ICT OMELETTE

Report

D5.2 - Initial discovery and composition report

Title	*D5.2 - Initial discovery and composition report*
Editor	*TUC*
Contributors	*UPM, TIE*
Reviewers	*T-SYSTEMS MMS, UNITN, UB*
Work Package	*WP5 – Automated discovery, selection and composition*
Task	*T5.2 – Automated discovery and lightweight semantic description of service components* *T5.3 – Automated mashup composition*
Date	*02th October, 2011*
Release	*1.3*
Status	*Final*
Distribution	*Public*

Theme ICT-2009.1.2
Service and Software Architecture, Infrastructure and Engineering

History

Release	Date	Remarks	Status	Distribution
0.1	14/07/11	Document outline	Draft	Confidential
0.2	14/07/11	OMR introduction, RDF model and samples of discovered services	Draft	Confidential
0.3	18/07/11	Discovery summary and RDF figure	Draft	Confidential
0.4	01/08/11	Initial automatic composition approach	Draft	Confidential
0.5	05/08/11	Initial OMELETTE Mashup Registry specification	Draft	Confidential
0.6	25/08/11	Framework architecture	Draft	Confidential
0.7	16/09/11	Refactored OMELETTE Mashup Registry Interface specification in accordance to latest advances of WP2 and WP3	Draft	Confidential
0.8	17/09/11	Testing approach specified	Draft	Confidential
0.9	18/09/11	Review and editing if the overall document	Draft	Confidential
1.0	20/09/11	A copy ready for an internal review	Draft	Confidential
1.1	20/09/11	Reduced summary of discovery	Draft	Confidential
1.2	29/09/11	Comments editing	Draft	Confidential
1.3	02/10/11	Integrated internal comments	Final	Public

Index

1. Executive Summary

OMELETTE aims at researching on the development, management, governance, execution and conception of converged services with a specific focus on the telco domain. OMELETTE has created a sound model of mashups that follows the REST architectural style, as well as a standard specification of a mashup-containing platform that guarantees portability and interoperability among different vendors and versions. These concepts are be based on a solid theoretical model of mashup foundations and the specific requirements gained from the telco domain.

OMELETTE breaks the barriers between the Web, telecommunication and hybrid services, taking into account the convergence of IT/telecom/content systems. OMELETTE fosters as well the reuse of existing components and mashups, by using cutting-edge machine learning and web-clipping techniques for its automated service discovery functionalities. In turn OMELETTE Registry provides a consistent and standard way for searching and ranking services and mashups appropriate to the users' goals. OMELETTE develops an open platform for building convergent mashups for the telco domain and thus obeys emerging state-of-practice interfaces for Widgets platforms such as e.g. Apache Rave and provides integration bridges with them.

2. Introduction: global picture

This WP aims at delivering a part of the overall OMELETTE picture that will be responsible for discovery, (semi)automatic annotation and composition of services and mashups. Composition will be driven by user goals expressed in a restricted/boxed natural language.

Illustration 1 depicts the components of OMELETTE that are under investigation and development of WP5. For the global picture of OMELETTE architecture, please refer to the deliverables of WP2, e.g. D2.2.

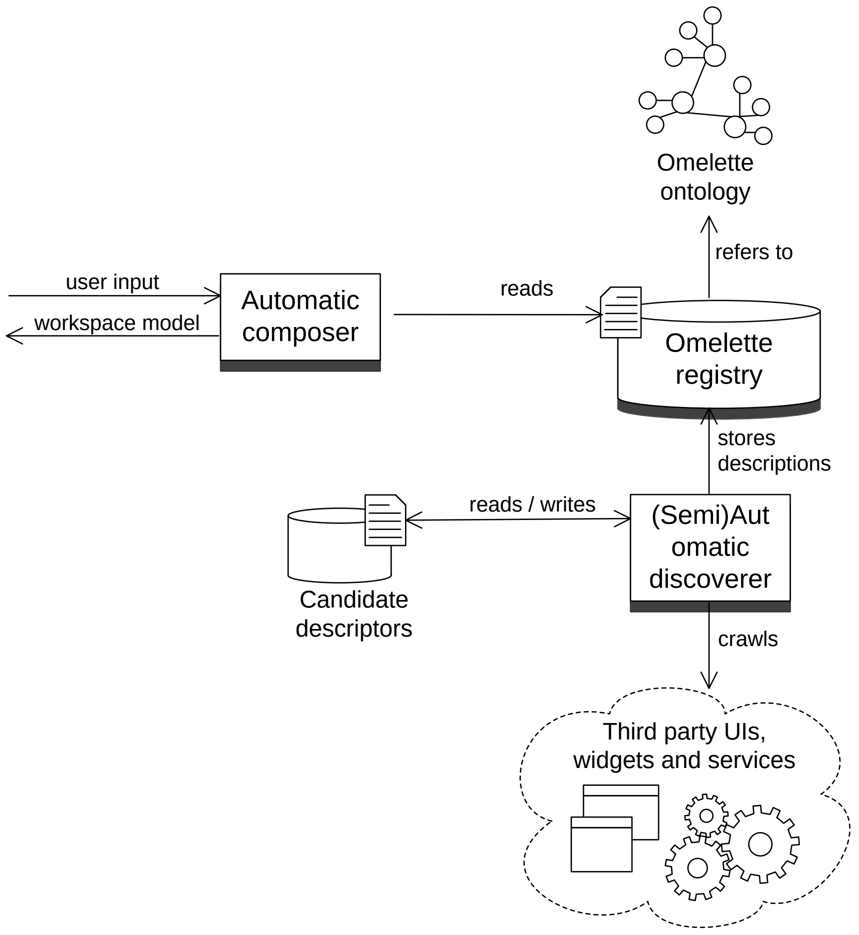


Illustration 1: Components of OMELETTE architecture that are the subjects of WP5

3. OMELETTE Mashup Registry

3.1. Introduction

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 mashups such as MyCocktail or mashArt 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 mashup composition.

However, developers face some difficulties when working on development of mashups. First, it is not easy for a developer to find the most appropriate services for a mashup 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 standardised descriptions, services need to be adapted in order to be used in a mashup platform.

The Omelette Mashup Registry (OMR) attempts to overcome these limitations. The registry 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.

3.2. RDF model

This section describes the architecture behind the 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. 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, 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. 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 Illustration 2.

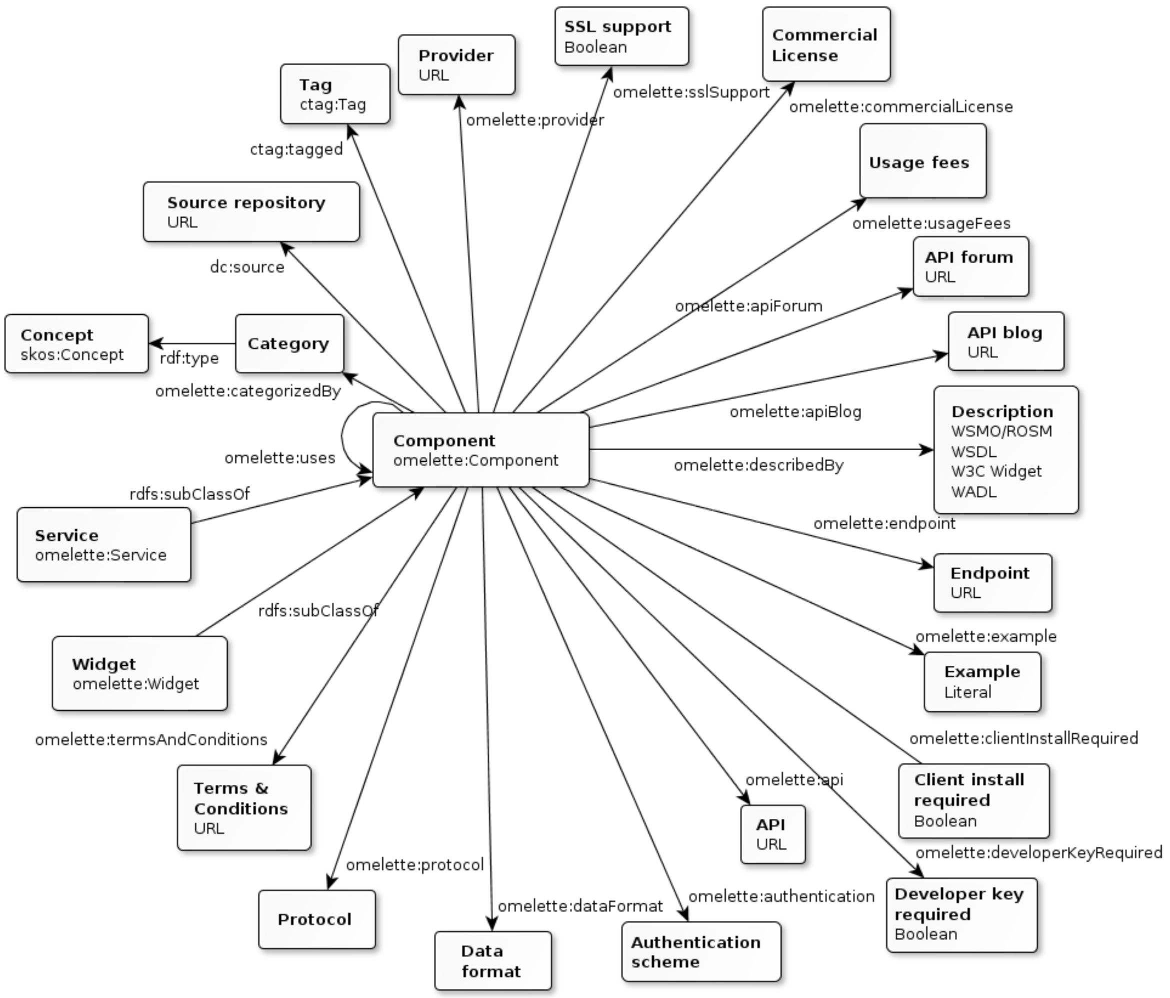


Illustration 2: Omelette Mashup Registry RDF model

This model covers aspects such as general categorization metadata, licensing or usage, and basic aspects of component execution. It reuses Simple Knowledge Organization System (SKOS[1]), Friend of a Friend (FOAF[2]), Dublin Core (DC[3]) and Common Tag (CTag[4]) 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 FP7 project[5] 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:

[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
[5] http://www.soa4all.eu/

```
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:

```
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 ]. }
```

Finally, another use case could consist of mashups about sport that make use of mapping services:

