*Journal of Universal Computer Science, vol. 18, no. 17 (2012), 2407-2431*
*submitted: 16/1/12, accepted: 29/8/12, appeared: 1/9/12 © J.UCS*

# A Metadirectory of Web Components for Mashup Composition

**José Ignacio Fernández-Villamor, Carlos Á. Iglesias, Mercedes Garijo**
(Departamento de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid, Spain
{jifv, cif, mga}@dit.upm.es)

**Abstract:** Because of the growing availability of third-party APIs, services, widgets and any other reusable web component, mashup developers now face a vast amount of candidate components for their developments. Moreover, these components quite often are scattered in many different repositories and web sites, which makes difficult their selection or discovery. In this paper, we discuss the problem of component selection in Service-Oriented Architectures (SOA) and Mashup-Driven Development, and introduce the Linked Mashups Ontology (LiMOn), a model that allows describing mashups and their components for integrating and sharing mashup information such as categorization or dependencies. The model has allowed the building of an integrated, centralized metadirectory of web components for query and selection, which has served to evaluate the model. The metadirectory allows accessing various heterogeneous repositories of mashups and web components while using external information from the Linked Data cloud, helping mashup development.
**Key Words:** mashups, services, widgets, components, discovery, integration
**Category:** H.3.4, H.3.5

## 1 Introduction

In today's Web, developers enjoy the availability of plenty of services, data feeds, widgets and other components 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[1] or mashArt[2] that facilitate developers' combining services for application construction. Also, Programmable Web[3], Yahoo Pipes[4], and Opera widgets[5] are examples of repositories that include services and widgets of many different kinds. They can be queried by users to search for useful applications and services that they can reuse for mashup composition.

However, because of this mushrooming of web components and mashup platforms, developers face some difficulties when working in this development process of mashup construction. First, it is not easy for a developer to find the most

[1] `http://www.ict-romulus.eu/web/mycocktail`
[2] `https://sites.google.com/site/mashtn/industrial-projects/mashart`
[3] `http://www.programmableweb.com`
[4] `http://pipes.yahoo.com`
[5] `http://widgets.opera.com`

appropriate component for a mashup being built, as there are many of them available and the information might be scattered in various repositories in the web. Second, the components employ different standards and semantics, thus requiring some study of the documentation. And third, the components often need to be adapted for their use by the mashup platform in question.

To overcome these limitations, an integrated metadirectory of web components for mashup composition is proposed in this paper. The metadirectory makes use of the Linked Mashups Ontology (LiMOn), a component model we introduce in this paper, which comprises useful information for querying web components and searching for the most appropriate ones. Additionally, LiMOn reuses other underlying standards, such as Web Service Modeling Ontology (WSMO) [Roman et al., 2005] or the World Wide Web Consortium (W3C) Widgets standard [Cáceres, 2011, Alario-Hoyos and Wilson, 2010], as low-level grounding description languages that allow web components to be readily executable. These descriptions are built automatically, when possible, in a discovery phase that has allowed of populating the metadirectory with actual components from the web.

This paper is structured as follows. Section 2 discusses the problem of choosing web components when developing mashups in Service-Oriented Architectures (SOA). Section 3 describes LiMOn, the model used to describe and annotate the components, and discusses how it fits into the framework of component selection. In Section 4, a metadirectory which makes use of LiMOn is described, as well as the approach that has been followed to populate the metadirectory with actual web components. The metadirectory and the data obtained are evaluated in Section 5. Finally, related works are summarized in Section 6 and some conclusions from the research done and some perspectives for research are given in Section 7.

## 2 Problem statement

Mashup-Driven Development [Iglesias et al., 2011] proposes reusing web components to build new applications. These kinds of combined web applications are commonly known as mashups, and are combinations of components which vary from a Representational State Transfer (REST) [Fielding, 2000] service to a widget, gadget, portlet, or even a web application that shares its information as a data feed. Therefore, when developing an application, developers can choose from a wide range of available components, combining them to obtain a new working system. Hence developers face the problem of choosing the right component for the right task. First, the component needs to fit the functional requirements of the tasks in the newly constructed application. E.g., if a service for geolocation is sought, a developer first needs to filter out all non-related services that do not deal with mapping services or geolocation. And second, the component needs to fit other non-functional requirements, such as trust in the company behind the component, or certain quality aspects that the component needs to meet.

Thus, developers need to search for the appropriate component according to some high-level needs they have. According to the type of component (API, service, widget. . . ) the developers would have to check one registry or another (e.g. either a widget repository or some service registry). Also, depending on the features sought in the component, some registries would be more appropriate than others (e.g. some registries might show information about the semantics of the service and others not). And again, according to the features sought, it would be necessary to query external sources to find the component information (e.g. it might be necessary to look up a components' vendor at Wikipedia in order to get an idea of the component's trustworthiness).

The Software Engineering Institute distinguishes several domains in the management and development of SOA [Lewis and Smith, 2007]. These domains are business domains, engineering domains, operations domains and cross-cutting concerns. Each domain maps to an aspect which is relevant to component selection when dealing with SOA:

- The business domain comprises all the consequences that a service orientation has on a given organization, application domain, or context. This includes *business aspects* such as cost or legal issues when selecting a component for its reuse.
- The engineering domain deals with the service-oriented lifecycle. At the time of selecting a component for reuse, this domain would contain the *technical aspects* of a service, i.e. formats, interface, semantic descriptions, and so on.
- The operations domain deals with the operation, evaluation and optimization of service-oriented systems. Namely, in this domain the aspects that are involved when selecting a component will be *Quality of Service aspects*, which determines the evaluation of a component's performance.
- Cross-cutting concerns include aspects that are orthogonal to all the domains. The main aspects are *trust and social aspects*, which affect and determine other aspects in some way or another.

Next, these aspects will be discussed in detail, along with the references found in the literature.

### 2.1 Business aspects

Business aspects are any selection criteria that fall into the business domain of Service-Oriented Architectures. Whenever an aspect of a decision relevant to selecting a service can have an impact on a given organization's structure, it is considered by us to be a business aspect.

*Cost* is the most notable business aspect when selecting a service. [Raj and Sasipraba, 2010] understands cost as *the economic condition of using a service*, and includes it in a Quality of Service model for service selection. Similarly, [Rehman et al., 2011], [Zeng et al., 2009] and [Li et al., 2010] define frameworks for comparison of cloud providers that also include cost as one of the selection aspects. Although some of these works use a broad definition of cost (involving non-monetary aspects as well), it can be considered a fundamental business aspect in component selection.

*Legal restrictions* are also a business aspect regarding component selection. Very often components are only usable under certain conditions in a reduced number of countries, which affects the selection process depending on the company's activities and targets. [Shimba, 2010] considers legal issues as one challenge for cloud computing, enumerating the different difficulties that are encountered because of countries having different regulations and laws on the topic. The complexity behind this diverse regulation makes legal restrictions hard to model, which, as will be seen later, might be the reason why most component registries dismiss legal restrictions.

The *vendor* and the possible agreements with the consumer company may influence the decision of component selection. A company often agrees to use another company's components under a certain domain. For example, Apple's mobile devices started out by provide tight integration with Twitter although other microblogging services were available, after reaching an agreement that benefited both companies [Eaton, 2011]. Therefore, most times knowing the specific vendor that provides the service behind a web component is needed prior to taking the decision to use it. Thus, the vendor is another business aspect to consider regarding component selection.

