrdf.org/download.jsp>
- LMF: <https://code.google.com/p/lmf/downloads>
- Scrappy: <https://github.com/gsi-upm/scrappy>
- OMR admin: <https://github.com/gsi-upm/omr-admin>
- OMR client: <https://github.com/gsi-upm/omr-client>

¹⁸<https://github.com/gsi-upm/Episteme/wiki/Create-a-custom-search-engine-with-Episteme>

Prototype and example usage

In this chapter we are going to describe a selected use case. It is going to be explained the running of all the tools involved and its purpose. It is based on how to crawl the web to find new mashups, then feed the repository, do the validation and rejections of the mashups, and finally the developer will be able to use the discovered services.

5.1 Introduction

In this use case 2 actors are involved, the administrator and the developer user.

Actor identifier	Role	Description
ACT-1	Admin	Administrator of the OMR, in charge of tasks such as inserting, deleting mashups, as well as including new available mashup repositories..
ACT-2	Developer	Technical developer which uses the OMR.

Table 5.1: Actors list

The goal of the administrator is to provide the user the content that he needs.

In this context, the developer is building a web application that consists of a social network in which users need to take photos and share them with other users. After sharing them, the photos need to be represented in a map. The developer wants to find an on-line service that already provides this functionality.

The developer user will query the repository using the OMR Web developer interface. The repository must have been fed up previously by an administrator, who doesn't know exactly the services that developers will need but he can think of which might be suitable for them.

To achieve the goals defined before we are going to follow the following steps. First of all launch scrappy to obtain all the web resources we need to feed the repository, once this has been done, the administrator user has to use de OMR admin interface in order to select the mash-ups that the developer user could possibly need for his application. The developer user will be able to find the mash-ups needed using the OMR Web developer interface.

In this scenario each module will run separately to demonstrate that they all can be standalone applications.

There is going to be a remote repository (OMR) located in Chemnitz University of Technology (Germany). The automated discovery process is done using one of the computers in the laboratory of Grupo de Sistemas Inteligentes. Both Administrative interface and

developer interface will run in a web server located also in Grupo de Sistemas Inteligentes. The LMF will run in a separated server in the laboratory. This is resumed in table 5.2.

Table 5.2: Execution enviroment

Component	where it runs
Omelette Mashup Registry (OMR)	Chemnitz University of Technology, (Germany)
Automated discovery service	Laboratory, GSI (shannon.gsi.dit.upm.es)
Ranking module	Laboratory, GSI, (shannon.gsi.dit.upm.es)
OMR Administrative interface	Laboratory, GSI, (minsky.gsi.dit.upm.es)
OMR Web developer interface	Laboratory, GSI, (minsky.gsi.dit.upm.es)
Linked Media Framework (LMF)	Laboratory, GSI, (krusti.gsi.dit.upm.es)

5.2 Automatic service discovery

As explained in section 4.2, the discovery of new services and mashups is done using Scrappy. Installation and configuration of Scrappy can be found in appendix A.

The administrator user has to run Scrappy and insert the results into the OMR. It is important to have configured the OMR endpoint in the config file as explained in appendix A.

For this example we are going to scrap the sites of *Programmable Web*, *Opera Widgets* and *Yahoo Pipes*.

Listing 5.1: Scrappy launching

```
scrappy -g programmableweb.com
scrappy -g pipes.yahoo.com
scrappy -g widgets.opera.com
```

By executing Scrappy it will directly feed the repository with services in RDF format like listing 5.2.

Table 5.3: Scrappy execution statistics

Elapsed time	1 day 3 hours and 31 minutes
--------------	------------------------------

Table 5.4: OMR components summary

Number of services	10194
Number of widgets	1804
Number of applications	7032
Total number of components	11998


After the time shown in table 5.3 the number of mash-ups shown in table 5.4 will be inserted in the repository. In this case we have directly use the OMR, which might have slightly slowed down the process as it needs to do HTTP connections to an external server

(and far miles away).

In the figure 5.1 we can see an example of page that is going to be converted into RDF by Scrappy. In listing 5.2 we can see the output that produces Scrappy for that webpage.

Listing 5.2: Full RDF of scrapped widget


```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:ctag="http://commontag.org/ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:limon="http://www.ict-omelette.eu/schema.rdf#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#">
  <skos:Concept rdf:about="http://www.ict-omelette.eu/omr/categories/mapping">
    <rdfs:label>Mapping</rdfs:label>
  </skos:Concept>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/display">
    <rdfs:label>display</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/mapping">
    <rdfs:label>mapping</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/places">
    <rdfs:label>places</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/viewer">
    <rdfs:label>viewer</rdfs:label>
  </ctag:Tag>
  <limon:Service rdf:about="http://www.programmableweb.com/api/google-maps">
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/display"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/mapping"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/places"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/viewer"/>
    <dc:description>The Google Maps API allow for the embedding of Google Maps
    onto web pages of outside developers, using a simple JavaScript interface or
    a Flash interface. It is designed to work on both mobile devices as well as
    traditional desktop browser applications. The API includes language
    localization for over 50 languages, region localization and geocoding, and
    has mechanisms for enterprise developers who want to utilize the Google Maps
    API within an intranet. The API HTTP services can be accessed over a secure
```



[Home](#)
[API News](#)
[API Directory](#)
[Mashups](#)
[Community](#)
[How-to](#)

Google Maps API

[Summary](#)
[Mashups \(2526\)](#)
[How-To \(38\)](#)
[Developers \(1011\)](#)
[Comments \(17\)](#)



The Google Maps API allow for the embedding of Google Maps onto web pages of outside developers, using a simple JavaScript interface or a Flash interface. It is designed to work on both mobile devices as well as traditional desktop browser applications. The API includes language localization for over 50 languages, region localization and geocoding, and has mechanisms for enterprise developers who want to utilize the Google Maps API within an intranet. The API HTTP services can be accessed over a secure (HTTPS) connection by Google Maps

[dc:description](#)

Google Maps: Highlights

Summary	Mapping services
Category	Mapping limon:categorizedBy
Tags	mapping places viewer display ctag:tagged
Protocols	JavaScript limon:protocol
Data Formats	XML , VML , JSON , KML limon:dataFormat
API home	https://developers.google.com/maps/ limon:api

Google Maps: Specifications

Functionality

Client Install Required	No	limon:installRequired
-------------------------	----	---------------------------------------

Signup and Licensing

Developer Key Required	Yes	limon:developerKeyRequired
Account Required	No	limon:AccountRequired
Commercial Licensing	Contact provider	
Provider	google.com	limon:provider
Usage Fees	None specified	limon:usageFees
Usage Limits	50,000 geocode requests per day	limon:usageLimits
Terms of Service	code.google.com/apis/maps/terms.html limon:termsAndConditions	

Security

Authentication Model	API Key	limon:authentication
SSL Support	No	limon:sslSupport
Read-only Without Login	Yes	limon:readOnlyWithoutLogin

Figure 5.1: Original site and corresponding LiMOn mapping

```

(HTTPS) connection by Google Maps API Premier customers.</dc:description>
<dc:source rdf:resource="http://www.programmableweb.com/api/google-maps"/>
<limon:api rdf:resource="http://code.google.com/apis/maps/index.html"/>
<limon:apiBlog rdf:resource="http://googlegeodevelopers.blogspot.com/">
<limon:apiForum
rdf:resource="http://groups.google.com/Google-Maps-API?pli=1"/>
<limon:authentication>API Key</omelette:authentication>
<limon:categorizedBy
rdf:resource="http://www.ict-omelette.eu/omr/categories/mapping"/>
<limon:installRequired>>false</omelette:installRequired>
<limon:dataFormat>JSON</omelette:dataFormat>
<limon:dataFormat>KML</omelette:dataFormat>
<limon:dataFormat>VML</omelette:dataFormat>
<limon:dataFormat>XML</omelette:dataFormat>
<limon:AccountRequired>>false</omelette:AccountRequired>
<limon:developerKeyRequired>>true</omelette:developerKeyRequired>
<limon:protocol>JavaScript</omelette:protocol>
<limon:provider rdf:resource="http://google.com"/>
<limon:rating>0.8201093479130551</omelette:rating>
<limon:readOnlyWithoutLogin>>true</omelette:readOnlyWithoutLogin>
<limon:sslSupport>>false</omelette:sslSupport>
<limon:termsAndConditions
rdf:resource="http://code.google.com/apis/maps/terms.html"/>
<limon:usageFees>None specified</omelette:usageFees>
<limon:usageLimits>50,000 geocode requests per day</omelette:usageLimits>
<limon:vendorApiKit>JavaScript library</omelette:vendorApiKit>
<rdfs:label>Google Maps API</rdfs:label>
</omelette:Service>
</rdf:RDF>

```

5.3 Ranking algorithm

The ranking algorithm needs to be executed by the administrator before accessing the OMR admin interface if we want to visualize the metrics corresponding to ranking. It could have done parallel to the scrapping process, but as explained in section 4.4 in order to calculate in an accurate way the ranking indexes all the mash-ups need to be present at the same time because there is correlated information. In the same way, if a new service is discovered by Scrappy because we repeated the process in section 5.2 we will need to execute the ranking algorithm again.