```
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  om:uses      [ rdf:type          om:Service;
                           om:categorizedBy [ rdfs:label  "Mapping" ] ];
            ctag:tagged  [ rdfs:label        "sports" ]. }
```

3.3. *OMELETTE Mashup Registry Interface*

The OMELETTE Mashup Registry (OMR) aims to support the discovery and composition process of heterogeneous mashup components within distributed service-oriented environments. The OMR is used to publish data feeds, Web services and mashups as linked data, so that clients can profit from the expressivity of component descriptions and can discover related items easily by exploring the data space. The basis of all OMR component descriptions is the RDF model as described in section 5.2. Its extensibility enables the definition of further component types and properties.

Following, we define functional requirements for the registry elicited from use cases of the deliverable 5.1.

3.3.1.Requirements

1. Registry should provide basic Create/Read/Update/Delete (CRUD) functionality for mashup component descriptions as presented in Illustration 2
2. Registry should provide multilingual free text search capabilities

3. Registry should facilitate the publishing process by extracting the metadata automatically out of the original component descriptions (e.g. W3C widget manifest files)
4. Registry should provide the following interfaces:
    a. Web-Interface for browsing and searching for component descriptions
    b. RESTful API for publishing and discovery of component descriptions
    c. SPARQL-endpoint for semantic queries on the complete data set
5. The write operations should be protected using a dedicated authentication and authorization mechanism
6. The registry should have necessary interfaces for integration with the available Widgets platforms, e.g. Apache Rave

We performed an analysis of the possible registry solutions and found out that only iServe [Ped10] and WebComposition/Data Grid Service [Chu10] satisfy most of the requirements above.

iServe is a platform for publishing Semantic Web Services as linked data, which has been developed within the Soa4All project. iServe enables users to publish service descriptions as linked data and perform search based on so called “high-level” APIs. The registry provides capability for free text search as well. iServe takes Web services described using Minimal Service Model into account and lacks support for further mashup components like e.g. widgets or data mashups. Furthermore, the CRUD operations on service descriptions are not supported sufficiently. For example, it is currently impossible to update a service description or extend the component model with further attributes. Finally, the discovery API is specified strictly and cannot be configured in a generic way to meet peculiarities of different mashup components.

The WebComposition/Data Grid Service is a lightweight Web service-based component targeting the content (data and semantic) perspective of Web applications (other perspectives are user experience and distributed system and architecture). The Data Grid Service provides a uniform REST/HTTP interface to extract, publish, annotate and share data scattered across multiple data sources. It is highly extensible due to the plugin infrastructure and can be easily adopted for new data sources. The built-in authentication and authorization mechanism enables fine-grained access control on the basis of WebID and WebAccessControl lists. A dedicated URL routing mechanism enables adaptation of service behaviour to concrete situational needs. However, free text search functionality is not available in the Data Grid Service.

From these two candidates we decided to extend WebComposition/Data Grid Service to satisfy the requirements above. First, the component is more flexible and adaptable because of the internal plugin architecture. Second, the reuse of its existing features will save more time and effort especially in the context of high-level APIs and security mechanisms. And finally, the team expertise in the application and development of Data Grid Service will enable more efficient, sustainable and goal-oriented development of OMR.

3.3.2. Extension of WebComposition/Data Grid Service

All resources within the Data Grid Service are identified using URIs, whereas some well-known patterns are used to access resource metadata ({resource URI}/meta), SPARQL endpoint ({service URI}/meta/sparql), access control lists ({resource URI}/meta/acl) etc. The standard HTTP methods GET, POST, PUT and DELETE are used to read, create, update and delete resources. Depending on the configuration, some of the resources may require authorization before executing the operations.

3.3.3.Planned extensions

Requirement	Extension
1.	A dedicated RDFDataSpaceEngine module will be implemented providing the basic CRUD functionality of component descriptions according to illustration 2. A single description will be represented as a serialization of an RDF graph, which can be accessed and manipulated by a dedicated URI. In addition, Atom[6] import and export filters will be implemented to simplify access to component descriptions.
2	To support a free-text search within component descriptions the RDFDataSpaceEngine module will be extended with Apache Lucene[7] or Apache SOLR[8] engines. The configured parts of the RDF graph will be indexed and optimized for free-text based search queries.
3	Dedicated modules will be implemented to import different types of mashup component descriptions. The import algorithms will extract semantic metadata from the incoming documents and convert it to the component model from illustration 2.
4	a) A dedicated Web interface to the WebComposition/Data Grid Service will be implemented to enable browsing and maintenance of the registry b) RESTful API will be implemented within the RDFDataSpaceEngine module c) SPARQL engine is already implemented within the core of the Data Grid Service. Another extension, which aims to enable simple access to high-level discovery API, is the so called *SPARQL templates*. By mapping URL templates to SPARQL queries clients can split the RDF graph into semantically related subgraphs and access them using pre-defined URIs. This makes the service interface both simple and powerful and supports clients with different needs. By providing SPARQL-defined views on RDF graph, clients can define e.g. categories or groups based on semantic component properties.
5	Currently, resources within Data Grid Service can be secured using the WebId [W3C11] and WebAccessControl [WAC11] mechanisms. Alternative authentication methods (like HTTP Basic [Fra99] or WS-Federation [Hai07]) can be implemented to support different types of clients.
6	To integrate the registry into the Mashup Delivery Platform we will implement a dedicated Apache Rave adapter, which implements the dedicated WidgetRepository interface[9]. The adapter will be

[6] http://tools.ietf.org/html/rfc4287
[7] http://lucene.apache.org/java/docs/index.html
[8] http://lucene.apache.org/solr/
[9] https://svn.apache.org/repos/asf/incubator/rave/trunk/rave-portal/src/main/java/org/apache/rave/portal/repository/WidgetRepository.java

	able to query and process data from remote registries.

3.3.4.Usage examples

Following, we describe in a scenario, how new component descriptions can be imported, discovered and retrieved within the OMELETTE mashup registry.

The OMELETTE team assumes that the RDFDataSpaceEngine module has been configured within the Data Grid Service and is available under http://datagridservice.example.org/components.

All component descriptions registered in the Data Grid Service can be requested using the RESTful API. The Content-Type header determines the incoming and outcoming representation format:

```
GET /components HTTP/1.1
Host: datagridservice.example.org
Content-Type: application/atom+xml

<feed xmlns='http://www.w3.org/2005/Atom'
       xmlns:wdg="http://www.w3.org/ns/widgets/"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:om="http://www.ict-omelette.eu/schema.rdf#">
  <id>http://datagridservice.example.org/components</id>
  <link rel='self' href='http://datagridservice.example.org/components'/>
  <title>OMR entries</title>
  <updated>2011-09-10T13:00:09.122Z</updated>
  <entry>
    <title>Chat Widget</title>
    <rdf:type rdf:resource="http://www.ict-omelette.eu/schema.rdf#Widget" />
    <link rel='alternate'
          href='http://datagridservice.example.org/components/dgsc549c8ff-7576-475d-a43a-
09cc97ea1e90'/>
    <id>http://datagridservice.example.org/components/dgsc549c8ff-7576-475d-a43a-
09cc97ea1e90</id>
    <author>Max Mustermann</author>
    <updated>2011-09-01T13:00:09.122Z</updated>
    <content xml:lang="en">This widget enables...</content>
    <wdg:height>300</wdg:height>
    <wdg:width>400</wdg:width>
    <om:commercialLicense>
    The source code is licensed under the Apache 2 license...
    </om:commercialLicense>
    ...
  </entry>
  <entry>
    <title>SMS Service</title>
       <rdf:type rdf:resource="http://www.ict-omelette.eu/schema.rdf#Widget" />
    <link rel='alternate'
          href='http://datagridservice.example.org/components/dgs2f88394c-6a47-4de0-972d-
44cd08ff1c27'/>
    <id>http://datagridservice.example.org/components/dgs2f88394c-6a47-4de0-972d-
44cd08ff1c27</id>
    <author>Max Mustermann</author>
    <updated>2011-09-01T13:00:09.122Z</updated>
    <content xml:lang="en">Enables clients to send SMS...</content>
    <om:usageFees>The usage fees are: 10 Ct/SMS,...</om:usageFees>
    ...
  </entry>
  ...
</feed>
```

Alternatively, *application/rdf+xml* can be used to retrieve data in RDF/XML format. By performing corresponding POST, PUT and DELETE requests on the URI above, new descriptions can be added, modified or deleted. The search API will provide free-text search capabilities in different languages.

```
GET /components?content=telefonkonferenz&lang=de HTTP/1.1
Host: datagridservice.example.org
Content-Type: application/atom+xml

<feed xmlns='http://www.w3.org/2005/Atom'
       xmlns:wdg="http://www.w3.org/ns/widgets/"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:om="http://www.ict-omelette.eu/schema.rdf#">
  <id>http://datagridservice.example.org/components?content=telefonkonferenz&lang=de</id>
  <link rel='self' href='http://datagridservice.example.org/components/'/>
  <title>Search results</title>
  <updated>2011-09-10T13:00:09.122Z</updated>
  <entry>
    <title>Chat Widget</title>
    <rdf:type rdf:resource="http://www.ict-omelette.eu/schema.rdf#Widget" />
    <link rel='alternate'
          href='http://datagridservice.example.org/components/dgsc549c8ff-7576-475d-a43a-
09cc97ea1e90'/>
    <id>http://datagridservice.example.org/components/dgsc549c8ff-7576-475d-a43a-
09cc97ea1e90</id>
    <author>Max Mustermann</author>
    <updated>2011-09-01T13:00:09.122Z</updated>
    <content xml:lang="en">This widget enables...</content>
    <wdg:height>300</wdg:height>
    <wdg:width>400</wdg:width>
    <om:commercialLicense>
    The source code is licensed under the Apache 2 license...
    </om:commercialLicense>
    ...
  </entry>
  ...
</feed>
```

W3C widget manifest files can be imported into the registry by uploading corresponding widgets or only manifest files to the Data Grid Service. The data will be automatically extracted and published using RDF:

```
POST /components/w3cwidgets HTTP/1.1
Host: datagridservice.example.org
Content-Type: application/widget

…*.wgt file…

HTTP/1.1 201 Created
Location: https://datagridservice.example.org/components/w3cwidgets/dgsb558e312-
6fcf-4f5c-974a-b5b17aeada73
```

```
GET /components/w3cwidgets/dgsb558e312-6fcf-4f5c-974a-b5b17aeada73 HTTP/1.1
Host: datagridservice.example.org

<rdf:Description
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:wdg="http://www.w3.org/ns/widgets/"
xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns:om="http://www.ict-omelette.eu/schema.rdf#"
       rdf:about="https://datagridservice.example.org/components/w3cwidgets/dgsb55
8e312-6fcf-4f5c-974a-b5b17aeada73"
>
       <rdf:type rdf:resource="http://www.ict-omelette.eu/schema.rdf#Widget" />
       <dc:title>HelloWorldWidget</dc:title>
       <dc:description>This is my first widget</dc:description>
       <dc:creator>Max Mustermann</dc:creator>
       <dc:language>en-us</dc:language>
       <wdg:height>300</wdg:height>
       <wdg:width>400</wdg:width>
       <wdg:viewport>windowed floating</wdg:viewport>
       <wdg:icon
rdf:resource="https://datagridservice.example.org/components/w3cwidgets/
dgsb558e312-6fcf-4f5c-974a-b5b17aeada73/meta/icon.ico" />
       <wdg:hasFeature>
              <rdf:Description>
                     <rdf:type rdf:resource="http://example.org/api/geolocation" />
                     <wdg:required>false</wdg:required>
              </rdf:Description>
       </wdg:hasFeature>
       <om:commercialLicense>This is my license</om:commercialLicense>
       <om:describedBy rdf:resource="https://datagridservice.example.org/ compo-
nents/w3cwidgets/dgsb558e312-6fcf-4f5c-974a-b5b17aeada73/meta/config.xml" />
       <om:version>1.0 Beta</om:version>
</rdf:Description>
```

The high-level API can be configured in run-time by defining the aforementioned SPARQL templates. The following example shows how SPARQL templates can be defined:

```
POST /components/meta/sparql-templates HTTP/1.1
Host: datagridservice.example.org
Content-type: text/n3

@prefix meta: <http://www.webcomposition.net/2008/02/dgs/meta/>.
[
meta:url "/w3cwidgets/{category}-widgets";
meta:sparql "
       PREFIX wgt: <http://www.w3.org/ns/widgets/>
       PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
       SELECT ?widget
       WHERE {
         ?widget rdf:type <http://www.ict-omelette.eu/schema.rdf#Widget>;
                wgt:hasFeature ?y.
         ?y rdf:type <http://example.org/api/{category}>.
       }
"
].