### 2.2 Trust

According to [Amoroso et al., 1991], software trust is *the degree of confidence that exists that the software will be acceptable for "one's needs."* This implies that, after a developer has been convinced about a software component's specifications, thanks to some documentation, trust would be the confidence that these specifications would be met over time. Related properties that have been identified are popularity, maturity, company trust, and community trust.

*Popularity* is an indicator of the success of a web component [Mileva et al., 2010]. Success can be understood in several ways, and different indicators can be used to measure it. A component that is widely used is considered succesful, while one with many bugs has a lack of success. However, a component which lacks bug reports might also lack community support because of a lack of popularity. Thus, popularity is a combination of different indicators that signal the active use by a big enough community of users. Popularity increases the so-called trusting-intention [Kutvonen, 2007], or the will to depend on another component with the involved risks. Therefore, popularity is a relevant metric of the trust aspect.

*Maturity* is another software feature which increases trust. The topic of maturity is widely covered in the area of Open Source [Polancic et al., 2004] because of the nature of these kinds of projects—they usually follow an iterative growth, with frequent releases until they reach some point of maturity [Raymond, 1999]. The Capability Maturity Model (CMM) [Paulk et al., 1993] and the Open Source Software Maturity Model (OSSMM) [Golden, 2005] are models to improve the software process's maturity in companies for traditional software development and Open Source software development, respectively. These models emphasize the importance of seeking maturity in software.

*Company trust* is the critical factor behind the global trustworthiness of a component. A component is more trustworthy if a trustworthy company is behind it. Many factors are involved in the trust in a company, e.g., company size, financial equity, and customer service. [Nguyen et al., 2006] identifies communication, cultural understanding, and capabilities, as the three top factors that determine trust in a software company. Similarly, a survey has for instance revealed that 75% of users place more trust in companies that use microblogging services such as Twitter [Gershberg, 2010]. All this reveals communication channels as key factors that determine company trust and thus the trust aspect.

### 2.3 Quality of Service aspects

Quality of Service in the Internet is traditionally regarded as *the combination of network-imposed delay, jitter, bandwidth and reliability* [Ferguson and Huston, 1998]. This is a typical network-level definition that can be extended to the application level by considering the metrics that a particular vendor offers for their commercial components. [Hu et al., 2005] proposes a decision model of Quality of Service applied to Web Services that can be extended to components. They propose a model for selecting Web Services according to the metrics of execution cost, execution time, reliability, and availability. Similarly, [Menasce and Almeida, 2002] define Quality of Service as a combination of availability, security, response time, and throughput. We will generalize these terms to availability, reliability, and performance.

*Availability* is the percentage of time a web component is operating [Menasce, 2002]. When applying this definition to complex web components, this availability depends on the availability of several resources. I.e., in the case of a widget, both its assets (external scripts and Hypertext Markup Language pages) and its services must be available for the widget's availability. [Zhang and Zhang, 2005] point out similar problems in the domain of mashups. Availability is an important aspect when selecting a web component.

*Reliability*, as a general term in software, is *the probability of failure-free operation of a computer program for a specified time in a specified environment* [Musa et al., 1987]. This definition can be applied to Service-Oriented Architectures by considering web components as the software elements that are to provide failure-free operation. [Zhang and Zhang, 2005] state that a reliable web service *must exhibit correctness, fault-tolerance, testability, availability, performance and interoperability.* Those are a set of requirements that a service must maintain in order to consider it failure-free. Other papers [Majer et al., 2009] consider as well the problem of reliability in more complex components, such as mashups, which reuse other services, thus depending on third-parties' reliability and availability.

We identify the *performance* aspect as a way to encompass execution times, responsiveness, and throughput issues, noted as important elements when regarding Quality of Service [Menasce and Almeida, 2002]. Depending on the nature of a web component, some metrics would make sense and others would not. For example, the concept of throughput cannot be applied to a web component such as a widget (but it can be to a service), although its user interaction's responsivity can be measured in the same way as a web service's. These kinds of issues are regarded as performance aspects.

### 2.4 Technical aspects

The technical aspects behind selecting a web component involve all the issues related to component operation. Several component description standards such as Web Service Definition Language (WSDL) [Christensen et al., 2001], WSMO [Roman et al., 2005] or W3C widgets [Alario-Hoyos and Wilson, 2010] model the main characteristics that involve operation aspects for specific web component types such as services or widgets. The operation aspects these standards consider can be grouped into a few areas: interface, dependencies, and cross-cutting concerns.

*Interface* comprises the aspects such as conditions that are involved in the communication with the component. WSMO employs preconditions and postconditions to model a service's interface, and offers ways to identify formats and protocols employed in the communication [Lara et al., 2004].

*Dependencies* include any requirement of external components. Especially, mashups [Majer et al., 2009] are components that are mainly built out of other components, such as web services, widgets, or data feeds. Awareness of these dependencies can help knowing about the indirect requirements or use restrictions (if, e.g., a client-side mashup requires a geolocation service that is not available in the user's country).

*Cross-cutting concerns* involve non-functional technical aspects such as security, choreography issues, or required standards. Web Services often group these aspects in the so-called WS-* standards [Alonso, 2004]. Similarly, WSMO allows defining non-functional properties for a service in order to specify these kinds of aspects [Toma and Foxvog, 2006]. An interesting cross-cutting concern in SOA is discoverability, which measures *the extent to which the service, service consumers expect to look for, is easily and correctly found* [Choi and Kim, 2008]. It takes place if a component's capabilities are published in one way or another. In order to allow web components to be found by developers, their capabilities need to be announced for an agent to discover them. A textual description of the component's functionality is the minimum requirement to make a component discoverable, although a semantic description allows automatic processing.

### 2.5 Support and coverage in existing repositories

In the web, we can find several repositories of components that can be reused for creating new applications. By glancing at these repositories, we can evaluate the support and coverage of the different aspects that we have identified for component selection.

A repository will be said to support a property if its schema allows a precise description of that property. For example, Programmable Web provides a field to link a component to a WSDL file. A WSDL file is a Web Service standard description language that allows describing a service's interface. This allows components in Programmable Web to have full support to the interface aspect. The range of support is from zero (the property is not considered in the repository) to four (full details of the property can be included in the repository).

On the other hand, a repository provides full coverage of a property if all of its components make use of the supported fields for that aspect. The coverage of a property in the repository represents the current usage of the property by the users, i.e. to what extent users fill those properties. The range goes from zero (completely missing property) to 5 (used by all components). For instance, Programmable Web supports WSDL annotated services by providing a field to reference a WSDL file. However, most APIs in Programmable Web do not make use of that field, which results in low coverage of the interface aspect. The metric of coverage thus represents the actual degree of use of a repository's capabilities.

Table 1 shows the analysis of the support to every aspect in three repositories. The support that each repository gives to each aspect is marked from zero points (no support) to four points (full support). We have selected three repositories that are both heterogeneous and popular. A short description of each of them is given next.

– Programmable Web is the most popular directory of mashups and APIs on the web. It is a collaborative directory where users can provide awareness of a mashup or an API. There are several fields that the users can fill in to provide information about a particular component. APIs such as Google Maps[6] and mashups such as Panoramio[7] have their own page where users add information on related APIs and mashups, or any other useful information.

- Yahoo Pipes is a repository of user-built mashups called “pipes”. Each pipe is created using an editor developed by Yahoo, which allows users to create new data feeds out of existing ones. Then, pipes appear listed in the Yahoo Pipes web site, where users can find, and clone, other pipes and even reuse them to build new ones.
- Opera Widgets is a repository of widgets that are created by users. Opera does not provide an editor for this task, and simply allows users to upload their widgets and publish them in their web site. The web site then provides a browsing interface so that users can search widgets by category and load them into their browser.