Before executing the algorithm, it needs to have an instance of Scrappy running as a web server. This is explained in appendix A.2.

The administrator user has to configure the configuration file `configuration.properties` (listing 5.3) and then run the algorithm .

```
bash startRanking.sh
```

Listing 5.3: Ranking algorithm configuration file

```
#OMR endpoint
omr=https://vsr-web.informatik.tu-chemnitz.de/omr-write/components/sparql

#Username
user=omr-client-upm

#Password
password=omr.client.upm.2012
```

Table 5.5: Ranking algorithm execution statistics

Elapsed time	7 hours and 4 minutes
---------------------	-----------------------

This process could take long time as showed in table 5.5.

Listing 5.4: Ranking algorithm output

```
<limon:DegCent>0.82</omelette:rating>
<limon:ClosCent>4.21</omelette:rating>
<limon:RatSoc>131</omelette:rating>
```

5.4 OMR administrative interface

After having the repository filled with all the services our purpose is to select those which we consider as interesting and also reject those which we don't want. The administrator must enter the administrative interface (see appendix C for installation and configuration). In this scenario this is done by opening in the webbrowser the following URL:

```
http://lab.gsi.dit.upm.es/~pmoncada/omr-admin-pfc
```

The administrator will see the main view showed in figure 5.2.

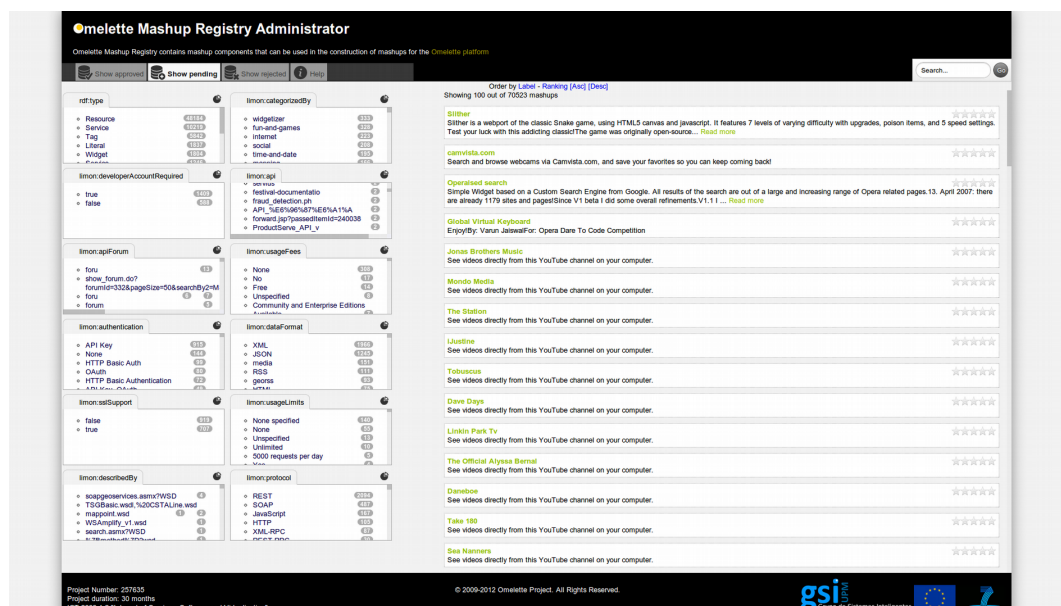


Figure 5.2: Main view OMR Administrator

In the main view of the OMR Administrator, a menu is shown at the top:

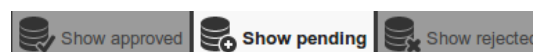


Figure 5.3: OMR admin top menu

Show approved shows those mash-ups which have been selected by the administrator as approved.

Show pending shows everything that that has been returned has the output of the scraping process.

Show rejected shows the lists of mash-ups that have been previously rejected by the administrator.

5.4.1 Available general actions

The following options will help the administrator to find those components he thinks of they are interesting or those who might be rejected.

5.4.1.1 Filtering

To filter using the filtering boxes click on a property from a filtering category. Many filters can be selected at the same time. First select one of them, the view will refresh with the new results. Then select next filter, the former filter will be still selected, and results will be filtered by many filters as chosen.

Example

We want add mash-ups compatible with *JSON* and *XML* data formats and supporting *REST*, *JavaScript* and *HTTP* protocols. To filter the mash-ups with this criteria, we select it in the facet boxes.

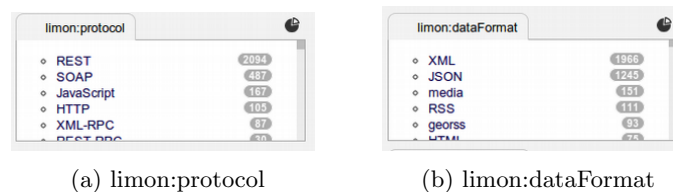


Figure 5.4: Filtering by LiMOn properties

5.4.1.2 Searching

The administrator user can use the search box to find a mash-up by name or description.

Example

We write "maps" in the search box (Figure 5.5), and the results are shown in the result area.



Figure 5.5: OMR admin search box

5.4.1.3 Obtaining help

If the administrator does not the meaning of anything appearing in the interface can consult the interactive help by mousing over the help icon.

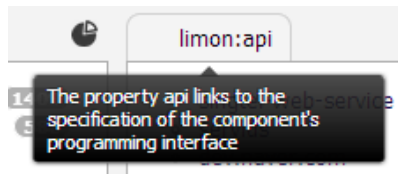


Figure 5.6: OMR admin help

5.4.1.4 Statistics

The administrator might know how many mash-ups of a kind are added to the repository. When large amount of services are present is kind of difficult to handle this information. The administrator user can check this information in graphics presentation (Figure ref-fig:omradminseestatistics).

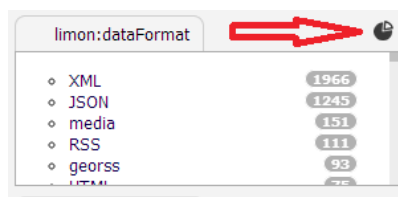


Figure 5.7: OMR see statistics

5.4.2 Reject a resource

There are several reasons why we would like to reject various discovered mash-ups, such as the functionality of its is not related to our working lines or it was wrong scrapped in the automated discovery process (bad character encoding like in figure 5.8, mismatch of the information, etc). We could think just in deleting them from the system, but there is the possibility that we change our mind about this action in the future and we regret it. Also this avoids repeating the scrapping process from re-inserting the mashup again in the repository. This is way the admin interface will allow us to mark the mash-up as rejected as explained in section 4.5.2.9.

Before validating a resource we may want to know more information about it, and this can be done by visualizing the full content of the service. This must be done before validat-



Figure 5.8: Bad charset encoding in widget description

ing, even if we are mostly sure about the component is the desired one, but sometimes as explained in the former paragraph there could be some problems in the scrapped information. The duty of the admin is to make sure the services that are validated are the correct ones.

The administrator can use the tools explained section 5.4.1 to get further information and help to complete the process. As an example the administrator could need extra information about the different fields and this can be achieved using the mouse-over functionality.

In our use case we want to have uniform types of mash-ups in the repository so the statistics diagrams provided by the admin interface would result very useful. We can find how to use them in section 5.4.1.4.

5.4.3 Validating resource example

As we saw in listing 5.2 a service from Google Maps was scrapped, so let's try to find it in the admin interface. The easiest way would be to type "Google Maps API" in the search box and the result would be the desired one. But considering we don't know anything about the scrapped mashup this is not useful. Nevertheless we know some characteristics about a mash-up we need, which are:

- It is a widget for maps.
- It must be compatible with XML and JSON data format.
- It must support javascript.