HTTP/1.1 201 Created
Location: https://datagridservice.example.org/components/meta/sparql-
templates/dgsbcff2e89-c552-45e7-875d-59d0e7ff30f2
```

Subsequently widgets can be discovered by their functional classification:

```
GET  /components/geolocation-widgets HTTP/1.1
Host: datagridservice.example.org

HTTP/1.1 200 OK
Content-type: application/sparql-results+xml

<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="widget" />
  </head>
  <results>
    <result>
      <binding name="widget">
<uri>https://datagridservice.example.org/components/w3cwidgets/dgsb558e312-6fcf-4f5c-974a-b5b17aeada73</uri>
      </binding>
    </result>
  </results>
</sparql>
```

Arbitrary SPARQL queries can be performed using the Data Grid Service SPARQL endpoint, which is accessible under *{service URI}/meta/sparql URI*.

4. Automated Discovery

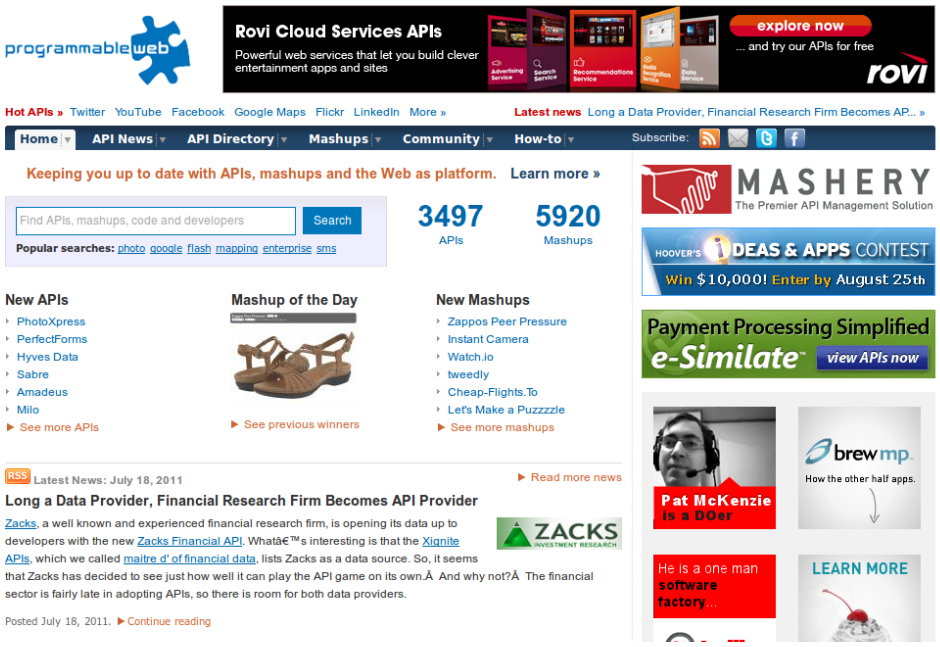
Automated discovery is one of the tasks present in the description of work inside WP5. Its objective is to enable OMELETTE users to access a wide amount of web components (i.e. both widgets and services) inside the OMR. Thanks to the automated discovery capabilities OMELETTE users will be able to use services and widgets as soon as they are published on external repositories.

4.1. Introduction

Three repositories were mined for services, widgets and mashups at this stage of the project, namely Programmable Web[10], Opera Widgets[11] and Yahoo Pipes[12].

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

[10] http://www.programmableweb.com/
[11] http://widgets.opera.com/
[12] http://widgets.opera.com/


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:

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

MASHERY The Premier API Management Solution
HOOVER'S IDEAS & APPS CONTEST Win $10,000! Enter by August 25th
Payment Processing Simplified e-Simlate view APIs now

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

Pat McKenzie is a DOer
He is a one man software factory...
brew mp. How the other half apps.
LEARN MORE

RSS Latest News: July 18, 2011
Read more news

Long a Data Provider, Financial Research Firm Becomes API Provider

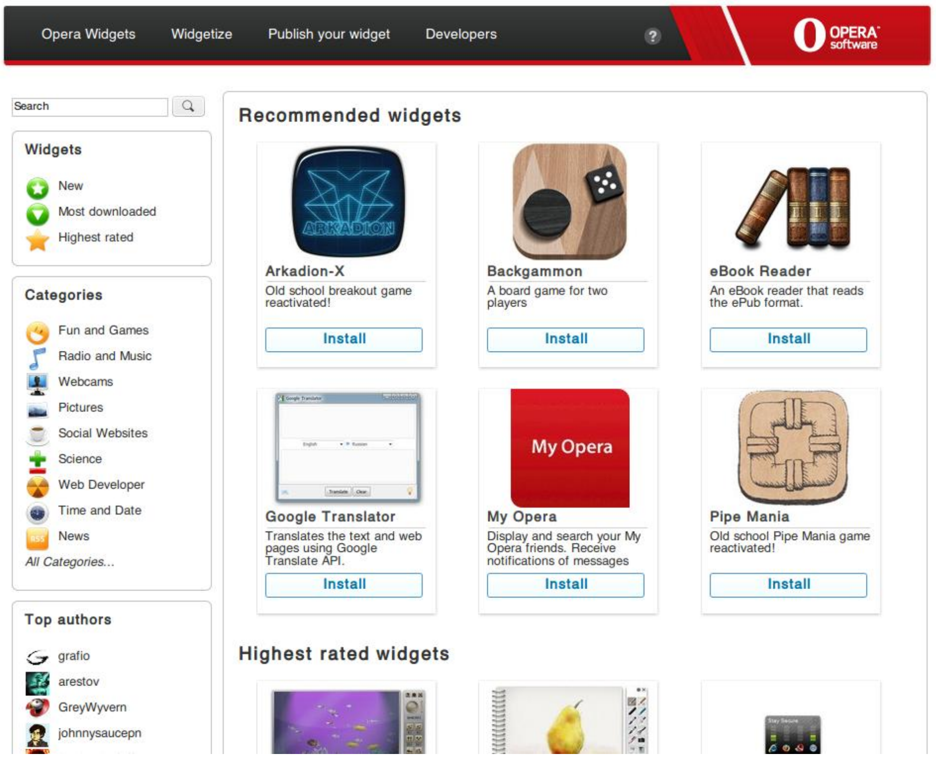
Zacks, a well known and experienced financial research firm, is opening its data up to developers with the new Zacks Financial API. Whatâ€™s interesting is that the Xignite APIs, which we called maitre d' of financial data, lists Zacks as a data source. So, it seems that Zacks has decided to see just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

ZACKS INVESTMENT RESEARCH

Posted July 18, 2011. Continue reading

Illustration 3: Programmable Web

Opera Widgets, shown in Illustration 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.


Opera Widgets | Widgetize | Publish your widget | Developers | ? | OPERA software

Search

Widgets
- New
- Most downloaded
- Highest rated

Categories
- Fun and Games
- Radio and Music
- Webcams
- Pictures
- Social Websites
- Science
- Web Developer
- Time and Date
- News
- All Categories...

Top authors
- grafio
- arestov
- GreyWyvern
- johnnysaucepn

Recommended widgets

Arkadion-X
Old school breakout game reactivated!
Install

Backgammon
A board game for two players
Install

eBook Reader
An eBook reader that reads the ePub format.
Install

Google Translator
Translates the text and web pages using Google Translate API.
Install

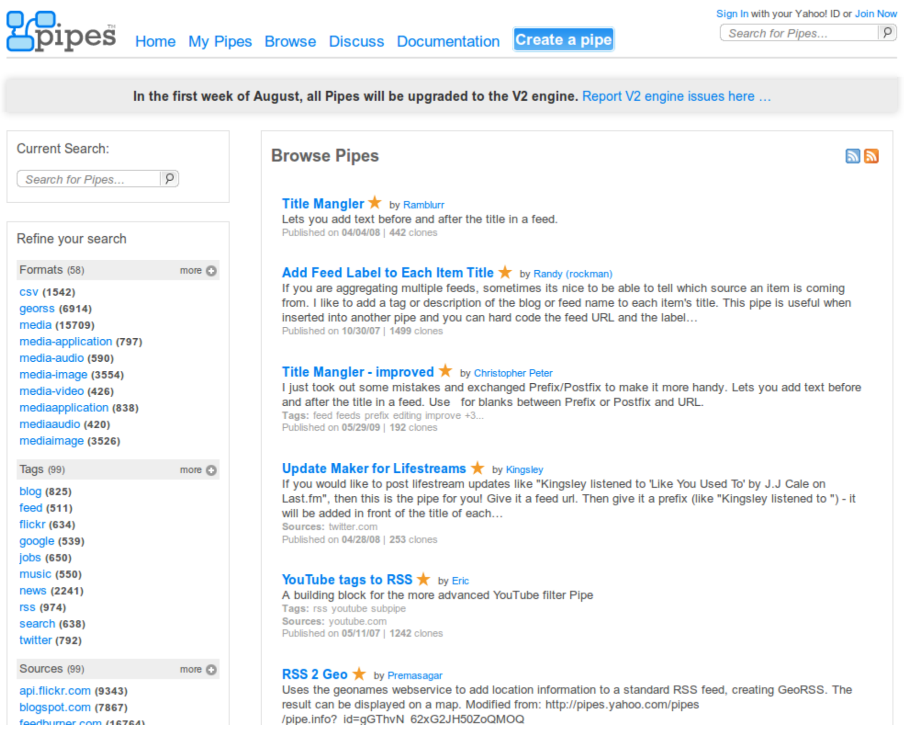
My Opera
Display and search your My Opera friends. Receive notifications of messages
Install

Pipe Mania
Old school Pipe Mania game reactivated!
Install

Highest rated widgets

Illustration 4: Opera Widgets

Yahoo Pipes, shown in Illustration 5, is a mashup 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 mashups, are categorized by tags, data format, sources, and also include short textual descriptions.



In the first week of August, all Pipes will be upgraded to the V2 engine. Report V2 engine issues here ...

Refine your search

Formats (58)
- 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)
- blog (825)
- feed (511)
- flickr (634)
- google (539)
- jobs (650)
- music (550)
- news (2241)
- rss (974)
- search (638)
- twitter (792)

Sources (99)
- api.flickr.com (9343)
- blogspot.com (7867)
- feedburner.com (16764)

Browse Pipes

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

Illustration 5: Yahoo Pipes

4.2. Discovery techniques

In order to mine the mentioned websites, a common approach was employed to extract the data. As part of the World Wide Web, the three repositories show a RESTful architecture, where a set of interlinked resources are published with resource specific descriptions. The format of the returned representations of the web resources is not standard, as they do not use any form of semantic annotations on top of the HTML data.

To map the HTML representations of the web resources available in these repositories to the RDF model defined for the OMELETTE Mashup Registry, the Scraping Ontology [Fer11] [ScOnt11] has be used by the Open Source screen scraper called Scrappy [Scr11], both developed in the context of OMELETTE.

Illustration 6 shows an example of mapping out of an unstructured HTML document into an OMELETTE RDF graph. In that example, a set of fragments in the HTML page are described along with the RDF data they represent. After processing that information on a particular sample HTML document, a scraper can produce the resulting RDF graph.

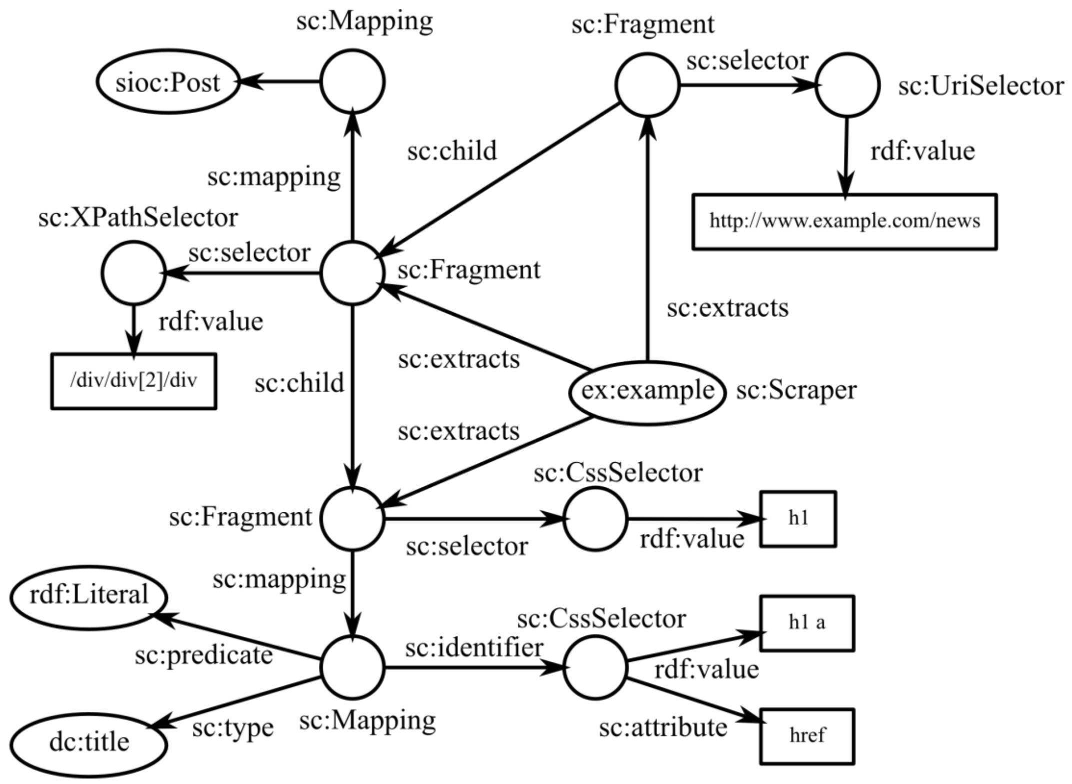


Illustration 6: Mapping example for data extraction

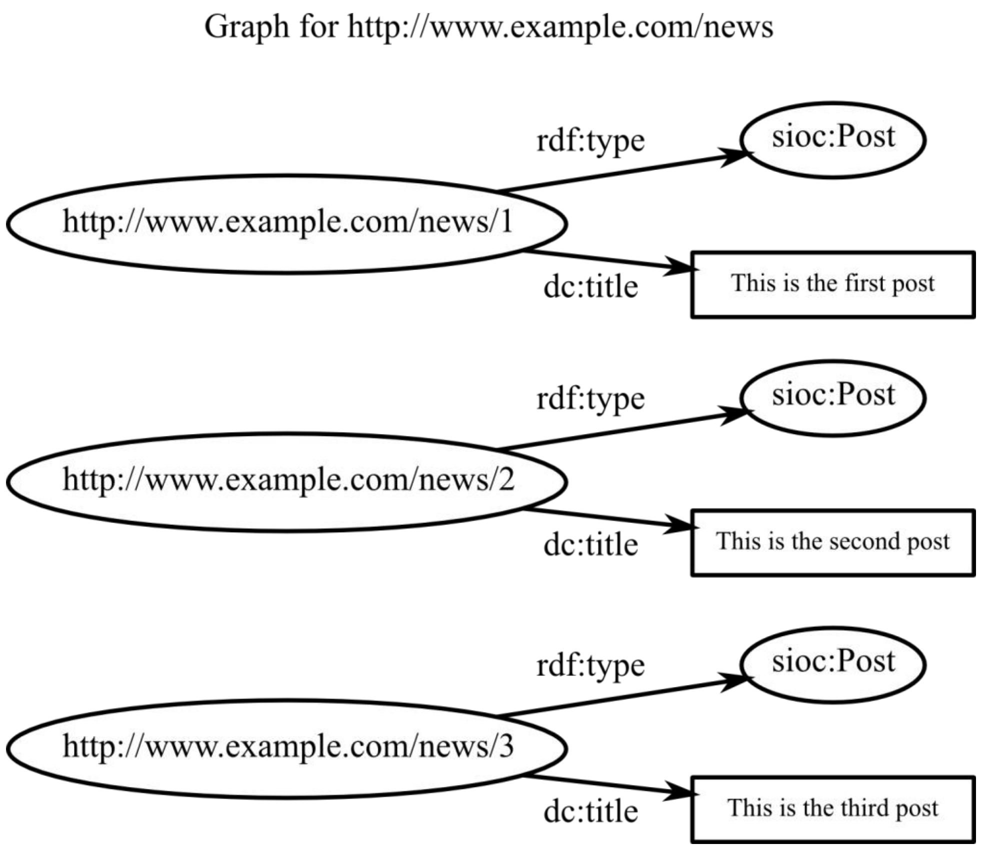


Illustration 7: Mapping example for data extraction

The methodology followed for the discovery assumes that first of all it is necessary to define a set of mappings for each of the repositories. These mappings state what data could be extracted, and how it could be done. Then prepared mappings are used by Scrappy to 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 mashup 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.

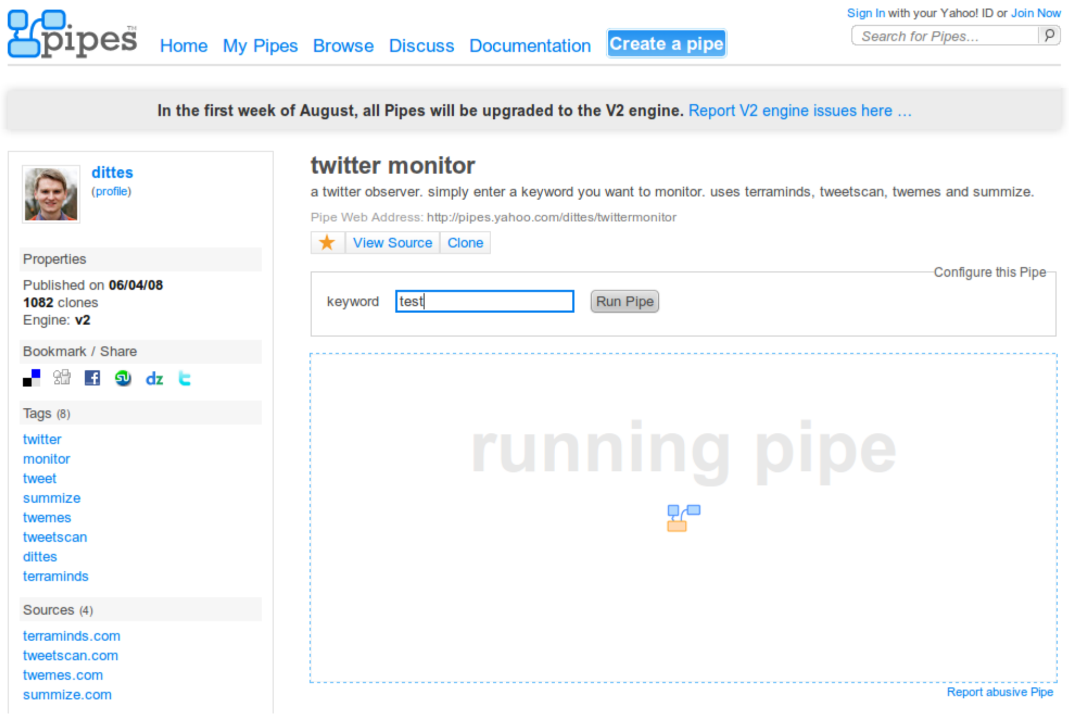
Similarly, for Opera Widgets each web resource represents a widget, so the information about the widget is mapped to the terms from the ontology. Also, its widget package (which uses WGT extension[13]) is mapped as well as the widget's endpoint.

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 mashup's webpage, as shown in Illustration 8.

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. An example of ROSM description, extracted from the pipe shown in the Illustration 6, which accepts a set of textual keywords on a URL, is shown next:

```
<rosm:Service>
  <rosm:supportsOperation>
    <rosm:Operation>
      <hrests:hasAddress
rdf:resource="http://pipes.yahoo.com/pipes/pipe.info?_id=c328dbc82f8019c0097eb331820
5fa09&amp;_render=rss"/>
      <rosm:requestURIParameter>
        <rdf:Description>
          <ctag:tagged>
            <rdf:Description>
              <rdfs:label>text</rdfs:label>
            </rdf:Description>
          </ctag:tagged>
          <rdfs:label>keyword</rdfs:label>
        </rdf:Description>
      </rosm:requestURIParameter>
    </rosm:Operation>
  </rosm:supportsOperation>
</rosm:Service>
```

[13] http://www.w3.org/TR/2011/REC-widgets-20110927/



In the first week of August, all Pipes will be upgraded to the V2 engine. Report V2 engine issues here ...

dittes
(profile)

Properties
Published on 06/04/08
1082 clones
Engine: v2

Bookmark / Share

Tags (8)
twitter
monitor
tweet
summize
twemes
tweetscan
dittes
terraminds

Sources (4)
terraminds.com
tweetscan.com
twemes.com
summize.com

twitter monitor

a twitter observer. simply enter a keyword you want to monitor. uses terraminds, tweetscan, twemes and summize.

Pipe Web Address: http://pipes.yahoo.com/dittes/twittermonitor

Configure this Pipe
keyword: test

Illustration 8: Execution page of a Yahoo Pipe's mashup

4.3. Results and discussion

Next is shown a sample service extracted from Programmable Web:

```
<rdf:RDF
   xmlns:ctag="http://commontag.org/ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:omelette="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/mapping">
    <rdfs:label>mapping</rdfs:label>
  </ctag:Tag>
  <omelette:Service rdf:about="http://www.programmableweb.com/api/geocoder">
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/mapping"/>
    <dc:description>Geocoder.us is a public service providing free geocoding of ad-
dresses and intersections in the United States. Using the service you can find the
latitude &amp; longitude of any US address and much more. Geocoder.us offers four
different ways to access our web services: an XML-RPC interface, a SOAP interface, a
REST interface that returns an RDF/XML document, and a REST interface that returns a
plain text comma separated values result. The methods and return values are equiva-
lent across all three interfaces.</dc:description>
    <dc:source rdf:resource="http://www.programmableweb.com/api/geocoder"/>
    <omelette:api rdf:resource="http://geocoder.us/help/"/>
    <omelette:authentication>HTTP Basic Auth</omelette:authentication>
    <omelette:categorizedBy rdf:resource="http://www.ict-
omelette.eu/omr/categories/mapping"/>
    <omelette:clientInstallRequired>false</omelette:clientInstallRequired>
    <omelette:commercialLicense>true</omelette:commercialLicense>
    <omelette:communityApiKit>PHP, C#</omelette:communityApiKit>
    <omelette:dataFormat>RDF/XML</omelette:dataFormat>
    <omelette:dataFormat>csv</omelette:dataFormat>
    <omelette:dataFormat>text</omelette:dataFormat>
    <omelette:describedBy
rdf:resource="http://geocoder.us/dist/eg/clients/GeoCoderPHP.wsdl"/>
    <omelette:endpoint rdf:resource="http://rpc.geocoder.us/service"/>
    <omelette:example>GeocoderResult, GeocoderAdressResult</omelette:example>
    <omelette:protocol>REST</omelette:protocol>
    <omelette:protocol>SOAP</omelette:protocol>
    <omelette:protocol>XML-RPC</omelette:protocol>
    <omelette:provider rdf:resource="http://geocoder.us"/>
    <omelette:readOnlyWithoutLogin>false</omelette:readOnlyWithoutLogin>
    <omelette:sslSupport>false</omelette:sslSupport>
    <omelette:termsAndConditions rdf:resource="http://geocoder.us/terms.shtml"/>
    <rdfs:label>geocoder API</rdfs:label>
  </omelette:Service>
</rdf:RDF>
```

Sample mashup extracted from Programmable Web:

```
<rdf:RDF
   xmlns:ctag="http://commontag.org/ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:omelette="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#">
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/books">
    <rdfs:label>books</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/lists">
    <rdfs:label>lists</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/shopping">
    <rdfs:label>shopping</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/social">
    <rdfs:label>social</rdfs:label>
  </ctag:Tag>
  <omelette:Service rdf:about="http://www.programmableweb.com/mashup/22books">
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/books"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/lists"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/shopping"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/social"/>
    <dc:description>22books is dedicated to the creating, sharing, and viewing of
book lists. Create your free book list and see the unique user experience when you
add books to your list.</dc:description>
    <dc:source rdf:resource="http://www.programmableweb.com/mashup/22books"/>
    <omelette:endpoint rdf:resource="http://www.22books.com"/>
    <omelette:uses rdf:resource="http://www.programmableweb.com/api/amazon-
ecommerce"/>
    <rdfs:label>22books</rdfs:label>
  </omelette:Service>
</rdf:RDF>
```

Sample service extracted from Yahoo Pipes:

```
<rdf:RDF
   xmlns:ctag="http://commontag.org/ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:hrests="http://www.wsmo.org/ns/hrests#"
   xmlns:omelette="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:rosm="http://www.wsmo.org/ns/rosm/0.1#">
```

```
  <omelette:Service
rdf:about="http://pipes.yahoo.com/pipes/pipe.info?_id=c328dbc82f8019c0097eb3318205fa
09">
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/dittes"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/monitor"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/summize"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/terraminds"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/tweet"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/tweetscan"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/twemes"/>
    <ctag:tagged rdf:resource="http://www.ict-omelette.eu/omr/tags/twitter"/>
    <dc:description>a twitter observer. simply enter a keyword you want to monitor.