As already mentioned, coverage is a complementary metric that reveals the actual degree of use of each aspect. Figure 1 illustrates the coverage for each repository and aspect. The region inside each graph represents the degree of coverage for each repository. It is worth noting that both Yahoo Pipes and Opera Widgets are strict repositories that require all fields to be filled in for each component. Programmable Web, on the other hand, accepts optional information, such as the mentioned case of WSDL descriptions. As not all APIs in Programmable Web are linked to a WSDL file, this reduces the values in the technical axes. Also, information about legal issues (terms and conditions), vendor, or popularity (rating for each component) are optional properties in Programmable Web that are not always filled out. Also, checking the quality and veracity of the property values is out of the scope.

## 3 Linked Mashups Ontology

In this section, we describe a model that integrates the properties and fields that are provided by current component repositories in the web. It is called the Linked Mashups Ontology (LiMOn), for its approach of bringing Linked Data to Mashup-Driven Development. Linked Data [Bizer et al., 2009] is an initiative towards publishing semantically annotated contents for their consumption by automatic processes, in a way to achieve the vision of the Semantic Web. The Linked Data initiative suggests using HTTP URIs for identifying resources and employing standard formats for semantic annotations, such as RDF, in order to allow web resources being dereferenceable while providing semantic information.

[6] `http://maps.google.com`
[7] `http://panoramio.com`

| Aspect | Property | Programmable Web | Yahoo Pipes | Opera Widgets |
|---|---|---|---|---|
| Business | Cost | A field that plainly indicates whether or not there are use fees is indicated (2 points) | All the "pipes" in the repository are free, so cost is implicit (4 points) | All the widgets in the repository are free, so cost is implicit (4 points) |
| | Legal issues | Links to commercial and free licenses (1 point) | All "pipes" share the same license (4 points) | All widgets share the same license (4 points) |
| | Vendor | Vendor is provided (4 points) | Author is shown (3 points) | Author is shown (3 points) |
| Trust | Popularity | Developers can rate APIs and mashups and the number of mashups that use an API is shown (4 points) | Number of "cloned pipes" are shown, which is an indicator of popularity (1 point) | Widget users can vote widgets up and down (3 points) |
| | Maturity | Addition date of a component is a naïve indicator of maturity (1 point) | Creation date of the "pipe" can be an indicator of maturity (2 points) | Addition date of a widget is a naïve indicator of maturity (1 point) |
| | Company trust | Vendor and home page are shown, but with no trust indicators (1 point) | Author is shown, but with no trust indicators (1 point) | Author is shown, but with no trust indicators (1 point) |
| QoS | Availability | No indicators (0 points) | No indicators (0 points) | No indicators (0 points) |
| | Reliability | No indicators (0 points) | No indicators (0 points) | No indicators (0 points) |
| | Performance | No indicators (0 points) | No indicators (0 points) | No indicators (0 points) |
| Technical | Interface | Yes, through a link to a WSDL service description (4 points) | Implicitly provided through an HTML form and a well-known uniform output for every "pipe" (4 points) | Yes, by sharing widgets the W3C widget standard (4 points) |
| | Dependencies | Mashups are connected to the APIs they use (4 points) | "Pipes" are connected with the feeds they use (4 points) | No information about dependencies (0 points) |
| | Cross-cutting concerns | Information about SSL use, category, tags, plus the information available in WSDL (4 points) | Tags and a textual description are provided to categorize a pipe (2 points) | Only a taxonomy and a textual description is provided (2 points) |

Table 1: Repositories' support for aspects

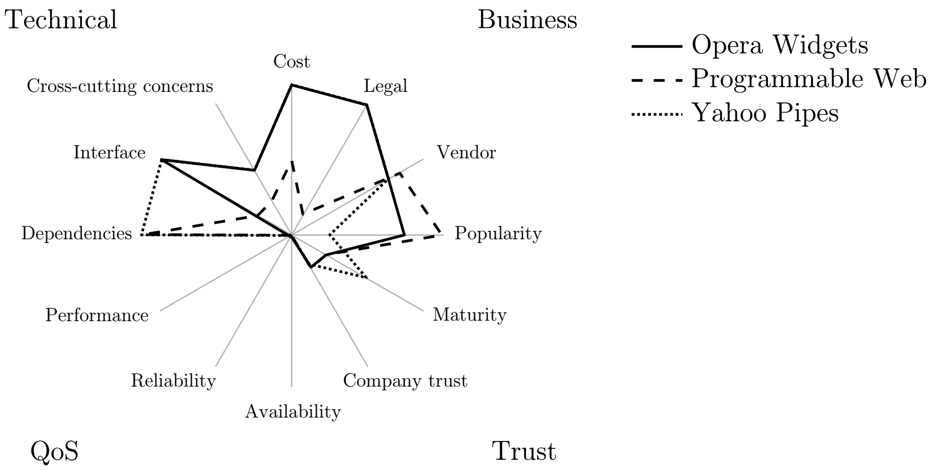


Figure 1: Repositories' coverage of aspects

With these considerations in mind, we have defined the ontology[8] presented in Figure 2. Regarding the technical aspect that was introduced in Section 2, the properties listed next have been included in the model.

– A set of properties allow covering interface aspects. The property *description* allows linking to a lower-level component description, such as WSMO, W3C widgets or WSDL, depending on the nature of the component. The property *endpoint* allows linking to the particular Uniform Resource Locator (URL) where the component runs. Also, the properties of *dataFormat* and *protocol* allow of specifying how the data, if any, is exchanged with the component.

– The property *uses* is employed to link to reused components. For example, it can be used to indicate which services or data feeds a mashup reuses.

– Some properties address cross-cutting concerns. The property *clientInstall-Required* indicates whether or not the web component requires an additional component installed client-side to work. The property *example* allows of referencing examples of the use of the component's API. The properties *tag* and *category* allow of linking to tags and categories, respectively, that represent the functionality of the component. The property *api* links to the specification of the component's programming interface. Finally, the properties *authentication* and *sslSupport* allow of specifying how security over the component's data transport is implemented.

[8] The full description of the ontology is available at `http://www.gsi.dit.upm.es/~jifv/limon`