Doing a fast search just writing "maps" in the search box and after this selecting "JSON", "XML" and "Javascript" in the facet boxes the desired result (figure 5.9 will be showed on the right.



Figure 5.9: Google maps api service in admin interface

To validate this service, the administrator has to click on the check box and then click on the *validate* button to submit the data.

In order to understand the process of validating resources see section 4.5.2.9.

When we have selected and validated all the services we wanted, our task as administration user will be finished. This task can be resumed at any time if more needed services come out.

5.5 Web developer interface

This application is used by the developer user, who wants to find suitable services to build an application. It is done using the OMR client interface through the web browser.

`http://krusti.gsi.dit.upm.es:8080/omr/`

The interface of this application is much clearer as easier to use than the administrative one, as it is built for end users. Also the functionalities are not exactly the same, in this second search engine there are intelligent technologies as semantic search.

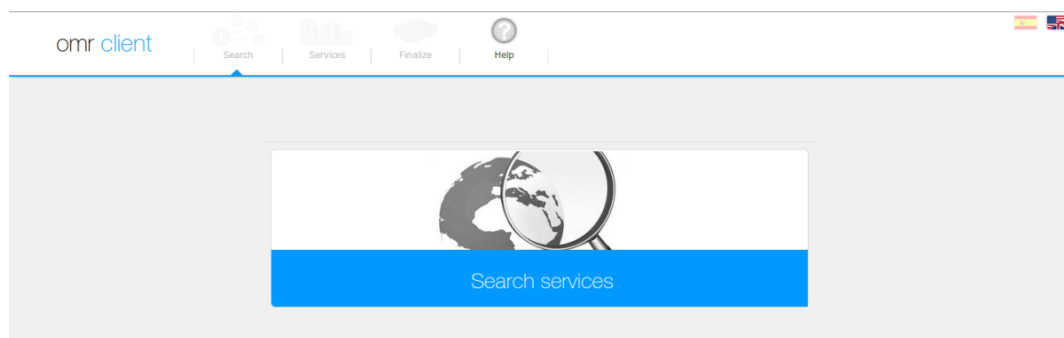


Figure 5.10: OMR Developer interface main

Example

Following the example exposed formerly, a developer wants to embed a map into an application to show the users interesting places. He doesn't know which one will better suit into his requirements. It must be a *widget* and does not want to create a developer account.

The developer user accesses the interface and clicks on the "Search service" button (figure 5.11).

In the next step we can create a new search or select a search (figure 5.14) we did before and we want to reuse it or refine it. The only difference will be that the form in next step

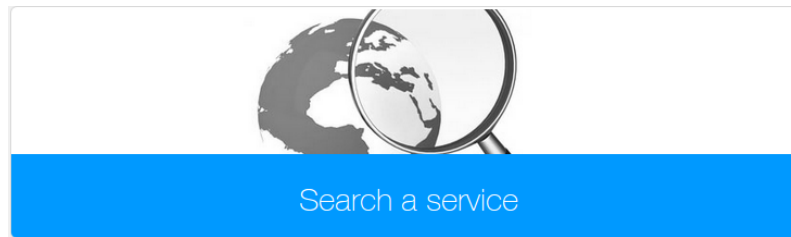


Figure 5.11: Search service button

will be filled or will be blank.

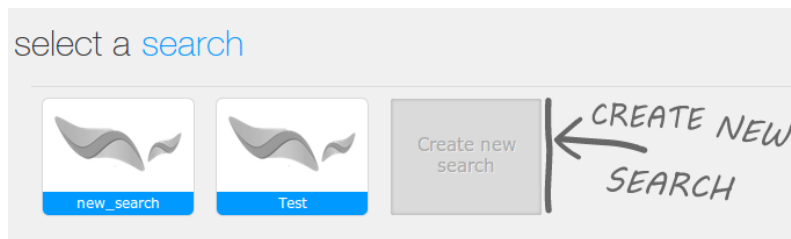


Figure 5.12: Select saved search or create a new one

Creating a new search will show us a fillable form which we will fill in according to our search criteria.

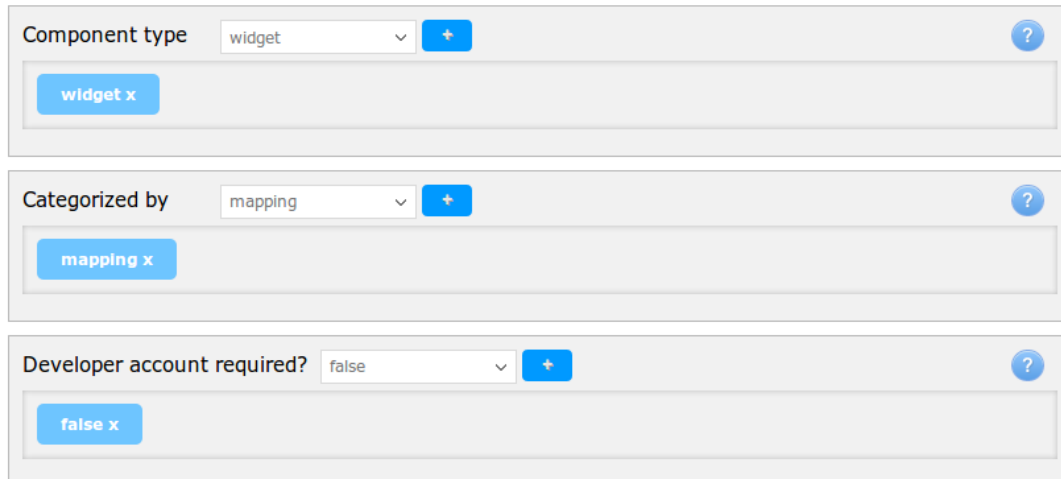
A form with three filter sections. Each section has a dropdown menu, a blue "+" button, and a blue button with an "x" to remove the filter. The first section is "Component type" with "widget" selected and a "widget x" button. The second section is "Categorized by" with "mapping" selected and a "mapping x" button. The third section is "Developer account required?" with "false" selected and a "false x" button. Each section also has a blue "?" icon for help.

Figure 5.13: Select filters for the search



Figure 5.14: Show results

And after pressing the "Search results" the semantic search will be done in background by the semantic module and they will be presented as shown in figure 5.15. The first three services are marked with *gold*, *silver*, or *bronze* medals which indicate that those are the best 3 services found by the semantic module. At the bottom the developer can see recommended services. These are services ordered by the ranking module having into account the social index.

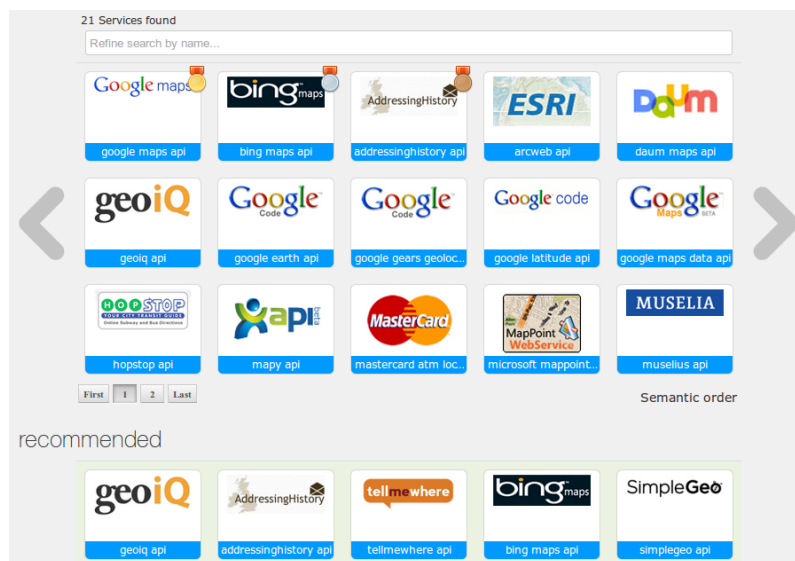


Figure 5.15: Found services by the semantic module

We can extend the information (figure 5.16) and check if it satisfies us to finally select it as a candidate to use it in our application.