uses terraminds, tweetscan, twemes and summize.</dc:description>
    <dc:source
rdf:resource="http://pipes.yahoo.com/pipes/pipe.info?_id=c328dbc82f8019c0097eb331820
5fa09"/>
    <omelette:describedBy>
      <rosm:Service>
        <rosm:supportsOperation>
          <rosm:Operation>
            <hrests:hasAddress
rdf:resource="http://pipes.yahoo.com/pipes/pipe.info?_id=c328dbc82f8019c0097eb331820
5fa09&amp;_render=rss"/>
            <rosm:requestURIParameter>
              <rdf:Description>
                <ctag:tagged>
                  <rdf:Description>
                    <rdfs:label>text</rdfs:label>
                  </rdf:Description>
                </ctag:tagged>
                <rdfs:label>keyword</rdfs:label>
              </rdf:Description>
            </rosm:requestURIParameter>
          </rosm:Operation>
        </rosm:supportsOperation>
      </rosm:Service>
    </omelette:describedBy>
    <omelette:endpoint
rdf:resource="http://pipes.yahoo.com/pipes/pipe.info?_id=c328dbc82f8019c0097eb331820
5fa09"/>
    <omelette:uses rdf:resource="http://summize.com"/>
    <omelette:uses rdf:resource="http://terraminds.com"/>
    <omelette:uses rdf:resource="http://tweetscan.com"/>
    <omelette:uses rdf:resource="http://twemes.com"/>
    <omelette:uses rdf:resource="http://www.ict-omelette.eu/omr/operator/fetch"/>
    <omelette:uses rdf:resource="http://www.ict-omelette.eu/omr/operator/sort"/>
    <omelette:uses rdf:resource="http://www.ict-
omelette.eu/omr/operator/strconcat"/>
    <omelette:uses rdf:resource="http://www.ict-
omelette.eu/omr/operator/textinput"/>
    <omelette:uses rdf:resource="http://www.ict-omelette.eu/omr/operator/uniq"/>
    <omelette:uses rdf:resource="http://www.ict-
omelette.eu/omr/operator/urlbuilder"/>
    <rdfs:label>twitter monitor</rdfs:label>
  </omelette:Service>
  <omelette:Service rdf:about="http://www.ict-omelette.eu/omr/operator/fetch">
    <rdfs:label>fetch</rdfs:label>
  </omelette:Service>
  <omelette:Service rdf:about="http://www.ict-omelette.eu/omr/operator/sort">
    <rdfs:label>sort</rdfs:label>
  </omelette:Service>
  <omelette:Service rdf:about="http://www.ict-omelette.eu/omr/operator/strconcat">
    <rdfs:label>strconcat</rdfs:label>
  </omelette:Service>
  <omelette:Service rdf:about="http://www.ict-omelette.eu/omr/operator/textinput">
```

```
    <rdfs:label>textinput</rdfs:label>
  </omelette:Service>
  <omelette:Service rdf:about="http://www.ict-omelette.eu/omr/operator/uniq">
    <rdfs:label>uniq</rdfs:label>
  </omelette:Service>
  <omelette:Service rdf:about="http://www.ict-omelette.eu/omr/operator/urlbuilder">
    <rdfs:label>urlbuilder</rdfs:label>
  </omelette:Service>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/dittes">
    <rdfs:label>dittes</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/monitor">
    <rdfs:label>monitor</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/summize">
    <rdfs:label>summize</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/terraminds">
    <rdfs:label>terraminds</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/tweet">
    <rdfs:label>tweet</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/tweetscan">
    <rdfs:label>tweetscan</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/twemes">
    <rdfs:label>twemes</rdfs:label>
  </ctag:Tag>
  <ctag:Tag rdf:about="http://www.ict-omelette.eu/omr/tags/twitter">
    <rdfs:label>twitter</rdfs:label>
  </ctag:Tag>
</rdf:RDF>
```

Sample widget extracted from Opera Widgets:

```
<rdf:RDF
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:omelette="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://widgets.opera.com/category/fun-and-games/">
    <rdfs:label>Fun and Games</rdfs:label>
  </skos:Concept>
  <omelette:Widget rdf:about="http://widgets.opera.com/widget/10322/">
    <dc:description>A simple but addictive puzzle game: pick all the sticks as fast
as possible. You can only pick the top stick at any one time. On "small" levels,
extra sticks will fall every few seconds.</dc:description>
    <dc:source rdf:resource="http://widgets.opera.com/widget/10322/"/>
    <omelette:categorizedBy rdf:resource="http://widgets.opera.com/category/fun-and-
games/"/>
  <omelette:endpoint rdf:resource=
"http://widgets.opera.com/widget/download/force/10322/1.0/"/>
    <rdfs:label>Marocco</rdfs:label>
  </omelette:Widget>

</rdf:RDF>
```

The following tables summarize the components that were discovered out of Programmable Web, Yahoo Pipes and Opera Widgets.

Table 1: OMR components summary

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

Table 2: Runnable components in OMR

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

5. Automated Composition

5.1. Introduction

There are many mashup tools which enable the development of mashups on all layers of composition (data, logic, user interface), but still it's a manual and time consuming process which is typically done by skilled developers. The goal of the composition task is to enable the composition of mashups even for unskilled end-users, without the knowledge about component nature, APIs and composition methods. The OMELETTE team envisions a composition engine that is able to process user goals, define components needed and configure them in order to provide the desired functionality.

A problem of automatic composition could be formulated as following: given behavioural description of a Web application (user goals) as well as a set of semantically described components, create a UI mashup, which consists of a configured subset of given components and fulfils the specified user goal.

Mashup components in OMELETTE mean data sources exposed as Atom/RSS feeds, Web services (SOAP or REST) and widgets (e.g. ones compliant with W3C widget specification). The user goal should be able to focus on what mashup should be able to do, not how. He is not required to specify components, data sources or composition logic. Thus, if a user wants to display some data on a map and perform telephone calls, he does not need to know about the underlying implementation, for example data feeds, widgets and message flows.

Particularly, the automatic composition task implies tackling following subtasks:

- **User goal acquisition**. The end user has to put his goals in an easy but expressive way. The behaviour definition may comprise functional and non-functional requirements of the resulting mashup. Its complexity and structure should be hidden from end users, so that they can focus on the business goals. Though many approaches exist to model user tasks [Pat97, Pat02, PatMar09], they are adjusted to workflow composition and require a lot of formalism. In contrast, the purpose of UI mashups is not to put together one single business process or link processes to automate some task, but to integrate separately created components together “on the glass” and enable them to interact without prior knowledge about each other [Phi05]. The OMELETTE approach aims at building a lightweight goal model, where a user defines only mashup capabilities and non-functional requirements of the mashup components. The detailed description of the goal metamodel and acquisition facilities is given in section 5.2

- **Selection of mashup components**. As described above, UI mashups may comprise different types of ingredients - data feeds, web services, widgets etc. A first step in the composition process usually is to find appropriate components, which implement the desired functionality. Hereby an automatic composition engine should decide which types of components are needed as well as what capabilities or features they should provide. Most of the current discovery approaches combine formal methods with Semantic Web technologies in order to find mashup ingredients [Ped10]. Though these techniques are the most powerful and efficient ones, they require precise and extensive description of components, which in reality are only rarely available. On the other side, there are also lightweight approaches, which are used to search for components using keywords or some limited number of search criteria [ProWeb11]. Keywords and natural language descriptions are usually the only sources of information about component functionality. In OMELETTE approach both techniques to search for the components are utilized and a reuse (and adoption) of a search and disambiguation framework developed earlier by one of the OMELETTE partners is foreseen. Furthermore, selection and ranking methods based on syntactic and semantic descriptions of component parameters have been investigated. They provide tools for measuring components compatibility [Ngu10] within given non-functional boundaries. The detailed description of the selection process is given in section 5.3.1**.**

- **Configuring the mashup components**. Once the required components have been identified, they need to be assembled and configured. In fact, there are several tasks dealing with assembly and configuration on different levels of composition. For example, on the data level, the composition engine has to find and compose data, which is required to fulfil user goals. On the service composition level services should be orchestrated in order to provide the required application logic. In the OMELETTE approach, the service layer is omitted based on the assumption, that the required application logic is limited in complexity and is implemented within widgets, which is usually the case in general UI mashups. Finally, on the presentation level, widgets should be initialized with data specified in the user goals and also configured to interact with other widgets. In the OMELETTE approach, data aggregation algorithms are investigated as proposed in [Ria08]. The sections 5.3.3 and 5.3.4 give detailed information on utilized composition techniques.

5.2. User goal analysis

This section gives an introduction of the goal representation meta-model and defines a grammar to enable text-based user input.

Illustration 9 presents an UML diagram which describes the constituent parts of the goal meta-model and their relationships.

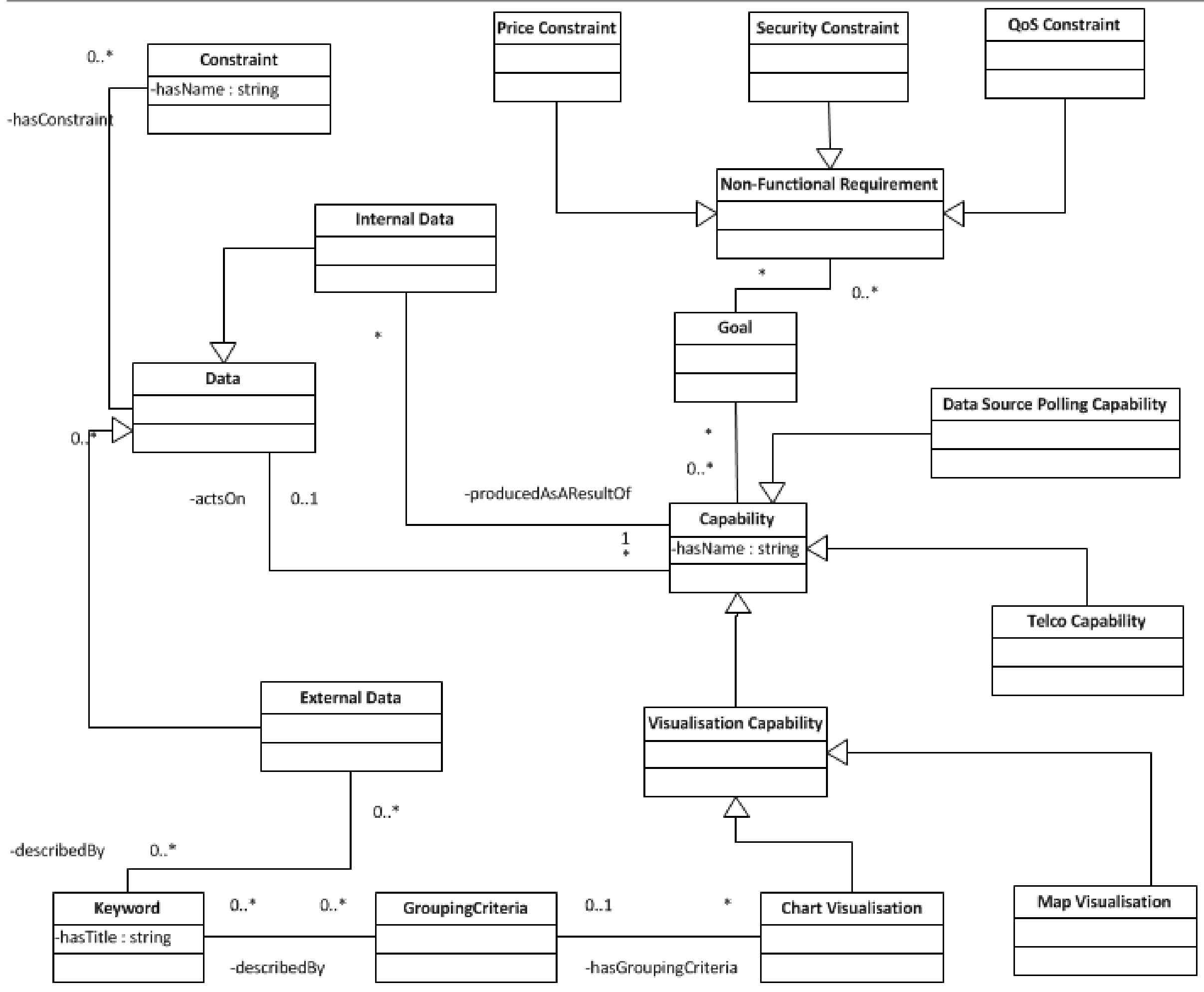


Illustration 9: User goal meta-model

The user formulates his goal by defining the desired *capabilities* of a mashup and its *non-functional requirements*. OMELETTE reuses the concept of *capabilities,* which was originally defined for Web services in WSMO ontology as following: “a **capability** describes the functionality of a Web service by specifying its preconditions, assumptions, post-conditions and effects. Capability descriptions are used for Web service discovery based on functional properties” [Dom05]. OMELETTE extends this concept, so that it can be applied to UI mashups.

Each capability processes, transforms, consumes or just displays some *Data*. OMELETTE distinguishes between *External Data* (usually representing a data source outside the UI mashup and described with keywords) or *Internal Data* that corresponds to internal mashup messages, produced as a result of consumption of some other capability. Within the goal description a user can filter data by defining so called *Constraints*. An example of a capability is “show museums nearby on the map”, which is a *Map Visualisation* capability with geo-locations of “museums” with short descriptions as *External Data* and “nearby” as a *Constraint*.

The non-functional requirements affect the component ranking and selection process, for example limiting the search e.g. only to free of charge components or the ones with the highest ranking. Within the OMELETTE goal model there were initially foreseen three groups of non-functional properties: price constraints, security constraint and QoS constraints.

To capture user goals OMELETTE will provide an extensible rich user interface, which guides a user through the behaviour specification process. A grammar to restrict the text-based input to the supported goal model is described, which can be easily produced by parse tree transformation. Following, there is an initial version of such a grammar using extended Backus-Naur form [Far95].