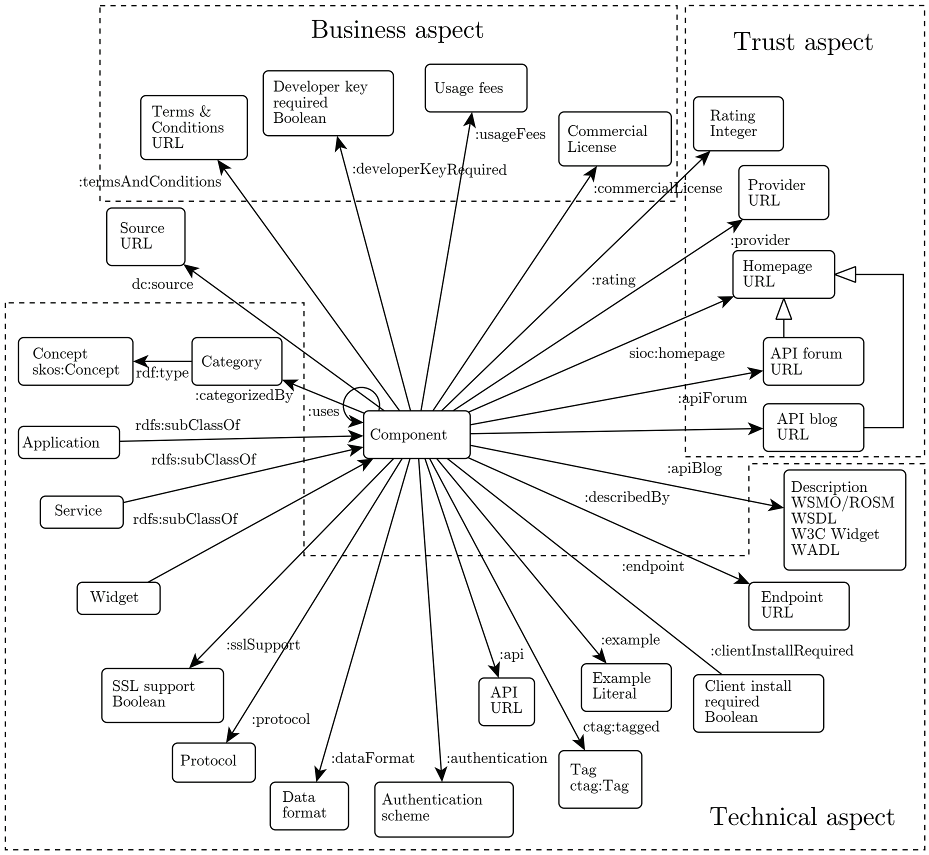


Figure 2: Linked Mashups Ontology (LiMOn)

The trust aspect comprises popularity and company trust issues, and includes the properties listed next.

– The *rating* property serves as an indicator of popularity. It represents the rating made by users in repositories reflecting their degree of satisfaction with a particular component.

– The properties of *apiForum* and *apiBlog* address the issue of company trustworthiness by providing means to reference support facilities (i.e., forums and blogs) that the vendor provides to component users. Also, the property *provider* allows of identifying the vendor of the component, for any company trust issues involved.

The business aspect comprises costs, and legal and vendor issues, and is covered by the model through the following properties:

– The property *usageFees* is a cost aspect property that links to any cost incurred when using the component.

– Regarding legal issues, the property *termsAndConditions* allows linking to a document that informs about the conditions for the use of the component. The property *commercialLicense* links to a commercial license for the use of the component, if any. Finally, the property *developerKeyRequired* indicates whether or not the component requires creating a developer account prior to its use.

– The property *provider*, already mentioned as part of the trust aspects, also serves to identify the vendor of the component for any business issues involved.

Furthermore, the source of a component is included as a property. In the next sections, this model will be used to build a metadirectory. This makes it useful to reference where the component was obtained from, thus requiring a property to link it to the source repository. Also, regarding the Quality of Service aspect, no information was found in any repository, so no field was included in the model.

In sum, the component repositories of Yahoo Pipes, Programmable Web, Opera Widgets, iGoogle Gadgets[9], AppStore[10], Android Market[11], and Ohloh[12] were analysed to identify the relevant properties for the model.

Figure 3 shows the connections between the component model and the ontologies that have been reused, illustrating how these links can be exploited thanks to already existing tools. Such tools generally comprise ways of exploiting the Linked Data graph, either by allowing the identification of new relations between resources or by providing ways to visualize the data.

Dublin Core schema [Weibel et al., 1998] is used to represent basic publishing metadata in LiMOn. SKOS (Simple Knowledge Organization System) [Miles and Bechhofer, 2008] is used for modelling categorizations and mapping the resulting taxonomies, as will be seen in section 4.1.1. CommonTag is employed for representing tags in the ontology. Semantically-Interlinked Online Communities (SIOC) [Breslin et al., 2006] is used for representing social sharing aspects such as blogs or forums that are related to the components. FOAF (Friend of a Friend) [Brickley and Miller, 2000] ontology is used to express authoring information, and finally Freebase [Bollacker et al., 2008] and DBPedia [Auer et al., 2007] are generalistic ontologies which can be used for expressing unambiguous entities for the actual individuals of the ontology.

[9] http://www.google.com/ig/directory
[10] http://itunes.apple.com/de/genre/ios/id36
[11] https://market.android.com
[12] http://www.ohloh.net

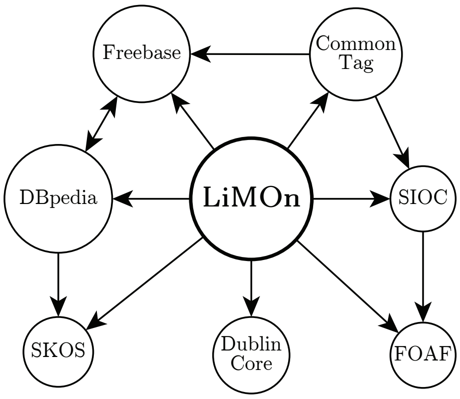


Figure 3: Connections between LiMOn and other ontologies

## 4 Metadirectory of mashups

A metadirectory that makes use of LiMOn has been built. This metadirectory integrates heterogeneous components that can be potentially used in various web applications. More specifically, mashup applications, services, and widgets from the Web are the components that will be included in the metadirectory because of the repositories that have been targeted, again, Programmable Web, Yahoo Pipes, and Opera Widgets.

In order to make the components addressable by developers, the metadirectory stores the 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 metadirectory with real components. Usually, web component repositories contain metadata such as a component's name, textual description, tags, and categorization. Other specific properties that depend on the nature of the component can also be found, such as inputs, endpoints, web service dependencies, and underlying formal descriptions such as WSMO or WSDL.

### 4.1 Data harvesting and integration

In this section, we will cover how the metadirectory has been populated with components from the targeted repositories (i.e. Programmable Web, Yahoo Pipes, and Opera Widgets) and how the data has been integrated.

We have defined a semantic proxy layer on top of the repositories. For each repository, we have defined the mappings between their HTML contents of their web resources and the Resource Description Framework (RDF) data they provide according to the model defined in Section 3. To define these mappings, we have used the Scraping Ontology[13] [Fernández-Villamor et al., 2011]. This approach lets the system have an RDF view of the unstructured data in the source repositories. With that, an automated agent crawls the source repositories and extracts the RDF data, which are then stored in the metadirectory.

Once the metadirectory is populated with components from the web, a unified categorization scheme is sought in order to provide a homogeneous interface for querying the metadirectory. This is necessary because of the diversity that is present in the categorization of the targeted repositories. For instance, components retrieved from Programmable Web are already tagged and use their own categorization scheme. The ones from Yahoo Pipes only have the tags set by the users. On the other hand, Opera Widgets provides components that are classified under a closed set of categories. Therefore, the components do not share a common categorization scheme, which limits the querying capabilities.

To integrate all the categorization schemes, we will define mappings between the concepts of each taxonomy. This enables querying the metadirectory by using any of the available categorization schemes without restricting the query to a particular repository. To achieve this, we will define a new categorization scheme by clustering the components available in the metadirectory. This automatically built scheme will be mapped to the categorization schemes provided by Programmable Web and Opera Widgets. Additionally, a mapping between the Programmable Web scheme and that of Opera Widgets will be manually defined.

### 4.1.1 Automatic categorization

In this section, we will describe how to automatically build a categorization system that allows users to query the metadirectory. In many cases, components already belong to a category that was defined in their source repository. As previously mentioned, both Programmable Web and Opera Widgets provide some categorization schemes, with categories such as "Tools", "Mapping", or "Sports". In the case of the Yahoo Pipes repository, only tags are used to categorize each pipe.

Whenever only tags are used to categorize components, we propose the following method to build a categorization scheme based on the most common tag combinations in the component space. We will use clustering techniques to identify the most common categories in the space, and thus to define a new categorization scheme. The resulting categorization scheme will be mapped to the other schemes in Section 4.1.2 to provide a uniform interface for querying the metadirectory.