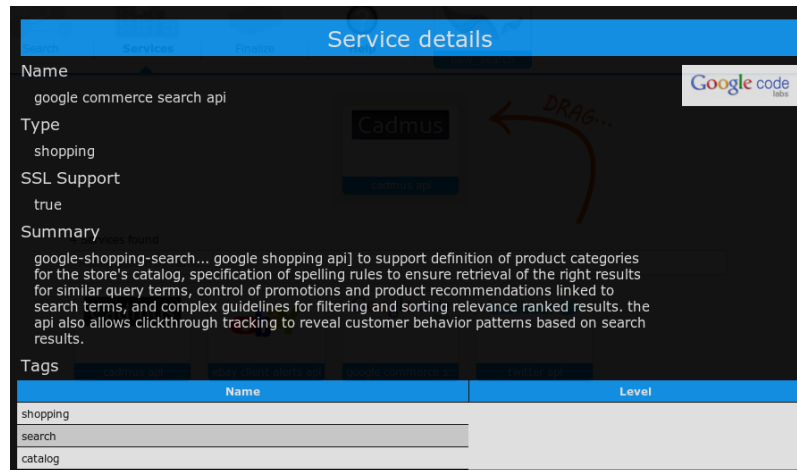


Figure 5.16: Extended service info

5.6 Conclusions

It is a very powerful search engine that enables real semantic search functionality to the final user, enabling search using natural language and offering similar results that suits into the user preferences that at a first instance he wouldn't have even thought about.

The setting up of the scenario (filling in the repository and filtering) is quite slow but comparing it to the results given after is more than acceptable. The human intervention of an user administrator to filter the mash-ups gives the final user a great search experience which cannot be done by full automatic search engine of this characteristics.

Conclusions and future lines

In this chapter we will describe the conclusions extracted from this master thesis, the achievements and thinkings about future work.

6.1 Conclusions

By fetching existing services and widgets on the Internet, we have created a powerful search engine without having any own content, enabling final user find those services that he needs, that is part of the mash-up generation philosophy.

This project has been developed in the scope of an European FP7 project contributing to the automated discovery section. This helped us to solve problems that without being part of a big problem we would not have noticed, like taking into account that final endpoint servers are not going to be as powerful as developing ones or making the system scalable.

Also helped to extend functionalities of other projects, such as Episteme.

We have used existing technologies whenever it was possible, and this helped improving them when more complex requisites where necessary. As an example, this happened with Scrappy, which was modified, created a branch and then merged again with extra functionalities.

Dividing the project into different modules forced us to rely on existing web and software standards which helped us to integrate and interconnect all of them.

We experienced big changes as early technology adopters, such as new versions of the SPARQL language fixing bugs and creating new functionalities. We have dared to use extremely new technologies still in alpha developments, with its pros and cons.

6.2 Achieved goals

Fetch content automatically This goal has been achieved successfully. We have been able to discover existing content from websites and then fetch it. This helped us to have a big repository of services and content that the user will need. This is deeply described in chapter 4.2.

Structure content It was essential to structure the content fetched from the Internet. This has been possible thanks to semantic and linked data technologies. This is detailed in chapter 2.3.2

Store content An other crucial feature that was successfully implemented was how to store all the information without losing the structured format described before. This has been achieved by using semantic repositories. More information can be consulted in chapter 4.3.

Rank content The content without knowing if it is useful or not is useless. One of the main goals was to make possible to distinguish good and bad discovered content. This has been achieved by defining and implementing algorithms that will automatically rank the stored content. The algorithm and its implementation is explained in chapter `refsec:rankingmodule`.

Manage content It was completely necessary to manage and administer the discovered content by the administrator user. This has been achieved by creating an interface that allows searching and filtering into the automated discovered content and after selecting the useful content. This is described in chapter 4.5.

Search content The main goal of the project was enabling the user search the services he needed. This has been done by creating an easy-to-use interface that queries the repository and lets the user find suitable services. To see detailed information see chapter 4.6.

Suggest similar content Many times the user does not know what he wants, an other important role of the project was suggesting the user possible results that matches his needs. This has been achieved by using semantic search empowered technologies. Nevertheless, this has been implemented in a simpler scenario using different content and different ontology. This feature is available in the Job Matching demo¹.

6.3 Future work

There are several lines than can be followed to continue and extend features of this work.

In the following points some fields of study or improvement are presented to continue the development.

- In automated discovery enable only search for new services. Without scrapping the entire web again we would gain a lot if processing time and this could make possible more frequent discoveries and more up to date content.
- Also fetching services and widgets from new Internet repositories.
- Not only fetch content from existing repositories, explore the web to discover new ones. This makes possible to have different content and always new and undiscovered services.

¹<http://demos.gsi.dit.upm.es/job-matching/>

- Make the discovering service to be launched from web interface. Now it is launched from console. Integrated into the administrative interface would be optimal.
- Make discovery date visible. This would make possible to know if the information about a service is reliable. Also making possible to update information of only selected services.
- New ways of ranking content, based on social networks and popularity on the Internet.
- Categorize widgets and services for existing platforms. For example Wordpress² plugins.
- Discover mobile services. Content discovered right now is desktop oriented.
- Let the user try and show a demonstration of the services and widgets in the search interface.
- Semantic search and suggestions to find similar content that could fit in the user's preferences. This has been already done in a smaller scenario as a proof of concept.

²<http://wordpress.org/>

Installing and configuring Scrappy

This tutorial goes through the process of installing and configuring Scrappy to be able crawl services for the OMELETTE project. Scrappy's code is available at <https://github.com/gsi-upm/scrappy> However, in order to use it with OMR, we have introduced some modifications. The new code can be found in the branch "omr".

A.1 Installation

A.1.1 Requirements

- Graphviz
- Raptor library: raptor-utils
- Ruby 1.8
- Ruby Gems
- Sesame 2.0¹ If the rdf is stored in sesame server.

¹<http://www.openrdf.org/doc/sesame2/users/ch06.html>

- The gems listed below should be installed. In linux, this can be done, for example 'sudo gem install nokogiri'.

- nokogiri 1.5.4
- lightrdf 0.4.1
- il8n 0.6.0
- iconv 0.1
- multi_json 1.3.6
- ntlm-http 0.1.1
- webrobots 0.0.13
- sinatra 1.3.2
- sinatra-flash 0.3.0
- eventmachine 0.12.10
- mechanize 2.5.1
- tilt 1.3.3
- haml
- rack 1.4.2
- rack-protection 1.2.0
- rack-flash 0.1.2

A.1.2 Installation steps

- Install ruby and rubygems
- Scrappy can be installed as a standard Ruby gem. However, some of the above gems may have extra dependencies. In particular, nokogiri requires several system libraries to be installed, and sinatra-flash may not be automatically installed, so you need to install them manually before installing scrappy:

```
sudo apt-get install ruby-dev libxslt-dev libxml2-dev
sudo gem install sinatra-flash
sudo gem install scrappy
```

- If you want to run scrappy with the OMR, you need to get the modified version from github, of both scrappy and lightrdf.


```
git clone https://github.com/gsi-upm/scrappy.git
git clone https://github.com/gsi-upm/lightrdf.git
cd scrappy
git checkout omr
cd ../lightrdf
git checkout omr
```

This will create a folder named “scrappy”, (along with the new lightrdf folder). Inside it, in the “bin” subdirectory you can find the runnable for scrappy. Be careful to maintain the scrappy and lightrdf folder in the same directories, so scrappy can find the correct lightrdf to load.

A.2 User manual

Scrappy provides a set of interfaces to extract RDF from a web page. In order to see the available options, the user can execute ‘scrappy –help’ as follows.

```
scrappy --help

Scrappy v0.4.10
Synopsis
Scrappy is a tool to scrape semantic data out of the unstructured web
Examples
This command retrieves a web page
scrappy -g http://www.example.com
Usage
scrappy [options]
For help use: scrappy -h
Options
-h, --help
Displays help message
-v, --version
Displays the version, then exits
-f, --format
Picks output format (json, ejson, rdf, ntriples, png)
-g, --get URL
Gets requested URL
-p, --post URL
Posts requested URL
-c, --concurrency VALUE Sets number of concurrent connections for crawling (default
    is
10)
-l, --levels VALUE
Sets recursion levels for resource crawling (default is infinite
crawling)
-d, --delay VALUE
```

```
Sets delay (in ms) between requests (default is 0)
-D, --dump
Dumps RDF data to disk
-u, --debug [KEYWORD] Shows debugging traces. Use optional keyword to filter
selectors' output
-o, --observe URLs
Observes the specified URLs storing their data into the repository
-s, --server [ROOT]
Runs web server (optionally specify server's root url)
-a, --admin [ROOT]
Runs admin web server (optionally specify server's root url)
-P, --port PORT
Selects port number (default is 3434)
-t, --time TIME
Returns repository data from the last given minutes
```

Scrappy provides several interfaces: command line interface, web admin interface, web service interface and Ruby interface.