```
goal ::= capabilities [“.” non-functional requirements]

capabilities ::= (visualisation | telco | polling ) [“and” capabilities]

visualisation ::= "show" [“their“ | “its”] data [“nearby”] "on the” visualizer

data ::= [“their“ | “its”] keywords [of keyword]

visualizer ::= "map" | chart | "table" | “timeline”

chart ::= “chart“ [“, grouped by” keywords]

telco ::= (“let me call” | “let me SMS” | “let me MMS”) subject

polling ::= “check” keywords

subject ::= “them“ | keywords

keywords ::= keyword (“,” keyword)*

non-functional   requirements   ::=   “Choose   components,   which“   (non-functional
requirements2)*

non-functional  requirement2  ::=  non-functional  requirement  (“and“  non-functional
requirement)*

non-functional requirement ::= “are free for use” | “have highest rank” | “provide
secure data transfer”
```

The following inputs are possible according to the defined grammar:

- **show** “plumber from Berlin” **on the map and let me call them**
- **show** “plumber from Berlin” **on the map and show them on the table**
- **show** “customers**” on the chart**, **grouped by** “city”
- **show** “news about sales and events**” on the timeline**
- **show** “friends” **and let me SMS them**
- **show** “museums” **nearby on the map**
- **let me call** “property management firms”

Consider the first goal definition: **show** “plumber from Berlin” **on the map and let me call them**. The corresponding parse tree would look like as depicted in Illustration 10.

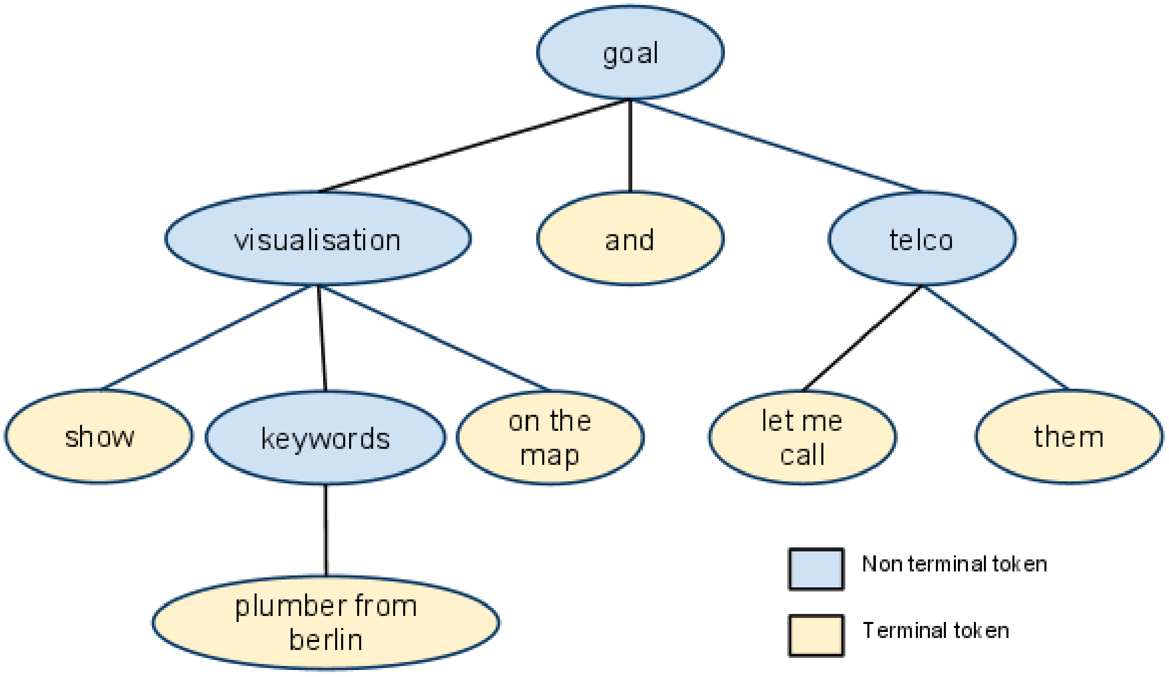


Illustration 10: Parse tree of the example input

Using a dedicated algorithm, the parsed tree will be transformed into the goal model, consisting of two capabilities - *Map Visualisation* and *Calling Capability*. Map Visualisation capability displays data coming from external data source on the map, while Calling Capability provides facilities to call persons selected on the map.

The resulting goal model is presented in Illustration 11.

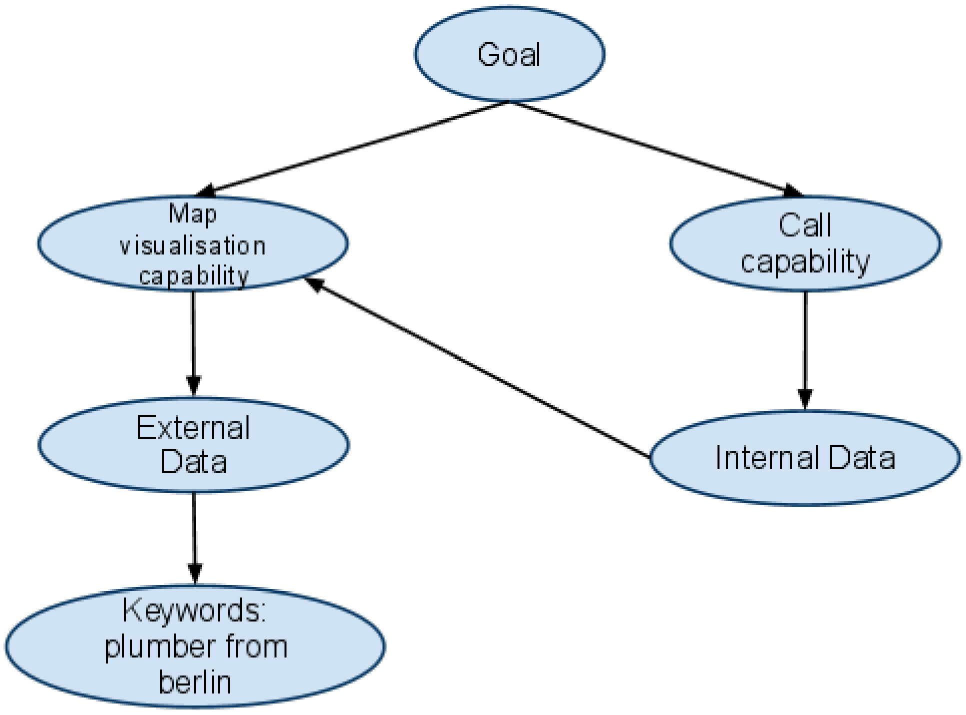


Illustration 11: Goal model for the example user input

A similar approach is demonstrated by GODO, which is a Goal-Driven approach for searching WSMO Web services. It consists of a repository with WSMO Goals and lets users state their goal by writing a sentence in plain English. A language analyser extracts keywords from the user sentence and a WSMO Goal will be composed based on those keywords. The WSMO Goal with the highest match will be sent to WSMX, an execution environment for WSMO service discovery and composition. WSMX then searches for a WSMO Web service that is linked to the given WSMO Goal via WSMO Mediators and returns the WSMO Web service back to the user. This approach makes good use of the capabilities of the WSMO framework, but it cannot be applied for other semantic languages like OWL-S and WSMO-Lite (the output of NESSI strategic projects named SOA4ALL), which do not have such goal representation elements.

5.3. Transforming the goal model into an UI mashup

The goal model acts as a starting point for the subsequent mashup composition. Before we proceed with a description of OMELETTE automatic composition approach, it is necessary to formalize the understanding of a UI mashup from the point of view of the composition engine. UI mashup is a triple of:

- *Widgets*, which encapsulate the application logic and provide user interface for their consumption
- *Data services*, which are used to provide the data for widgets. These can be REST/SOAP services or Atom/RSS feeds, which are displayed or processed within the widgets.
- *Configuration of Inter-widget communication*, which is the specification of message flows between widgets. Widgets can interact with each other in order to implement capabilities from user goals. Formally, the configuration of inter-widget communication is a set of message flow elements E, each of them is pair of widgets (W1, W2), where widget W2 consumes events produced by widget W1.

In order to compose a mashup, first it is necessary to identify its basic components and describe them using information provided by the user. The resulting conceptual (or abstract) description can be also referred to as a *structural* goal model - it reflects the capabilities defined by the user in terms of mashup components (widgets, data sources and message flow elements). Subsequently OMELETTE refines the conceptual mashup description by finding concrete widget implementations or data services.

The algorithm to produce the conceptual mashup description is the following.

Algorithm1. Transformation of user goal model to conceptual UI mashup description

Input: user goal model

Output: conceptual UI mashup specification

1. Map capabilities to widgets.
    a. Let W be an empty set of widgets.
    b. For each capability *k* from user model G: Find a RDF concept *c*, which corresponds to the capability *k* (for example, the capability “Call capability” can correspond to the concept “http://ict-omelette.eu/ontology/CallingCapability”). Add a new widget element *w* to the set W and assign the concept *c* to the widget.
2. Define and assign data sources. In the goal model 2 data types are foreseen: the external and the internal one. The external data describes data sources coming from the Web, whereas the internal one indicates the data dependency between capabilities. For this reason, external data elements from the goal model are treated as data services (like Atom/RSS/REST-based services), whereas internal data is mapped to message flow elements between widgets.
    a. Let M be an empty set of message flow elements.

b. For each capability *k* from user model G: let *w* be a corresponding widget element from the abstract UI mashup description. If capability *k* acts on external data, add its keywords to the widget specification. Otherwise, if capability *k* acts on an internal data, define a new message flow element E = (x, w), where x is the widget, that corresponds to the capability which produces the internal data. Add the element E to the set M.

The goal model from the example above corresponds to the following abstract UI mashup description:

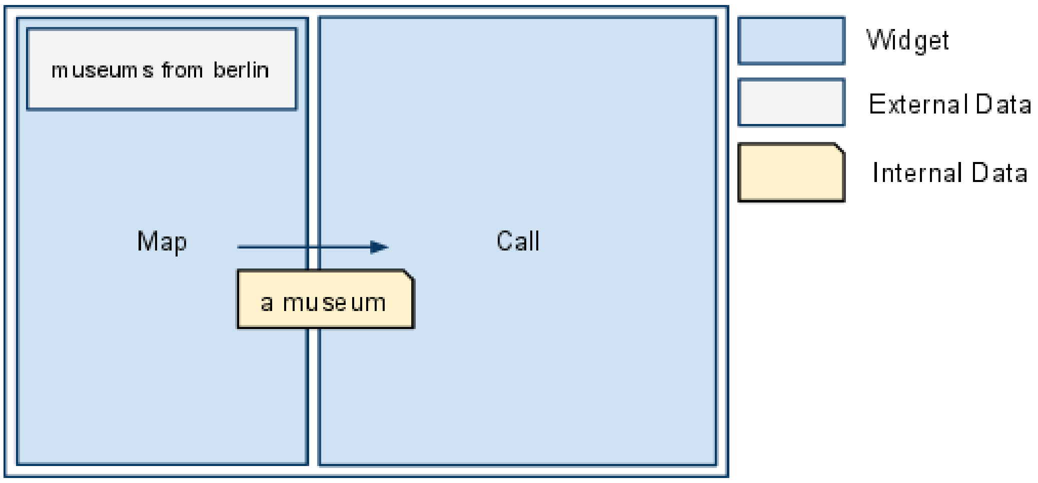


Illustration 12: Abstract UI mashup description

The serialization of the model is done using the abstract MDL and looks as following:

```
<workspace>
  <widgets>
  <widget id="1" type=”http://ict-omelette.eu/ontology/MapVisualizationCapability”/>
  <widget id="2" type=”http://ict-omelette.eu/ontology/CallingCapability” />
  </widgets>
  <messageflows>
       <messageflow id="3" from="1" to="2"/>
  </messageflows>
  <dataservices>
       <dataservice id=”4” widgetid=”1”>
         <keywords>plumber from berlin</keywords>
       </dataservice>
  </dataservices>