[13] `http://lab.gsi.dit.upm.es/scraping.rdf`

To perform the clustering, components are modeled as a vector representing their tags:

$$a = (a_1, a_2, ..., a_n), a_i \in \{0, 1\} \tag{1}$$

A weighted Euclidean distance between a pair of components $a$ and $b$ is used by the clustering algorithm:

$$d(a, b) = \sqrt{\sum_{i=1}^{n} w_i \cdot (a_i - b_i)^2} \tag{2}$$

The weights for each dimension are adjusted according to the popularity of the tag. This way, less relevant tags will have less weight in the measuring.

According to (1), an example of a simple set of components such as the following:

$$\begin{aligned} foursquare &= (mapping, social, games) \\ googlemaps &= (mapping) \\ facebook &= (social) \\ bluevia &= (mapping, telephony, geolocation) \end{aligned} \tag{3}$$

would be represented by the next vectors:

$$\begin{aligned} foursquare &= (1, 1, 1, 0, 0) \\ googlemaps &= (1, 0, 0, 0, 0) \\ facebook &= (0, 1, 0, 0, 0) \\ bluevia &= (1, 0, 0, 1, 1) \end{aligned} \tag{4}$$

According to the popularity of each tag, the set of weights would be the following:

$$W = (0.375, 0.250, 0.125, 0.125, 0.125) \tag{5}$$

and thus some sample distances would be as follows.

$$\begin{aligned} d(bluevia, googlemaps) &= \sqrt{0.125^2 + 0.125^2} \approx 0.1768 \\ d(foursquare, facebook) &= \sqrt{0.375^2 + 0.125^2} \approx 0.3953 \\ d(facebook, bluevia) &= \sqrt{0.375^2 + 0.250^2 + 0.125^2 + 0.125^2} \approx 0.4841 \end{aligned} \tag{6}$$

With this we can compute the similarity between two components in the metadirectory. By using this similarity measure, we can perform some clustering to identify which are the most characteristic sets of components in the metadirectory.

A Sammon mapping has been used to represent the components and clusters [Sammon, 1969]. The Sammon's mapping function allows performing a dimensionality reduction on the component space and map the $n$-dimensional space to a two dimensional one, while attempting to preserve the distances between the represented vectors. This allowed us to visually estimate the number of clusters that were present in the system.

#### 4.1.2 Mapping identification

Mappings between categorization schemes are identified automatically using an algorithm that checks set intersections. Given two categories $\mathcal{A}$ and $\mathcal{B}$ with the component sets $A$ and $B$, respectively, the following mappings are identified according to the overlap between sets:

- If $\frac{|A \cap B|}{max(|A|,|B|)} \geq 0.95$, then $\mathcal{A}$ and $\mathcal{B}$ are considered equivalent categories.
- If $\frac{|A \cap B|}{max(|A|,|B|)} \geq 0.85$, then $\mathcal{A}$ and $\mathcal{B}$ are considered close categories.
- If $\frac{|A - B|}{min(|A|,|B|)} \leq 0.05$, then $\mathcal{A}$ is considered a subcategory of $\mathcal{B}$.
- If $\frac{|B - A|}{min(|A|,|B|)} \leq 0.05$, then $\mathcal{B}$ is considered a subcategory of $\mathcal{A}$.

These conditions are illustrated in Figure 4. As shown, SKOS ontology concepts are employed to define the mappings between categories. SKOS proposes a schema for the definition of taxonomies and mappings between them. The relation `skos:exactMatch` is employed for categories that are considered equivalent; `skos:closeMatch` indicates that two categories are very similar and could be used interchangeably in certain contexts; `skos:narrowMatch` indicates that the subject category is a subcategory of the object; `skos:broadMatch` states that the subject category is a supercategory of the object.

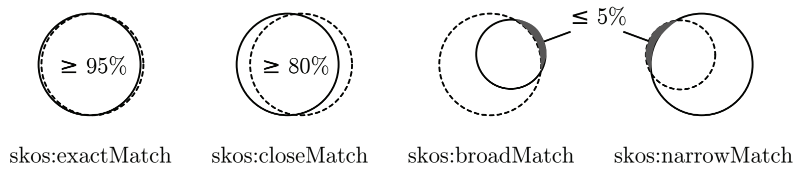


Figure 4: Mapping detection among categories

This allowed identifying a set of mappings among the different taxonomies. Figure 5 shows some of the mappings. The Opera Widgets repository provides no tags, so mappings with Programmable Web's taxonomy were defined manually. As can be seen, some categories are defined as sub- or super-categories of others, while others are defined as close or exact matches. In the case of the Yahoo Pipes repository, the previously described method for automatically building a taxonomy was used. We executed a clustering algorithm to obtain nine different categories. Then, the resulting categories were applied to Programmable Web's

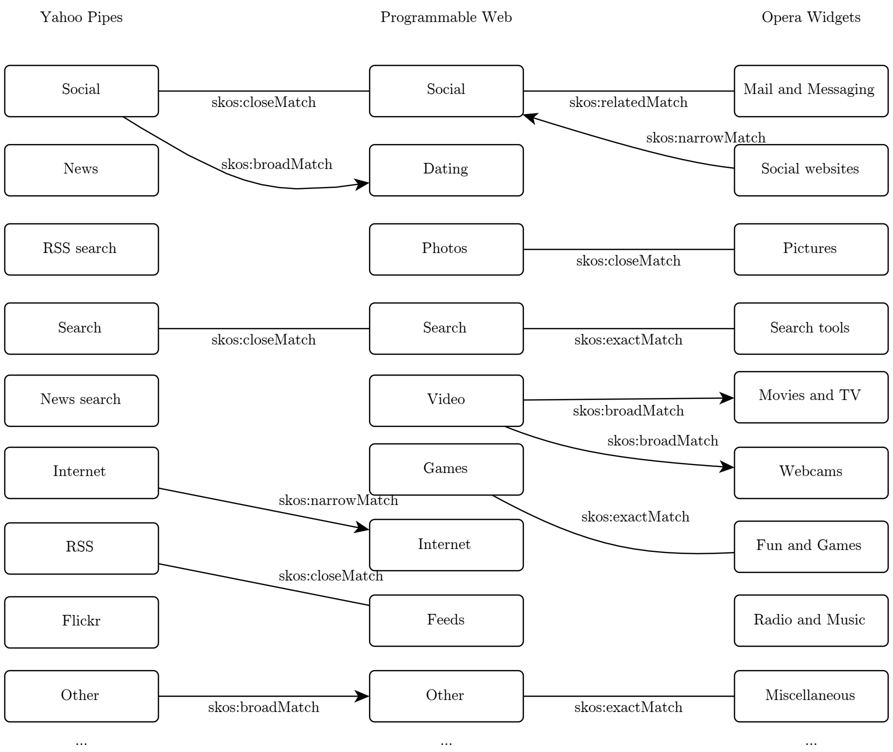


Figure 5: Mapping among the different categorization schemes

data. The resulting sets were compared to the ones that Programmable Web already provides as shown in this section, which resulted in the identified relations among the categories of the different taxonomies.

## 5 Results and evaluation

The metadirectory contains 10,194 services, 7,032 mashups and 1,804 widgets, as of a crawling performed in July 2011 on the mentioned repositories of Yahoo Pipes, Programmable Web, and Opera Widgets.