A.2.1 Command line interface

The command line interface can be used in a command shell window as shown in Figure 2. For example, the web example.com can be scraped with the following command.

```
scrappy -g example.com
```

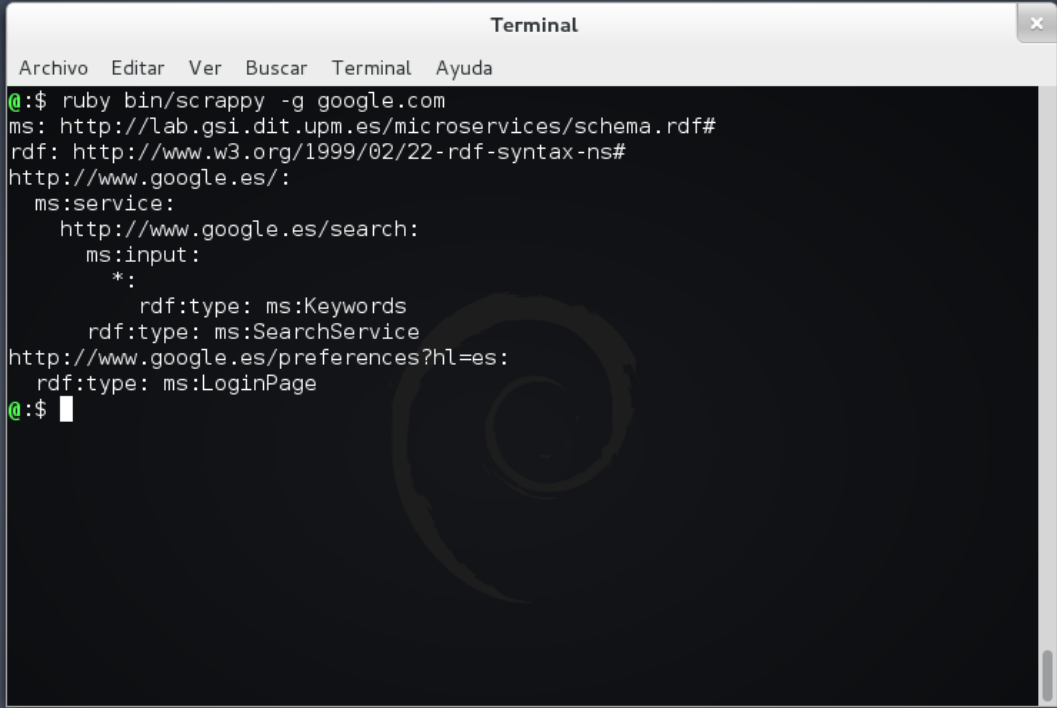
For example, the output from google.com would look like the following window. Be advised: most of the times the RDF data will not fit in a single window, so its recommended to pipe it to another command or a text file.

A.2.2 Web admin interface

The web admin interface can be used in a regular web browser, to either scrape a site or browse the different resources, such as the extractors. To launch it, you need to open a command line, and execute the following command:

```
scrappy -a
Launching Scrappy Web Admin (browse http://localhost:3434)...
== Sinatra/1.1.3 has taken the stage on 3434 for production with backup from Thin
```

Once scrappy has been launched, user can access the admin interface (Figure A.2). In order to scrape a web site, it is only required to introduce the URI and select the desired output (RDF, JSON, YARF or PNG).



```

Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
@:$ ruby bin/scrappy -g google.com
ms: http://lab.gsi.dit.upm.es/microservices/schema.rdf#
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
http://www.google.es/:
  ms:service:
    http://www.google.es/search:
      ms:input:
        *:
          rdf:type: ms:Keywords
          rdf:type: ms:SearchService
http://www.google.es/preferences?hl=es:
  rdf:type: ms:LoginPage
@:$

```

Figure A.1: Scrappy command line interface

This interface also allows managing the extractors from each page (extractor tab, Figure A.3), define visual patterns for improving the extraction (patterns) and train the extractors (samples tab). These last two tabs are still experimental facilities, so an end user would not need to use them.

A.2.3 Web service interface

The web service interface provides a web service interface that mimics the admin web interface but for read-only operations.

```

scrappy -s
Launching Scrappy Web Server...
== Sinatra/1.1.3 has taken the stage on 3434 for production with backup from Thin

```

This service does not provide a front page, so you need to make a GET petition to a specific url, following this syntax: `http://[scrappyhost:port]/[format]/[url]`. For example, to get rdf data out of example.com, launching scrappy in localhost with the default port, it would be: `http://localhost:3434/rdf/example.com`.



Figure A.2: Web admin interface of Scrappy

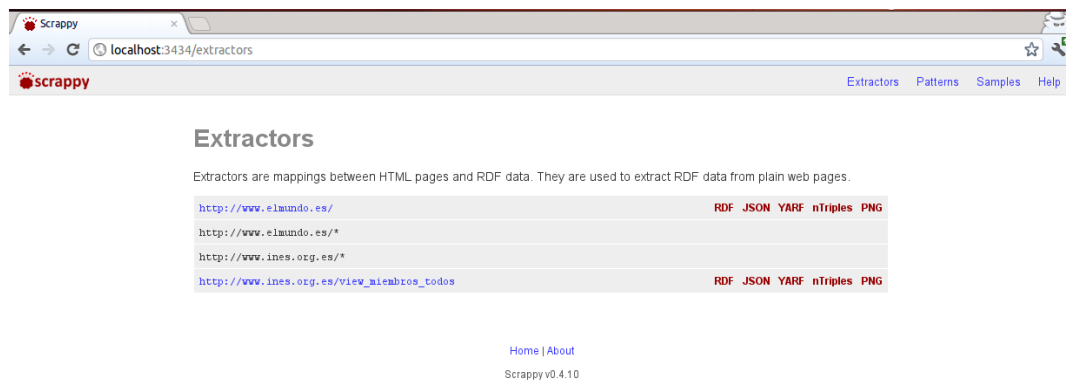


Figure A.3: Extractors Admin Interface

A.2.4 Ruby interface

Scrappy can be used in a Ruby program by requiring the gem.

```
require 'rubygems'
require 'scrappy'
# Parse a knowledge base
kb = RDF::Parser.parse :yarf,
open("https://raw.githubusercontent.com/gsi-upm/scrappy/omr/extractors/programmableweb.yarf").
  re
ad
# Set kb as default knowledge base
Scrappy::Agent::Options.kb = kb
# Create an agent
agent = Scrappy::Agent.new
# Get RDF output
output = agent.request :method=>:get, :uri=>'http://www.programmableweb.com'
# Output all titles from the web page
```

```
titles = output.find([], Node('dc:title'), nil)
titles.each { |title| puts title }
```

For example, to extract the data from <http://www.programmableweb.com>, you will need:

```
require 'rubygems'
require 'scrappy'
# Parse a knowledge base
kb = RDF::Parser.parse :yarf,
open("HOME_DIR/.scrappy/extractors/programmableweb.yarf").read
# Set kb as default knowledge base
Scrappy::Agent::Options.kb = kb
# Create an agent
agent = Scrappy::Agent.new
# Get RDF output
output = agent.request :method=>:get, :uri=>'http://www.programmableweb.com'
# This will return a graph object. To get the ntriples
# ntriples
ntriples = output.serialize :ntriples
# rdf
rdf = output.serialize :rdf
```

A.2.5 Integration with Sesame

While using OMR, several modifications have been introduced into scrappy, in order to user both Sesame and OMR. The new configuration file includes the options to connect to both platforms:

```
# The host where omr is. Do not add the trailing '/'
omr_host: https://vsr-web.informatik.tu-chemnitz.de/
omr_complete: omr-write/components
omr_user: omr-user
omr_pass: omr-pass
omr_port: 443
# The time to consider the data in the repository valid, in minutes
time: 5
# The format to communicate with the repository
format: ntriples
# You can use any of the following formats:
# rdf, ntriples, turtle, n3, trix, trig
```

In general, it has the same values as a normal scrappy config file has, but, also, several “omr” values:

- complete: how to complete the url for the sesame server.
- omr_host: The host where omr is installed
- omr_complete: how to complete the url for omr.
- omr_user: the user to connect with omr
- omr_pass: The password for the given user
- omr_port: The port for the connection. Since omr uses https, the default port is 443

A.2.6 Extractors

Extractors are based on the Scraping ontology and define mappings between HTML content and RDF data. An example of extractor is shown in Listing A.1.