</workspace>
```

The produced mashups fulfil the users' goal by providing a workspace with a set of standalone widgets, which implement the desired functionality and are configured to process the specified data. In order to produce a concrete implementation of the mashup, OMELETTE performs the following steps:

1. Find concrete widget implementations, according to the conceptual description
2. Configure inter-widget communication according to the message flows defined in conceptual description
3. Find or build data services, which provide data according to the conceptual description. Assign them to the corresponding widgets

Following, we describe the above steps in detail.

5.3.1. Finding widgets that implement the basic capabilities

The selection process assumes that widgets are described within the OMELETTE Mashup registry using the Resource Description Framework (RDF). Exposed widget properties may vary from widget to widget, but comprise of a minimal set of data, which is {title, category, description, tags}. These are also the attributes, which are commonly used in the Web repositories. If further attributes are available such as the ones from the OMELETTE model above, more sophisticated discovery methods based on widget events, operations, pricing models, ranking etc. can be applied. The selection process is based on the ranking of the results. As described below, widgets are ranked higher if they satisfy desired capabilities (completely or partially), if they correspond to non-functional requirements, if they provide inter-widget communication facilities and if they can be used together with other widgets with compatible event types.

Following, there is a proposed widget selection algorithm.

Algorithm 2. Finding concrete widget implementations from the conceptual UI mashup description

Input: conceptual description of UI mashup

Output: set S of concrete widget implementations, assigned to widget elements from conceptual description.

1. For each widget description *d* from the conceptual description:
   a. Let *c* be the RDF concept associated with the description *d*. Find widgets in the repository annotated with the property *hasCapability* and object *c*
   b. If no widgets found, consider the tags, which correspond to the concept *c* in the capability ontology (e.g., capability “http://.../CallingCapability” can be described using Tags “call”,”phone”,”telco”,”telephony” etc.)
      i. For each tag *t*: search for widgets, which are tagged using the tag *t*. If no widgets found stop the execution.
   c. Rank the found widgets according to the number of matching tags
   d. If non-functional requirements are defined, map them to the corresponding attribute constraints (e.g. price, security, quality of service, context information, language preferences, relation to the social graph of the current user etc.) and increase the rank of those widgets, which meet the constraints.
2. Connect widgets from sets found for each capability *k* with an imaginary edge, if they use compatible events (at least one). Set the weight of the edge to the sum of the ranks of the connected widgets.
3. By selecting exactly one widget in each set, maximize the sum weight of edges between the sets. If no edges exist to and from the set, select the widget with the highest rank.

Step 3 of the algorithm searches for the “best” set of widgets, so that they implement the desired functionality and also use compatible events. The used strategy advocates the choice of the widgets with the highest rank, whereas possible solutions with lower ranked widgets but more compatible ones are ignored. In case of same rank random set of widgets is chosen. Consider the following configuration:

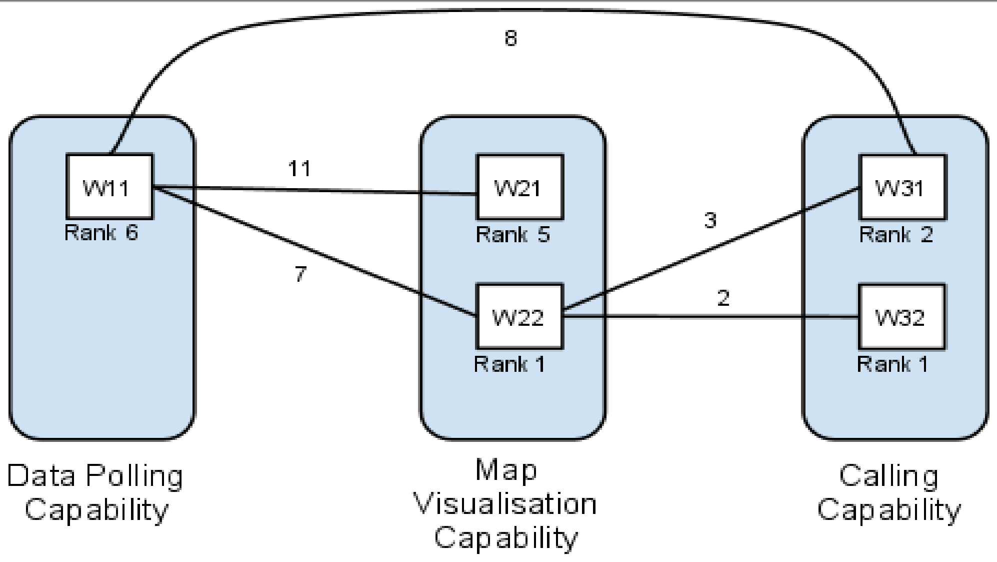


Illustration 13: Example of widget ranking process

Assume that a user expects three capabilities from the mashup - data polling, map visualisation and calling. The algorithm has found a set of ranked widgets for each capability. The last step selects one widget from each set so that the sum of weights of edges between them is maximal. In our case these are W11, W21 and W31. The sum of the edges is 19. However, other feasible choices could be possible - for example, W11, W22 and W31. The sum of the weights of the edges is lower (18), which means, that some of the widgets do not satisfy the desired capabilities as good as the first selection, but the compatibility of events is higher, which implies better user experience and more seamless workflows. At the end it is the user, who can decide, what the “best” is for him, especially if a set of highly ranked incompatible widgets is compared to the low ranked one but with compatible widgets.

The OMELETTE team envisions two different strategies in finding “optimal” set of widgets and present the results to the user, so he can choose between the possible solutions. The mentioned strategies are the following:

- By selecting exactly one widget in each set, maximize the sum weight of edges between the sets. (Higher rank is more important in the resulting solution)
- By selecting exactly one widget in each set, maximize the length of the path between the edges. If several paths with equal length were found, choose the one with the maximal summary weight (higher compatibility is more important in the resulting solution)

Which strategy is the best will be evaluated in future user studies. From the implementation point of view, both strategies require solving optimization algorithms on the graphs. The optimal solution can be found e.g. using brute force algorithms. Though they run in general in exponential time, we expect that the number of widget candidates in each set will be manageable and the overall algorithm performance will be not impeded.

5.3.2.Proposed solution for search and ranking of services

The proposed framework for OMELETTE fits business and technical requirements elicited from a number of scenarios.

The resulting framework for service search and ranking is rather flexible and could be used in a number of domains although further user studies are necessary. It is also open to the extensions and modifications due to the use of patterns and interfaces among them.

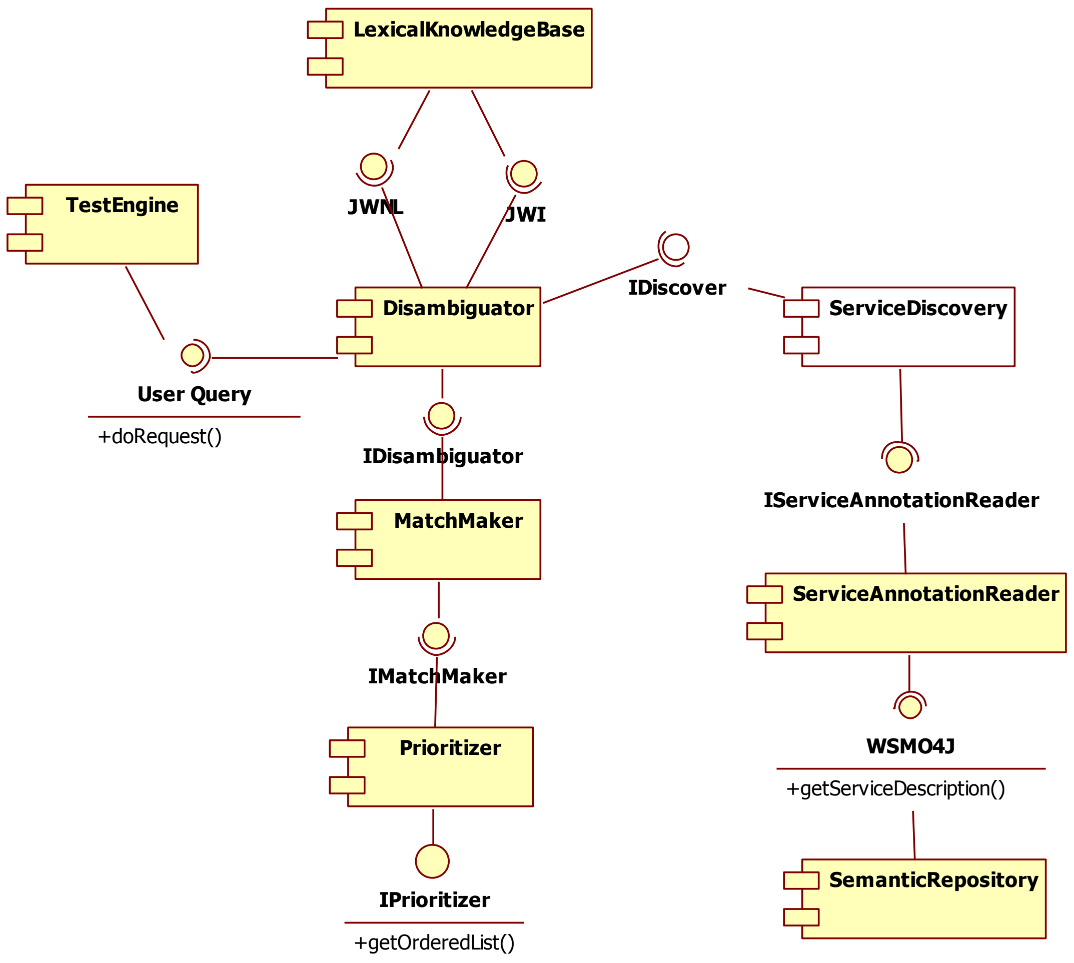


Illustration 14: Component diagram for the search and ranking framework

5.3.3. Configure inter-widget communication

After the required widgets have been identified, they need to be configured in order to exchange messages as specified in the conceptual description.

The message flow definitions describe the data transfer between required widgets, which means that the one widget consumes the data, which was produced by some other one. Example of such a dependency can be the map widget, which produces events when user clicks on a marker, and the calling widget, which consumes these events and sets the callee number to the one from the event parameter (Illustration 15).

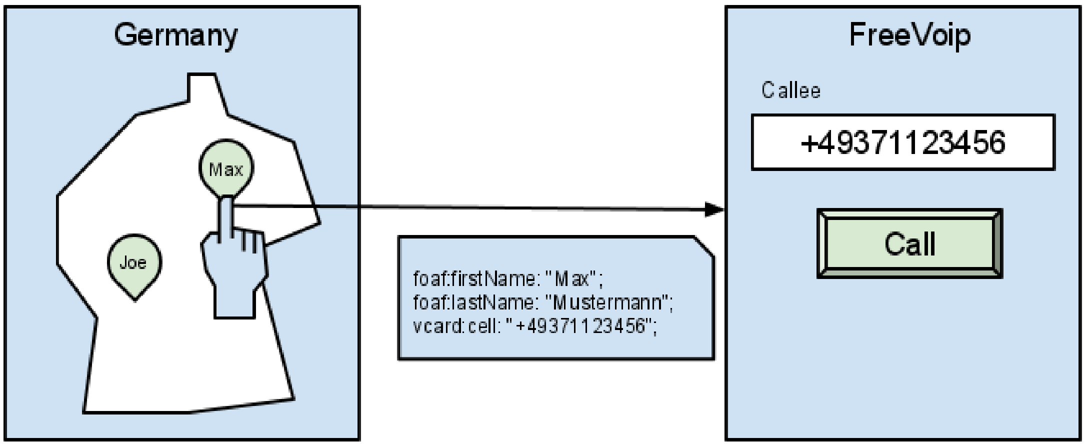


Illustration 15: Example of inter-widget communication

The configuration of inter-widget communication is based on the message flow elements defined in the conceptual description. Depending on the model, which is used to implement the inter-widget communication in concrete mashup environment, different configuration settings can be applied. In [Wil11] authors define three different models:

- *Orchestrated model*, where the interactions between the widgets are defined using a central control logic. This corresponds to the so called “wiring” of widgets, i.e. explicit definition of message flows. Events produced by the widgets are forwarded to specified operations of other widgets in a controlled manner.
- *Choreographed model*, where the interactions between the widgets in the mashup are not defined, but instead emerge in a distributed fashion from the internal capabilities of the widgets. This model corresponds to the publish-subscribe architecture, where widgets produce and consume all events, which occur on the common message bus. Though, this behaviour may be desirable in most of the cases, some scenarios require sealing off particular widgets to avoid loops or undesired side effects.
- *Hybrid model*, where individual behaviours of the widgets are inhibited, e.g. by isolating them from certain event types. This model combines the advantages of the first two models - simplicity and expressiveness.

The following steps should be performed for different models in order to transfer the configuration of inter-widget communication as produced by algorithm 1:

- In case of an orchestrated model we map the produced message flow elements to the direct inter-widget communications defined within the model, i.e. connections between events of the source widget with an operation of the target widget.
- The choreographed model does not let us configure the behaviour of the mashup, as it emerges in a distributed way when widgets are put into the workspace.
- For the hybrid model we first define constraints to disable any data flow between widgets. Afterwards, for each message flow element E=(W1, W2) from the configuration set M, we subscribe the widget W2 to receive all compatible types of events coming from widget W1.

5.3.4.Building data services

The data services within the conceptual mashup description are the inputs for widgets. For example, one may “feed” a generic map widget with location and contact information of museums in Berlin. Before introducing the OMELETTE approach to implement this functionality, the following assumptions should be made.

- Widget repository contains widgets that can accept data inputs coming from other widgets. Usually widgets, which were developed with inter-widget communication scenarios in mind, will provide the desired functionality in form of exposed initialising operations and by listening to special kind of initialisation events.
- The structure and semantics of initialisation events should be shared between all widgets. For example, an agreement can foresee to use RDF/XML as message format or JSON with pre-defined structure and semantics. In this case, all generic widgets, like map, chart, table etc. can understand and process the same initialisation data set. An example of initialisation data can be the following:

```
{
        "source" : "system";
        "name" : "http://ict-omelette.de/ontology/WidgetInitializationEvent";
        "value" : [
                {
                        "http://xmlns.com/foaf/0.1/firstName":"Max";
                        "http://xmlns.com/foaf/0.1/lastName":"Mustermann";
                        "http://www.w3.org/2006/vcard/ns#Cell":"+49371123456";
                },
                {
                        "http://xmlns.com/foaf/0.1/firstName":"Joe";
                        "http://xmlns.com/foaf/0.1/lastName":"Smith";
                        "http://www.w3.org/2006/vcard/ns#Cell":"+49157654321";
                },
                ...
        ]
}
```

- Data services within the registry are annotated using keywords, which describe the produced output.

Under the above assumptions the problem is reduced to building the data feed in the pre-defined format out of keywords given by the user. The simplest approach would be to search for registered data services (e.g. Atom/RSS feeds), which are annotated with a subset of given keywords. However, it is expected to be insufficient, as many data feeds like "museums in berlin" are not atomic and require filtering or aggregation functions on data sources. For that reason, OMELETTE will research on how to adopt the approach described in Wishful Search [Ria08], which is based on search and orchestration of data services based on keywords. An option would be to let user compose the data feed using the interface similar to MARIO [Bou08], which is then sent into the corresponding widget.

5.4. *Evaluation methodology*

The "reasonability" of resulting compositions/mashups can only be checked by human users during usability studies. Since this part is reserved by another OMELETTE work package here it is important to evaluate Precision and Recall of a proposed solution for retrieval and ranking of service/mashup descriptions.

First of all, it will be necessary to create descriptions for services using the automatic service discovery and annotation approach described in section 4. Then, those descriptions will be checked and validated by human expert users. During this step descriptions will be extended and corrected whenever it is possible and appropriate.

Then, a group of test queries will be prepared to measure the performance using queries designed to search for Web services in the repository that are known to be there and that are not present in there. In the latter case, a number of similar Web services retrieved from the repository will be evaluated.

Testing with N queries and M matching algorithms will provide N*M Precision-Recall graphs (PR-graphs). Because comparing so many graphs is impossible, PR-graphs consisting of average precision values for the recall points. This enables comparing all the different algorithms at once. However, the testing will be done with lists of preferred Web services that can vary in the number of Web services they consist of. For testing, lists that contain two to four preferred Web services will be used.

Because these variations in the number of Web services cause different recall values, average precision values could only be calculated for queries that have the same amount of preferred Web services.

Example PR-graphs shown in demonstrate the average results for the approximate matching tests.

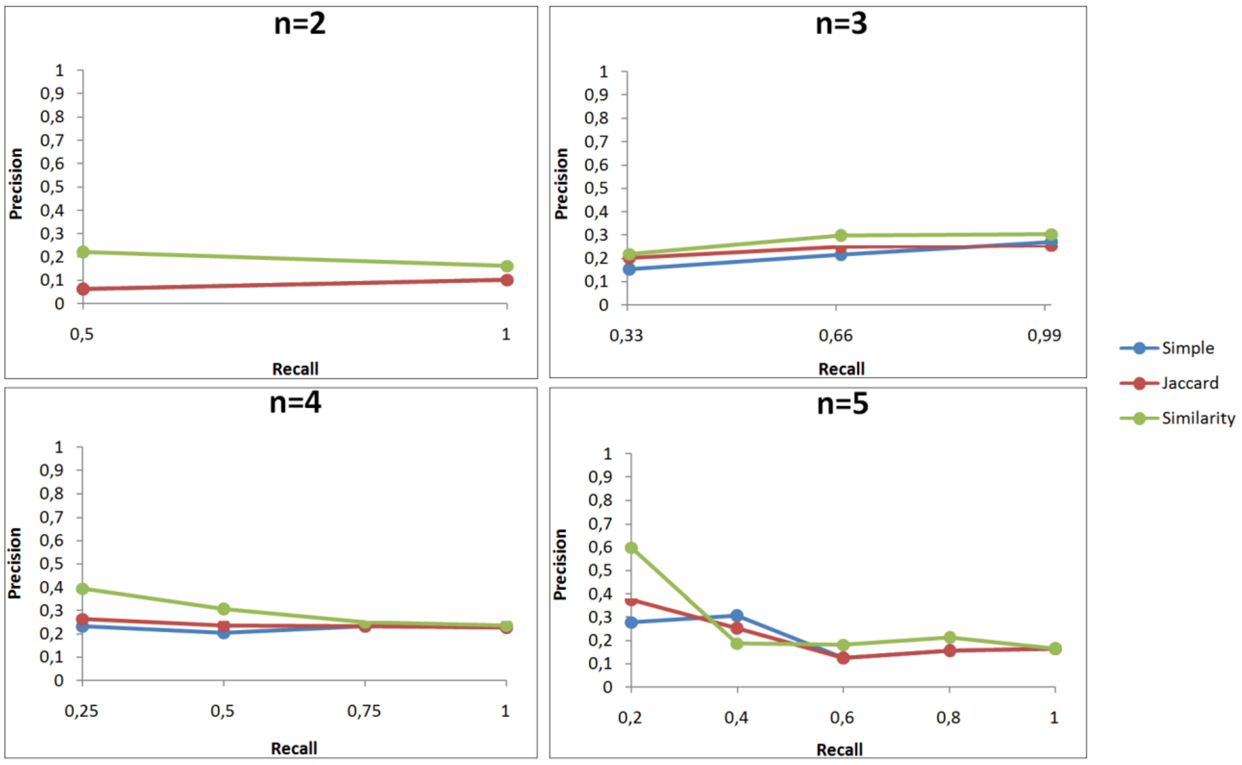


Illustration 16: Example of PR-Graphs for discovery of similar services

This information will be used as a feedback to the research and engineering activities of WP5 in order to improve the quality of a solution.

6. Conclusions and next steps

The proposed OMELETTE solution is based on the state-of-the-art technologies and approaches driven by academic research and industrial partners. This approach proposes a combination of goal driven service composition with keyword-based discovery phase and ranking based on non-functional attributes. A knowledge bottleneck in this case represented by a lack of necessary service and mashup descriptions is overcome by the novel approach that utilizes machine learning techniques.

OMELETTE approach also distinguishes from other UI mashup frameworks by the use of natural language processing and a WordNet-based similarity measure for matching keywords. This enables the framework to also discover related Web services or Web services that are described using synonyms of the words from the search query. By searching directly for semantic annotations, this framework can use ontologies to support the compilation of the senses of words used in Web service descriptions.
The future work will be to integrate the results of the all work package partners in the consistent picture of holistic OMELETTE architecture being developed within WP2.

7. References

[Fer11]	J. I. Fernández-Villamor, Carlos Ángel Iglesias, Mercedes Garijo Ayestarán, "A Semantic Scraping Model for Web Resources -- Applying Linked Data to Web Page Screen Scraping," In Proceedings of the Third International Conference on Agents and Artificial Intelligence, 2011.
[Scr11]	Scrappy screen scraper. http://github.com/josei/scrappy
[ScOnt11]	Scraping Ontology. http://lab.gsi.dit.upm.es/scraping.rdf
[Pat97]	Paterno, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A diagrammatic notation for specifying task models. pp. 362-369. Chapman & Hall (1997)
[Pat02]	Mori G. Paternò F. Santoro C.: CTTE: Support for Developing and Analysing Task Models for Interactive System Design. Ieee Transactions on Software Engineering, vol. 28 (8) pp. 797 - 813. IEEE (2002).
[PatMar09]	Paterno, F., Santoro, C., Spano, L.D.: MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact. 16(4), 1-30 (2009)
[Phi05]	Phifer G., "Portals Provide a Fast Track to SOA," *Business Integration Journal*, vol. 7, no. 9, p. 76, 2005.
[Ngu10]	Ngu A. H. H., Carlson M. P., Sheng Q. Z., and Paik H.-young, "Semantic-Based Mashup of Composite Applications," *IEEE Transactions on Services Computing*, vol. 3, no. 1, pp. 2-15, Jan. 2010.
[Ped10]	C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert, J. Kopecky, and J. Domingue, "iServe: a linked services publishing platform," *Workshop Ontology Repositories and Editors for the Semantic Web at 7th Extended Semantic Web Conference*, 2010.
[ProWeb11]	ProgrammableWeb - Mashups, APIs, and the Web as Platform. [Online]. Available: http://www.programmableweb.com/. [Accessed: 09-Jun-2011].
[Ria08]	A. V. Riabov, E. Boillet, M. D. Feblowitz, Z. Liu, and A. Ranganathan, "Wishful search: interactive composition of data mashups," *Proceeding of the 17th international conference on World Wide Web WWW 08*, vol. 8, p. 775, 2008.
[Dom05]	J. Domingue, D. Roman, and M. Stollberg, "Web Service Modeling Ontology (WSMO): an ontology for Semantic Web Services," *Architecture*, pp. 776-784, 2005.
[Wil11]	S. Wilson, F. Daniel, U. Jugel, and S. Soi, "Orchestrated User Interface

	Mashups Using W3C Widgets," *World Wide Web Internet And Web Information Systems*.
[Gae97]	H.-W. Gellersen, R. Wicke, and M. Gaedke, "WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle," 1997.
[Chu10]	O. Chudnovskyy and M. Gaedke, "Development of Web 2.0 Applications using WebComposition / Data Grid Service," Service Computation 2010, 2010.
[Bou08]	E. Bouillet, M. Feblowitz, Z. Liu, A. Ranganathan, and A. Riabov. 2008. A tag-based approach for the design and composition of information processing applications. *SIGPLAN Not.* 43, 10 (October 2008), 585-602.
[W3C11]	W3C, WebID - Wiki. [Online]. Available: http://www.w3.org/wiki/WebID [Accessed: 29-Sep-2011]
[WAC11]	W3C, WebAccessControl - Wiki. [Online]. Available: http://www.w3.org/wiki/WebAccessControl [Accessed: 29-Sep-2011]
[Hai07]	Hai, L. et al.: Web Services Federation Language (WS-Federation). 2007. [Online]. Available: http://www.ibm.com/developerworks/library/specification/ws-fed/. [Accessed: 29-Sep-2011]
[Fra99]	J. Franks, P. Hallam-Baker et al.: RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc2617.txt. [Accessed: 29-Sep-2011]
[Far95]	Farrell, James A: Extended Backus Naur Form. 1995. [Online]. http://www.cs.man.ac.uk/~pjj/farrell/comp2.html#EBNF. [Accessed: 29-Sep-2011]