The metadirectory is used in the context of project OMELETTE (FP7-ICT-2009-5), in which an integrated environment for the construction of mashups has been built. A developer can build mashups by combining services and widgets into different kinds of mashups. For this, the centralized metadirectory contains the available components that developers can reuse. Therefore, the main use case for the metadirectory is a developer seeking a component that suits a mashup under development. For this, a browsing interface has been developed, and is shown in figure 6. Through this interface, the properties defined in LiMOn on-

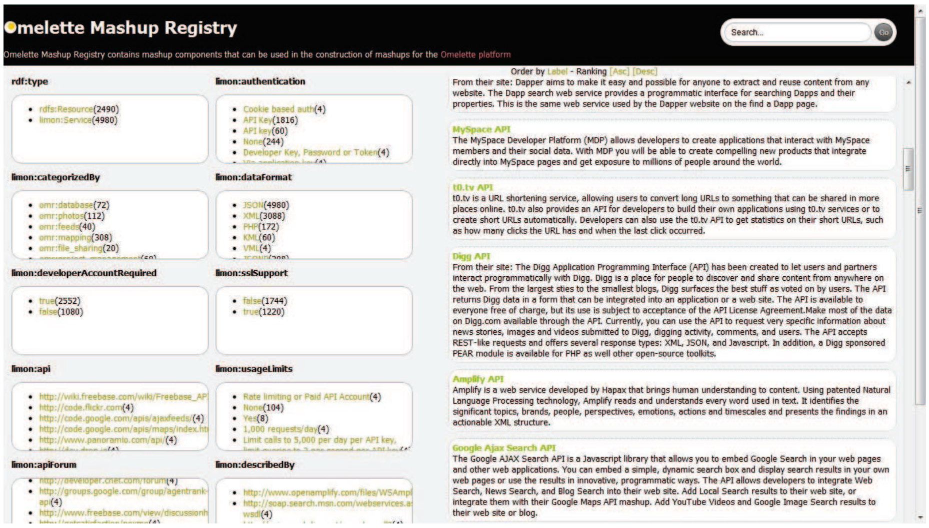


Figure 6: User interface for browsing the metadirectory

tology can be used for mashup filtering. Additionally, more advanced queries can be performed thanks to the use of semantic technologies such as SPARQL Protocol and RDF Query Language (SPARQL).

In order to evaluate the metadirectory, a set of queries have been defined to check its selection capabilities. Table 2 shows the SPARQL queries that result for each of the previously stated queries, along with the resulting components retrieved from the metadirectory. The queries are ones that a developer would make in order to retrieve the appropriate component for a particular problem.

1. *Which free components deal with photos/pictures?*. Usually, mashup editors offer a list of available components organized by categories. This is easily achievable in the metadirectory by filtering components that do not match a particular category. Without a metadirectory, this would imply visiting several repositories and browsing the desired category in order to get a list of suitable components.

2. *What mapping services are provided by Microsoft?*. Because of business issues, often developers need to select components based on the provider of the component. Registries such as Programmable Web provide vendor information but do not allow filtering by vendor.

3. *Which APIs are more commonly used by telco mashups?*. Occasionally, developers need insights on the use of a component in a particular domain. This query is focused towards telco mashups and the kind of APIs they use. A query such as this, though simple, is not allowed in the repositories where the components were extracted from.

4. *What commercial mapping services are readily usable?*. In some cases, component repositories only help provide *awareness* of a component, i.e. knowledge that the component exists and is available. This populates the metadirectory with components with purely general metadata such as a broad categorization and components with precise semantic descriptions such as WSMO descriptions. I.e., some components are already runnable and others are not. Our metadirectory retrieved services from Yahoo Pipes and transformed the execution forms into WSMO service descriptions. Also, many components in Programmable Web are linked to their WSDL file. This make it possible to find readily runnable components in our metadirectory and filter them in a query.

5. *What data sources are more often employed by news mashups?*. This query gives insights into the sources that are employed in the field of digital news. The metadirectory allows performing the query among applications present in different repositories.

6. *Which mapping APIs are provided by the most trustworthy companies?*. This query attempts to select services according to their trustworthiness. We will model trust by employing the provider's number of employees, as an indicator of the company's size. The query could be reformulated as retrieving all APIs which belong to the mapping category, sorted by the vendor's number of employees. Vendor information is retrieved from DBpedia, which illustrates the advantage of using LiMOn for linking information about mashup components.

In comparison with searching multiple repositories manually, the metadirectory enables:

– Accessing information about components that were originally available in separate, heterogeneous repositories from the web. As seen, component repositories offer information in their own format, which requires extraction and integration in a harvesting task. After that integration, the metadirectory allows querying through a uniform interface.

– Performing complex queries about these components. Usually, component repositories such as the ones employed are very limited in their querying capabilities. Although they offer plenty of information, they offer browsing functionalities rather than complex search interfaces.

– Using external information that is available in the Linked Data cloud to complement the information from the source repositories. Information available in DBpedia or other Linked Data sources can be integrated in queries to the metadirectory, allowing of extending the queries with data that is present in other systems.

## 6 Related work

In the present paper, a model for defining the aspects that are involved in web component selection for mashup composition has been defined. There are several approaches for dealing with web components of different kinds, from services to widgets. There are many initiatives to describe services' interface to allow the automation of certain tasks, in the Web Service field [Christensen et al., 2001] [Roman et al., 2005], or in the REST service area with heavy-weights such as the Web Application Description Language (WADL) [Hadley, 2006], or more light-weight approaches such as Microservices [Fernández-Villamor et al., 2010], WSMO-Lite [Vitvar et al., 2007], Semantically-Annotated REST (SA-REST) [Sheth et al., 2007], or hRESTS [Wright State University, 2008]. W3C widgets [Alario-Hoyos and Wilson, 2010] defines a standard for describing widgets. While these approaches allow describing the inners of these components, they operate at an abstraction level that is lower than that of our model, thus there are different standards for each kind of component. Therefore, we make use of them in our model by allowing the linking of a LiMOn description to a WSDL/WSMO/W3C widget description.

Furthermore, a metadirectory that integrates several repositories has been built. The automatic categorization approach allows mapping different categorization schemes. Some research works that perform similar tasks are available in the current literature. [Arabshian et al., 2012] focuses on the semi-automatic construction of an ontology out of textual descriptions of Programmable Web's components. Although Arabshian et al. do not consider mappings with other taxonomies, their approach successfully exploits textual descriptions of Programmable Web's components to build an ontology. Similarly, [Wang et al., 2011] mine Programmable Web and build a domain ontology out of the keywords available in the textual descriptions of the services. This helps to validate Programmable Web's categorization scheme. [Blake and Nowlan, 2011] performs an automatic categorization of services using the internals of WSDL descriptions, not just the keywords available in the textual descriptions. Neither work explores mappings with other categorizations or service repositories. Approaches such as [Bianchini et al., 2010] employ semantic technologies by using category labels and thesauri to compute semantic distances and mappings between concepts. In contrast, our approach exploits the information that is present in components'