Listing A.1: Extractor example for Yahoo! Pipes

```
dc: http://purl.org/dc/elements/1.1/
owl: http://www.w3.org/2002/07/owl#
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs: http://www.w3.org/2000/01/rdf-schema#
sioc: http://rdfs.org/sioc/ns#
sc: http://lab.gsi.dit.upm.es/scraping.rdf#
vu: http://vulneranet.org/vulneranet.owl#
```

```

omelette: http://www.ict-omelette.eu/schema.rdf#
ctag: http://commontag.org/ns#
rosm: http://www.wsmo.org/ns/rosm/0.1#
hrests: http://www.wsmo.org/ns/hrests#

# Home page
*:
  rdf:type: sc:Fragment
  sc:selector:
    *:
      rdf:type: sc:UriSelector
      rdf:value: "http://pipes.yahoo.com/pipes/"
  sc:subfragment:
    *:
      rdf:type: sc:Fragment
      sc:type: sc:Index
      sc:selector:
        *:
          rdf:type: sc:CssSelector
          rdf:value: "a.navlink"
          sc:keyword: "browse"
      sc:identifier:
        *:
          rdf:type: sc:RootSelector
          sc:attribute: "href"

# Popular pipes page
*:
  rdf:type: sc:Fragment
  sc:selector:
    *:
      rdf:type: sc:UriSelector
      rdf:value: "http://pipes.yahoo.com/pipes/pipes.popular"
      "
  sc:subfragment:
    *:
      rdf:type: sc:Fragment

```

```
sc:type: sc:Page
sc:selector:
  *:
    rdf:type: sc:CssSelector
    rdf:value: "#mTagstoptags a"
sc:identifier:
  *:
    rdf:type: sc:NewUriSelector
    sc:prefix: "http://pipes.yahoo.com/pipes/search?r=
      tag:"
    sc:follow: "true"
    sc:downcase: "true"

# Any index page
*:
  rdf:type: sc:Fragment
  sc:selector:
    *:
      rdf:type: sc:UriSelector
      rdf:value:
        "http://pipes.yahoo.com/pipes/pipes.popular"
        "http://pipes.yahoo.com/pipes/search"
  sc:subfragment:
    *:
      rdf:type: sc:Fragment
      sc:type: omelette:Service
      sc:selector:
        *:
          rdf:type: sc:CssSelector
          rdf:value: "li.pipelistli"
      sc:identifier:
        *:
          rdf:type: sc:CssSelector
          rdf:value: "h3 a"
          sc:attribute: "href"
      sc:subfragment:
        *:

```



```

        rdf:type: sc:Fragment
        sc:relation: omelette:dataFormat
        sc:type: rdf:Literal
        sc:selector:
            *:
                rdf:type: sc:CssSelector
                rdf:value: "li.first"
                sc:keyword: "formats:"
                sc:selector:
                    *:
                        rdf:type: sc:XPathSelector
                        rdf:value: "../li[@class='tag']/a"
*:
    rdf:type: sc:Fragment
    sc:type: sc:Page
    sc:selector:
        *:
            rdf:type: sc:CssSelector
            rdf:value: ".paginate a"
            sc:keyword: "next >"
    sc:identifier:
        *:
            rdf:type: sc:RootSelector
            sc:attribute: "href"

# Yahoo Pipe
*:
    rdf:type: sc:Fragment
    sc:type: omelette:Service
    sc:selector:
        *:
            rdf:type: sc:UriPatternSelector
            rdf:value: "http://pipes.yahoo.com/pipes/pipe.info?_id
                =*"
    sc:identifier:
        *:
            rdf:type: sc:BaseUriSelector

```

```
sc:subfragment:
  *:
    rdf:type: sc:Fragment
    sc:relation: rdfs:label
    sc:type: rdf:Literal
    sc:selector:
      *:
        rdf:type: sc:CssSelector
        rdf:value: "h1.title"
  *:
    rdf:type: sc:Fragment
    sc:relation: dc:description
    sc:type: rdf:Literal
    sc:selector:
      *:
        rdf:type: sc:CssSelector
        rdf:value: ".bd .desc"
  *:
    rdf:type: sc:Fragment
    sc:relation: ctag:tagged
    sc:type: ctag:Tag
    sc:selector:
      *:
        rdf:type: sc:CssSelector
        rdf:value: "#mTagstoptags a"
    sc:identifier:
      *:
        rdf:type: sc:NewUriSelector
        sc:prefix: "http://www.ict-omelette.eu/omr/tags/"
        sc:downcase: "true"
    sc:subfragment:
      *:
        rdf:type: sc:Fragment
        sc:type: rdf:Literal
        sc:relation: rdfs:label
        sc:selector:
```

```

        *:
            rdf:type: sc:RootSelector

*:
    rdf:type: sc:Fragment
    sc:relation: omelette:uses
    sc:selector:
        *:
            rdf:type: sc:CssSelector
            rdf:value: "#mSourcestoptags a"
        sc:identifier:
            *:
                rdf:type: sc:NewUriSelector
                sc:prefix: "http://"
                sc:downcase: "true"

*:
    rdf:type: sc:Fragment
    sc:relation: omelette:uses
    sc:type: omelette:Service
    sc:selector:
        *:
            rdf:type: sc:CssSelector
            rdf:value: "#mModulestoptags a"
        sc:identifier:
            *:
                rdf:type: sc:NewUriSelector
                sc:prefix: "http://www.ict-omelette.eu/omr/
                    operator/"
                sc:downcase: "true"
        sc:subfragment:
            *:
                rdf:type: sc:Fragment
                sc:type: rdf:Literal
                sc:relation: rdfs:label
                sc:selector:
                    *:

```

```
        rdf:type: sc:RootSelector
*:
  rdf:type: sc:Fragment
  sc:relation:
    omelette:endpoint
    dc:source
  sc:selector:
    *:
      rdf:type: sc:RootSelector
  sc:identifier:
    *:
      rdf:type: sc:BaseUriSelector

*:
  rdf:type: sc:Fragment
  sc:relation: omelette:describedBy
  sc:type: rosm:Service
  sc:selector:
    *:
      rdf:type: sc:RootSelector
  sc:subfragment:
    *:
      rdf:type: sc:Fragment
      sc:relation: rosm:supportsOperation
      sc:type: rosm:Operation
      sc:selector:
        *:
          rdf:type: sc:RootSelector
      sc:subfragment:
        *:
          rdf:type: sc:Fragment
          sc:relation: hrests:hasAddress
          sc:selector:
            *:
              rdf:type: sc:BaseUriSelector
          sc:identifier:
            *:

```

```

        rdf:type: sc:NewUriSelector
        sc:suffix: "&_render=rss"
*:
    rdf:type: sc:Fragment
    sc:relation: rosm:requestURIParameter
    sc:selector:
        *:
            rdf:type: sc:CssSelector
            rdf:value: "#runform table input[@type!='
                        submit']"
    sc:subfragment:
        *:
            rdf:type: sc:Fragment
            sc:relation: rdfs:label
            sc:type: rdf:Literal
            sc:selector:
                *:
                    rdf:type: sc:RootSelector
                    sc:attribute: "name"
        *:
            rdf:type: sc:Fragment
            sc:relation: ctag:tagged
            sc:selector:
                *:
                    rdf:type: sc:RootSelector
                    sc:attribute: "type"
    sc:subfragment:
        *:
            rdf:type: sc:Fragment
            sc:type: rdf:Literal
            sc:relation: rdfs:label
            sc:selector:
                *:
                    rdf:type: sc:RootSelector
                    sc:attribute: "type"

```

elector

```
rdf:value: "a"
```

OMELETTE Mashup Registry (OMR)

B.1 Installation of Sesame with uSeekM (+PostgreSQL +PostGIS)

Requirements:

- A Java Servlet Container that supports Java Servlet API 2.4 and Java Server Pages (JSP) 2.0, or newer. We recommend using a recent, stable version of Apache Tomcat. (<http://tomcat.apache.org/>)
- A recent, stable version of PostgreSQL Server (32Bit). (<http://www.postgresql.org/download/windows/>)
- PostGIS extension for PostgreSQL. (Can be installed with the Stack Builder that comes with PostgreSQL.)
- Extended Sesame HTTP Server with Indexing - USeekM (<https://dev.opensahara.com/projects/useekm/wiki/HttpServer>) Installation steps:
- First install the PostgreSQL Server for Windows 32Bit (32Bit because the PostGIS

extension is not yet 100% compatible with the 64Bit server and thus is not available in the Stack Builder Installer).