| Query | SPARQL query | Results |
|---|---|---|
| *Which free components deal with photos/pictures?* | `SELECT ?component`<br>`WHERE`<br>`{ ?component rdf:type limon:Component;`<br>`    limon:categorizedBy limon:PhotoCategory;`<br>`  FILTER NOT EXISTS {`<br>`    ?component limon:usageFees ?fees .  } }` | Photobucket, TweetPhoto, AOL Pictures, Lockerz, Pixlr, Moodstocks, Fonxvard, Steply, Pixenate, Fishup, Shutterfly, Picmember, ExposureManager, PicApp, and 46 more |
| *What mapping services are provided by Microsoft?* | `SELECT ?service`<br>`WHERE`<br>`{ ?service rdf:type limon:Service;`<br>`  limon:categorizedBy limon:MappingCategory;`<br>`  limon:provider <http://www.microsoft.com> .  }` | Bing Maps |
| *Which APIs are more commonly used by telco mashups?* | `SELECT (count(?api) as ?apis) ?api`<br>`WHERE {`<br>`  ?mashup limon:uses ?api ;`<br>`          limon:categorizedBy`<br>`              limon:TelcoCategory .  }`<br>`GROUP BY ?api`<br>`ORDER BY DESC (?apis)` | Twilio (52%), Twitter (5.7%), Tropo (3.9%), Facebook (3.6%), other (34.5%) |
| *What commercial mapping services are readily usable?* | `SELECT ?service`<br>`WHERE`<br>`{ ?service rdf:type limon:Service;`<br>`    limon:categorizedBy limon:MappingCategory ;`<br>`    limon:usageFees ?fees ;`<br>`    limon:describedBy ?wsdl .  }` | CDYNE IP2Geo, ArcWeb, Postcode Anywhere, PeekaCity, ShowMyIP, FraudLabs Mexico Postal Code, FraudLabs ZIPCodeWorld United States |
| *What data sources are more often employed by news mashups?* | `SELECT (count(?api) as ?apis) ?api`<br>`WHERE {`<br>`  ?mashup limon:uses ?api ;`<br>`          limon:categorizedBy`<br>`              limon:NewsCategory ;`<br>`  ?api    limon:categorizedBy`<br>`              limon:FeedCategory .  }`<br>`GROUP BY ?api`<br>`ORDER BY DESC (?apis)` | CNN (2.18%), Google News (1.36%), NY Times (1.18%), BBC (1.09%), Yahoo News (0.91%), others (93.17%) |
| *Which mapping APIs are provided by the most trustworthy companies?* | `SELECT ?api ?provider ?employees`<br>`WHERE {`<br>`  ?api    limon:categorizedBy  omr:mapping ;`<br>`          limon:provider ?provider .`<br>`  ?dbpcompany dbpedia-owl:wikiPageExternalLink`<br>`                  ?provider ;`<br>`              dbpedia-owl:numberOfEmployees`<br>`                  ?employees .  }`<br>`ORDER BY ?employees` | Nokia Ovi Maps (Nokia: 132,430 employees), Ericsson Mobile Maps (Ericsson: 90,260 employees), Bing Maps (Microsoft: 89,000 employees), Google Maps (Google: 24,400 employees), Yahoo Maps (Yahoo: 13,600 employees), others |

Table 2: Evaluation of metadirectory's interface

tags to produce mappings between taxonomies. Unlike most approaches, we do not make use of semantic technologies. This allows identifying mappings that are not obviously related from a semantics point of view. Examples of these mappings are mappings to general concepts such as “other”, which could aggregate many diverse categories with names such as “miscellaneous” or “sports”, or mapping “RSS” (usually not present in a thesaurus) to “feeds”.

The metadirectory attempts to improve the experience of building mashups by allowing users to more easily discover useful components. There are approaches related to component recommendation that are similar to the one presented in this paper. [Elmeleegy et al., 2008] is a mashup advisor, which also builds a catalogue of mashup components to exploit in recommendations for mashup development. They rank components for their use in a mashup under development, which is outside the scope of our paper. Unlike our work, they do not consider integration with other component repositories. [Bianchini et al., 2010], on the other hand, integrates heterogeneous repositories and provides proactive recommendations during mashup development. Their approach do not employ tags for category construction, but semantic distances using WordNet’s thesaurus information on category labels, which allows matching components with no tagging information but ignores the potential data present in tags. [Picozzi et al., 2010] analyze proactive component recommendation from the point of view of the quality of the resulting mashup. The also define a component quality model that shares properties with our aspect-based framework such as trust, reliability, or functionality. Finally, [Pietschmann, 2010] is an approach on top of the CRUISe system [Pietschmann et al., 2009] which provides task-based recommendation of mashup components based on user-specified mashup requirements. It employs semantics to define a unified component model for accurate matching out of user requirements, but does not focus on the integration of such descriptions out of component repositories from the web. As future work, it would be interesting to research mapping the LiMOn ontology onto their task ontology.

## 7 Conclusions and future work

Through this paper, the different challenges that developers face when selecting components for building a mashup have been summarized. A common framework has been defined, which allowed of defining Linked Mashups Ontology (LiMOn), a unified model for components. With this model, several component repositories have been mined and loaded onto a metadirectory. A clustering method has been proposed and used to integrate the different taxonomies of the repositories in order to unify the categorization of the metadirectory. The metadirectory then offers a unified query interface that allows retrieving components through complex queries, involving components of different nature, making use of external data from the Linked Data cloud.

Future work involves refining discovery techniques to extend the available low-level information in services and make readily-executable service descriptions be available in the metadirectory. Namely, these techniques could consist of crawling the API documentation for concrete patterns that indicate service endpoints or use examples, which might enable administrators or semi-automatic tools to build WADL descriptions.

## Acknowledgements

Thanks to Pablo Moncada (UPM) for his support as well as to OMELETTE participants for their valuable feedback. This research project was funded by the European Commission under the R&D project OMELETTE (FP7-ICT-2009-5).

## References

[Alario-Hoyos and Wilson, 2010] Alario-Hoyos, C. and Wilson, S. (2010). Comparison of the main alternatives to the integration of external tools in different platforms. In *Intl. Conf. of Education, Research and Innovation (ICERI)*, pages 3466–3476.

[Alonso, 2004] Alonso, G. (2004). *Web Services: Concepts, Architectures and Applications*. Springer.

[Amoroso et al., 1991] Amoroso, E., Nguyen, T., Weiss, J., Watson, J., Lapiska, P., and Starr, T. (1991). Toward an approach to measuring software trust. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 198–218.

[Arabshian et al., 2012] Arabshian, K., Danielsen, P., and Afroz, S. (2012). Lexont: A semi-automatic ontology creation tool for programmable web. In *2012 AAAI Spring Symposium Series*.

[Auer et al., 2007] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. *The Semantic Web*, pages 722–735.

[Bianchini et al., 2010] Bianchini, D., De Antonellis, V., and Melchiori, M. (2010). A recommendation system for semantic mashup design. In *Database and Expert Systems Applications (DEXA), 2010 Workshop on*, pages 159–163. IEEE.

[Bizer et al., 2009] Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked data—The story so far. *sbc*, 14(w3c):9.

[Blake and Nowlan, 2011] Blake, M. B. and Nowlan, M. E. (2011). Knowledge Discovery in Services (KDS): Aggregating Software Services to Discover Enterprise Mashups. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):889–901.

[Bollacker et al., 2008] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *ACM SIGMOD*, pages 1247–1250.

[Breslin et al., 2006] Breslin, J., Decker, S., Harth, A., and Bojars, U. (2006). Sioc: an approach to connect web-based communities. *International Journal of Web Based Communities*, 2(2):133–142.

[Brickley and Miller, 2000] Brickley, D. and Miller, L. (2000). Foaf vocabulary specification 0.91. Technical report, ILRT Bristol.

[Cáceres, 2011] Cáceres, M. (2011). Widget Packaging and XML Configuration. `http://www.w3.org/TR/widgets/`.

[Choi and Kim, 2008] Choi, S. W. and Kim, S. D. (2008). A Quality Model for Evaluating Reusability of Services in SOA. *Quality*, pages 293–298.

[Christensen et al., 2001] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., and et al. (2001). Web services description language (WSDL) 1.1.

[Eaton, 2011] Eaton, K. (2011). Facebook Won't Like This Apple-Twitter Union.

[Elmeleegy et al., 2008] Elmeleegy, H., Ivan, A., Akkiraju, R., and Goodwin, R. (2008). Mashup advisor: A recommendation tool for mashup development. In *Web Services, 2008. ICWS'08. IEEE Intl. Conf. on*, pages 337–344. IEEE.

[Ferguson and Huston, 1998] Ferguson, P. and Huston, G. (1998). Quality of Service in the Internet: Fact, Fiction, or Compromise. *AUUGN*, page 231.

[Fernández-Villamor et al., 2011] Fernández-Villamor, J. I., Blasco-García, J., Iglesias, C. A., and Garijo, M. (2011). A Semantic Scraping Model for Web Resources—Applying Linked Data to Web Page Screen Scraping. In *Third Intl. Conf. on Agents and Artificial Intelligence*.

[Fernández-Villamor et al., 2010] Fernández-Villamor, J. I., Iglesias, C. A., and Garijo, M. (2010). A vocabulary for the modelling of image search microservices. In *Fifth Intl. Conf. on Evaluation of Novel Approaches to Software Engineering*.

[Fielding, 2000] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California.

[Gershberg, 2010] Gershberg, M. (2010). Consumers say: 'In tweets we trust'. http://www.reuters.com/article/2010/06/23/us-retail-summit-tweets-idUSTRE65L6C320100623.

[Golden, 2005] Golden, B. (2005). *Succeeding with Open Source*. Addison-Wesley.

[Hadley, 2006] Hadley, M. (2006). Web application description language. `https://wadl.dev.java.net/wadl20061109.pdf`.

[Hu et al., 2005] Hu, J., Guo, C., Wang, H., and Zou, P. (2005). Quality driven web services selection. In *IEEE Intl. Conf. on e-Business Engineering (ICEBE)*, pages 681–688.

[Iglesias et al., 2011] Iglesias, C. A., Fernández-Villamor, J. I., del Pozo, D., Garulli, L., and García, B. (2011). *Service Engineering: European research results*, chapter Combining, pages 171–200. Springer.

[Kutvonen, 2007] Kutvonen, L. (2007). Trust aspects in the architecture of interoperable systems. In *2nd Intl. workshop on Interoperability solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems (IS-TSPQ)*.

[Lara et al., 2004] Lara, R., Roman, D., Polleres, A., and Fensel, D. (2004). A conceptual comparison of WSMO and OWL-S. *Web Services*, pages 254–269.

[Lewis and Smith, 2007] Lewis, G. A. and Smith, D. B. (2007). International workshop on the foundations of service-oriented architecture (fsoa). *Special report CMU/SEI-2008-SR-011*.

[Li et al., 2010] Li, A., Yang, X., Kandula, S., and Zhang, M. (2010). CloudCmp: Comparing public cloud providers. In *10th Annual Conference on Internet Measurement*, pages 1–14. ACM.

[Majer et al., 2009] Majer, F., Nussbaumer, M., and Freudenstein, P. (2009). Operational challenges and solutions for mashups—An experience report. In *2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM)*.

[Menasce, 2002] Menasce, D. A. (2002). QoS issues in Web services. *Internet Computing, IEEE*, 6(6):72–75.

[Menasce and Almeida, 2002] Menasce, D. A. and Almeida, V. A. F. (2002). *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice-Hall.

[Miles and Bechhofer, 2008] Miles, A. and Bechhofer, S. (2008). SKOS simple knowledge organization system reference. *W3C Recommendation*.

[Mileva et al., 2010] Mileva, Y., Dallmeier, V., and Zeller, A. (2010). Mining API popularity. *Testing—Practice and Research Techniques*, pages 173–180.

[Musa et al., 1987] Musa, J. D., Iannino, A., and Okumoto, K. (1987). *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill.

[Nguyen et al., 2006] Nguyen, P. T., Babar, M. A., and Verner, J. M. (2006). Critical factors in establishing and maintaining trust in software outsourcing relationships. In *28th Intl. Conf. on Software Engineering*, pages 624–627. ACM.

[Paulk et al., 1993] Paulk, M., Curtis, B., Chrissis, M., and Weber, C. (1993). Capability maturity model. *Software, IEEE*, 10(4):18–27.

[Picozzi et al., 2010] Picozzi, M., Rodolfi, M., Cappiello, C., and Matera, M. (2010). Quality-based recommendations for mashup composition. *Current Trends in Web Engineering*, pages 360–371.

[Pietschmann, 2010] Pietschmann, S. (2010). A model-driven development process and runtime platform for adaptive composite web applications. *International Journal on Advances in Internet Technology*, 2(4):277–288.

[Pietschmann et al., 2009] Pietschmann, S., Voigt, M., Rümpel, A., and Meißner, K. (2009). Cruise: Composition of rich user interface services. *Web Engineering*, pages 473–476.

[Polancic et al., 2004] Polancic, G., Horvat, R. V., and Rozman, T. (2004). Comparative assessment of open source software using easy accessible data. In *26th Intl. Conf. on Information Technology Interfaces, 2004. Vol. 1*, pages 673–678.

[Raj and Sasipraba, 2010] Raj, R. and Sasipraba, T. (2010). Web service selection based on qos constraints. In *Trendz in Information Sciences Computing (TISC)*, pages 156 –162.

[Raymond, 1999] Raymond, E. S. (1999). *The Cathedral and the Bazaar*. O'Reilly.

[Rehman et al., 2011] Rehman, Z. U., Hussain, F. K., and Hussain, O. K. (2011). Towards Multi-criteria Cloud Service Selection. *2011 Fifth Intl. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 44–48.

[Roman et al., 2005] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., and Fensel, D. (2005). Web service modeling ontology. *Applied Ontology*, 1(1):77–106.

[Sammon, 1969] Sammon, J. W. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409.

[Sheth et al., 2007] Sheth, A., Gomadam, K., and Lathem, J. (2007). Sa-rest: Semantically interoperable and easier-to-use services and mashups. *IEEE Internet Computing*, 11(6):91–94.

[Shimba, 2010] Shimba, F. (2010). Cloud computing: Strategies for cloud computing adoption. Master's thesis, Dublin Institute of Technology.

[Toma and Foxvog, 2006] Toma, I. and Foxvog, D. (2006). Non-functional properties in web services. *WSMO Deliverable*.

[Vitvar et al., 2007] Vitvar, T., Kopecky, J., and Fensel, D. (2007). Wsmo-lite: Lightweight semantic descriptions for services on the web. In *Fifth European Conference on Web Services*, pages 77–86.

[Wang et al., 2011] Wang, J., Zhang, J., Hung, P. C. K., Li, Z., Liu, J., and He, K. (2011). Leveraging fragmental semantic data to enhance services discovery. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th Intl. Conf. on*, pages 687–694. IEEE.

[Weibel et al., 1998] Weibel, S., Kunze, J., Lagoze, C., and Wolf, M. (1998). Dublin core metadata for resource discovery. *Internet Engineering Task Force RFC*, 2413:222.

[Wright State University, 2008] Wright State University (2008). HTML Microformat for Describing RESTful Web Services and APIs. `http://knoesis.wright.edu/research/srl/projects/hRESTs/#hRESTs`.

[Zeng et al., 2009] Zeng, W., Zhao, Y., and Zeng, J. (2009). Cloud service and service selection algorithm research. In *First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 1045–1048. ACM.

[Zhang and Zhang, 2005] Zhang, J. and Zhang, L. (2005). Criteria analysis and validation of the reliability of Web services-oriented systems. In *Intl. Conf. on Web Services*.