- After installing the PostgreSQL Server start the Stack Builder and select PostGIS 1.x under "Spatial Extensions" and install it to the running PostgreSQL instance.
- Start the pgAdmin tool to create a database user and database for your index.
- Create a new Login-Role (useekm is used as name and password in the example) with superuser rights.
- Create a new database (useekm is used as name in the example). Select the created user as owner. Choose "template_postgis" as template (IMPORTANT!).
- Your database should be good to go.
- Install Apache Tomcat and get it running. There should be no configuration needed.
- Download the latest useekm-http-server and useekm-http-workbench *.war files from <https://dev.opensahara.com/nexus/content/repositories/releases/com/opensahara/>
- You may rename them to openrdf-sesame.war and openrdf-workbench.war if you want to replace your current sesame installation.
- Put them in the Tomcat webapps folder. They should automatically get deployed.
- Your extended sesame server should be good to go.
- Create a configuration file for your indexed repository like this.
- The example indexes the <http://purl.org/dc/elements/1.1/description> predicate.
- You may need to edit the database connection settings, like username, password and the URL with the database name.

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
<!-- The id "repository" is mandatory! -->
<bean id="repository" class="org.openrdf.repository.sail.SailRepository">
<constructor-arg>
<bean class="com.useekm.indexing.IndexingSail">
<constructor-arg ref="sail" />
<constructor-arg ref="indexerSettings" />
</bean>
</constructor-arg>
```



```
</bean>
<!-- This example uses the NativeStore as the underlying sail, you could also use
the
MemoryStore -->
<bean id="sail" class="org.openrdf.sail.nativerdf.NativeStore" />
<!-- Please customize the indexer settings: -->
<bean id="indexerSettings" lazy-init="true"
class="com.useekm.indexing.postgis.PostgisIndexerSettings">
<property name="defaultSearchConfig" value="simple" />
<property name="dataSource" ref="pgDatasource" />
<property name="matchers">
<list>
<bean class="com.useekm.indexing.postgis.PostgisIndexMatcher">
<property name="predicate"
value="http://purl.org/dc/elements/1.1/description" />
<property name="searchConfig" value="simple" />
</bean>
</list>
</property>
<property name="partitions">
<list>
<bean class="com.useekm.indexing.postgis.PartitionDef">
<property name="name" value="description" />
<property name="predicates">
<list>
<value>http://purl.org/dc/elements/1.1/descr
iption</value>
</list>
</property>
</bean>
</list>
</property>
<!-- You can add additional configuration, such as index partitions to optimize
performance. See the documentation. -->
</bean>
<bean id="pgDatasource" lazy-init="true"
class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
<property name="driverClassName" value="org.postgresql.Driver"/>
<property name="url" value="jdbc:postgresql://localhost:5432/useekm"/> <!--
CUSTOMIZE! -->
<property name="username" value="useekm"/>
<!--
CUSTOMIZE! -->
<property name="password" value="useekm"/>
<!--
CUSTOMIZE! -->
</bean>
</beans>
```

- Save this as configuration.xml to a folder in which the webapps have access to.

- Open the URL to the workbench <http://localhost:8080/useekm-http-workbench>).
- Enter the URL to your Sesame server. (ex.: <http://localhost:8080/useekm-http-server>).
- Create a new repository.
- Choose USeekM Store as Type.
- Choose a name and ID.
- Press Next and enter the full path to the configuration file mentioned above. (use slashes instead of backslashes!).
- Press create and the repository and the tables in the database should get created and initialised.

Now you have the prerequisites to install the DataGridService for hosting your own OMELETTE Mashup Registry.

B.2 Installation of the DataGridService

We created an installable package for Windows to run a stand-alone DataGridService with all current features of the OMR and a RESTClient with a GUI (<https://vsr.informatik.tu-chemnitz.de/demo/omr/omrsetup.zip>). Just install the package and two folders will be created in the selected folder - one with the RESTClient and one with the DGS. Before starting the DataGridService you may need to configure it to connect to the correct Sesame triple store and repository. In the file “Server DgsTestServer.exe.config” please set the following values corresponding to these you set up earlier during the installation of Sesame and the uSeekM repository:

```
<appSettings>
  ...
  <!-- Service Registry -->
  <add key="sesameStoreUrl" value="[Store URL,
i.e.: http://localhost:8080/useekm-http-server]/"/>
  <add key="sesameRepository" value="[repository]"/>
  <add key="sesameUsername" value="[username]" />
  <add key="sesamePassword" value="[password]" />
</appSettings>
```

After you have done this just run the DgsTestServer.exe with administrative privileges. Now your OMELETTE Mashup Registry is ready for usage.

B.3 User manual

The OMR provides a REST interface which is illustrated in this section. The following usage scenarios will help you get started. Create a new resource for storing semantic component descriptions:

```
POST /omr HTTP/1.1
Host: datagridservice.example.org
Content-Type: text/xml
Content-Length: xxx
<collection xmlns="http://www.w3.org/2007/app"
xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:dgs="http://www.webcomposition.net/2008/02/dgs/">
  <atom:title>components</atom:title>
  <dgs:dataspaceengines>
    <dgs:dataspaceengine
      dgs:type="http://www.webcomposition.net/2008/02/dgs/DataSpaceEngines/
        ServiceRegistr
        yDataSpaceEngine" />
    </dgs:dataspaceengine>
  </dgs:dataspaceengines>
</collection>
```

Add new data to the OMR:

```
POST /omr/components HTTP/1.1
Host: datagridservice.example.org
Content-Type: text/xml
Content-Length: xxx
<rdf:RDF
xml:base="https://datagridservice.example.org/omr/components/graphs/1705f351-db6e-4
037-899c-08156ab31e13" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:ns0="http://www.ict-omelette.eu/schema.rdf#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <ns0:Widget rdf:about="http://eco.netvibes.com/themes/368491/wasabi">
    <ns1:description xmlns:ns1="http://purl.org/dc/elements/1.1/">Official theme
      for
      Netvibes Wasabi</ns1:description>
    <ns2:source xmlns:ns2="http://purl.org/dc/elements/1.1/"
      rdf:resource="http://eco.netvibes.com/themes/368491/wasabi" />
    <ns3:isPartOf xmlns:ns3="http://purl.org/dc/terms/"
      rdf:resource="https://datagridservice.example.org/omr/components/graphs/1705
        f351-db6
        e-4037-899c-08156ab31e13" />
    <ns0:categorizedBy>Textures; Official</ns0:categorizedBy>
    <ns0:endpoint
      rdf:resource="http://widgets.opera.com/widget/download/force/10322/1.0/" />
    <ns0:hasRegistryEntry
      rdf:resource="http://datagridservice.example.org/omr/components/feedItem?
        name=https://datagridservice.example.org/omr/components/graphs/1705f351-db6e
        -4037-
```

```

899c-08156ab31e13" />
<ns4:installs xmlns:ns4="http://www.netvibes.com/#">29678</ns4:installs>
<ns5:regions xmlns:ns5="http://www.netvibes.com/#"></ns5:regions>
<rdfs:label>Wasabi</rdfs:label>
</ns0:Widget>
</rdf:RDF>

```

Get all available components registered in OMR:

```

GET /omr/components HTTP/1.1
Host: datagridservice.example.org
Accept: text/xml

<feed xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Available Components</title>
  <subtitle type="text">Component descriptions</subtitle>
  <id>http://datagridservice.example.org/omr/components</id>
  <updated>2012-05-30T15:42:26+02:00</updated>
  <link rel="meta" type="application/rdf+xml" title="Feed metadata"
href="http://datagridservice.example.org/omr/components/meta" />
  <link rel="views" type="application/atom+xml" title="Views on RDF dataset"
href="http://datagridservice.example.org/omr/components/views" />
  <link rel="sparql" type="application/sparql-results+xml" title="Sparql endpoint"
href="http://datagridservice.example.org/omr/components/sparql" />
  <link rel="sparql-update" type="application/sparql-results+xml" title="Sparql-
Update
endpoint" href="http://datagridservice.example.org/omr/components/sparql-update"
/>
  <link rel="data" type="application/rdf+xml" title="RDF dataset"
href="http://datagridservice.example.org/omr/components/graphs/all" />
  <link rel="search" type="application/opensearchdescription+xml" title="OpenSearch
endpoint"
href="http://datagridservice.example.org/omr/components/opensearch/document" />
  <entry>
    <id>https://vsr-web.informatik.tu-chemnitz.de/omr/components/graphs/bf5cdceb-38c7
-4a
0c-b5ed-49f1eae6bfa2</id>
    <title type="text">Wasabi</title>
    <summary type="text">Official theme for Netvibes Wasabi</summary>
    <published>2012-05-30T15:42:27+02:00</published>
    <updated>2012-05-30T15:42:27+02:00</updated>
    <link rel="edit-media" type="application/rdf+xml"
href="http://datagridservice.example.org/omr/components/graphs/?
name=https://vsr-web.informatik.tu-chemnitz.de/omr/components/graphs/bf5cdceb-38
c7-4
a0c-b5ed-49f1eae6bfa2" />
    <link rel="edit" type="application/atom+xml;type=entry;"
href="http://datagridservice.example.org/omr/components/feedItem?
name=https://vsr-web.informatik.tu-chemnitz.de/omr/components/graphs/bf5cdceb-38
c7-4
a0c-b5ed-49f1eae6bfa2" />
    <content type="application/rdf+xml"

```

```

src="http://datagridservice.example.org/omr/components/graphs/?
name=https://vsr-web.informatik.tu-chemnitz.de/omr/components/graphs/bf5cdceb-38
c7-4
a0c-b5ed-49f1eae6bfa2" />
</entry>
...
</feed>

```

Creation of example view for searching components by their description (SPARQL template):

```

POST /omr/components/views HTTP/1.1
Host: datagridservice.example.org
Content-Type: text/xml
Content-Length: xxx
<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title type="text">services</title>
  <content type="application/xml+vnd.omr">
    <omr:view xmlns:omr="http://www.ict-omelette.eu/schema.rdf#omr">
      <omr:url?query={query}></omr:url>
      <omr:sparql>
        <![CDATA[
          PREFIX pdc:<http://purl.org/dc/elements/1.1/>
          PREFIX search:<http://rdf.opensahara.com/search#>
          SELECT DISTINCT ?result
          WHERE
          {
            ?result pdc:description ?description.
            FILTER search:text(?description, "{query}")
          }
        ]]>
      </omr:sparql>
    </omr:view>
  </content>
</entry>

```

Execute view to get all components matching a given query string:

```

GET /omr/components/views/services/data?query=geo HTTP/1.1
Host: datagridservice.example.org
Accept: application/sparql-results+xml
<?xml version="1.0" encoding="utf-8"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="result" />
  </head>
  <results>
    <result>
      <binding name="result">
        <uri>http://geoservice.example.org</uri>
      </binding>
    </result>
  </results>
</sparql>

```

APPENDIX B. OMELETTE MASHUP REGISTRY (OMR)

```
</result>  
</results>  
</sparql>
```

OMR Administrative interface

C.1 Installation and configuration

Requirements

- OMR or Sesame 2.0 for testing.
- Apache 2 web server with PHP 5.2

Extract all the content wherever it is preferred. Before executing it, some configuration must be done.

The administrative interface can be configured to use a semantic data source by means of an OMR REST interface or a general SPARQL endpoint. Within the project Omelette, it has been configured to use both the OMR interface or a Sesame Open-rdf Workbench, which has been used for testing purposes.

The semantic data source is configured in the `omr.php` class file. There are two kinds of URL to be configured.

1. To configure the URL of the SPARQL endpoint, the server variable must be config-

ured.

- OMR:

<https://vsr-web.informatik.tu-chemnitz.de/omr-write/components/sparql>

- Sesame:

<http://shannon.gsi.dit.upm.es:18080/openrdf-workbench/repositories/repository/query>

2. It is also necessary another URL where the data to be added to the repository is posted. This is defined by `post_url`

- OMR:

<https://vsr-web.informatik.tu-chemnitz.de/omr-write/components/>

- Sesame:

<http://shannon.gsi.dit.upm.es:18080/openrdf-workbench/repositories/repository/add>

```
/* OMR configuration */
var $user = "omr-client-upm";
var $password = "omr.client.upm.2012";
var $omr_post_url = "https://vsr-web.[...].tu-chemnitz.de/omr-write/components/";
var $omr_server = "https://vsr-web.[...].de/omr-write/components/sparql";
/* Sesame configuration */
var $repository = "repository_name";
var $sesame_post_url = "http://[...]/repositories/". $this->repository. "/add";
var $sesame_server = "[...]/repositories/". $this->repository. "/query";
```

Both can be configured at the same time. In order to switch between the endpoint to use, it is just needed to change the value of the variable `$endpoint`, and select “omr”, “sesame”

```
endpoint = "sesame"; //functions.omelette.php
```


OMELETTE Ranking System

Its code is available at <https://github.com/gsi-upm/omr-admin/tree/master/ranking>

Requirements:

- Sesame 2.6
- Apache Tomcat. (<http://tomcat.apache.org/>)

Installation steps:

- First download the ranking system's code available at project's repository.
- Once downloaded it, uncompress.
- Configure the ranking system. To this, the file “configuration.properties” available in root folder should be edited. In particular, fill the following properties.

Once the system has been configured, the Omelette ranking system is ready to be used.

- Finally, execute the script startRanking.sh:

```
$ ./startRanking.sh
```

After running the program, it will have been introduced the fields for the different ranking algorithms in the sesame's database: `limon:DegCent`, `limon:ClosCent` and `limon:RatSoc` correspondingly. These fields represents the distinct mechanisms that we have it to evaluate like service is important compared another.

- This information should be updated periodically by a timer.

Bibliography

- [1] O. Chudnovskyy, T. Nestler, M. Gaedke, F. Daniel, J. I. Fernández-Villamor, V. Chepegin, J. A. Fornas, S. Wilson, C. Kögler, and H. Chang, “End-user-oriented telco mashups: the omelette approach,” in *Proceedings of the 21st international conference companion on World Wide Web*, pp. 235–238, ACM, 2012.
- [2] C. A. Iglesias, J. I. Fernández-Villamor, D. Del Pozo, L. Garulli, and B. García, “Combining domain-driven design and mashups for service development,” in *Service Engineering*, pp. 171–200, Springer, 2011.
- [3] F. Daniel, F. Casati, B. Benatallah, and M.-C. Shan, “Hosted universal composition: Models, languages and infrastructure in mashart,” in *Conceptual Modeling-ER 2009*, pp. 428–443, Springer, 2009.
- [4] F. Villamor, C. A. Iglesias Fernandez, and M. Garijo Ayestaran, “Linked mashups ontology (limon) specification.” <http://www.gsi.dit.upm.es/ontologies/limon/>. Accessed April 4, 2012.
- [5] M. Zuccalà, “Soa4all in action: Enabling a web of billions of services,” in *Towards a Service-Based Internet* (E. Nitto and R. Yahyapour, eds.), vol. 6481 of *Lecture Notes in Computer Science*, pp. 227–228, Springer Berlin Heidelberg, 2010.
- [6] J. I. Fernández Villamor, J. Blasco Garcia, C. A. Iglesias Fernandez, and M. Garijo Ayestaran, “A semantic scraping model for web resources-applying linked data to web page screen scraping,” 2011.
- [7] “Scraping ontology <http://lab.gsi.dit.upm.es/scraping.rdf>.”
- [8] “Scrappy screen scraper <http://github.com/josei/scrappy>.”
- [9] T. Zemke, J. I. Fernández-Villamor, and C. A. Iglesias, “Ranking Web Services using Centralities and Social Indicators,” in *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012)*, (Wrocław, PL), SciTePress, 2012.

