AAAI Technical Report SS-12-04
Intelligent Web Services Meet Social Computing

A Semantic Metadirectory of Services Based on Web Mining Techniques

José Ignacio Fernández-Villamor
Universidad Politécnica de Madrid
Avenida Complutense, 30
28040 Madrid, Spain

Tilo Zemke
Technische Universität Chemnitz
Straße der Nationen, 62
09111 Chemnitz, Germany

Carlos Á. Iglesias and **Mercedes Garijo**
Universidad Politécnica de Madrid
Avenida Complutense, 30
28040 Madrid, Spain

Abstract

In the current web, developers are able to create new applications by composing already existing services from third-party vendors. However, the vast amount of choices, technologies and repositories can make it a tedious task. This paper describes a semantic metadirectory of services that helps in the process of discovering services. We propose a semantic service discovery process and description of existing service repositories, such as Programmable Web and Yahoo Pipes, which are two service repositories which provide plenty of services that can be reused by developers to build new web applications. The challenges behind integrating these repositories involved the problems of defining a common model, identifying relevant data and integrating and ranking the extracted data.

In today's Web, developers enjoy the availability of plenty of services that can be reused to build new web applications. Examples of reusable web services include feeds that provide data from different domains or even telco services. There is also a growing set of tools for the creation of mashups that ease developers the combination of services for application construction, and various service registries, such as Programmable Web or Yahoo Pipes, that help to find mashup components. They can be queried by users in order to search useful applications and services that they can reuse for mashup composition.

Nevertheless, because of this mushrooming of services 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 appropriate service 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 services employ different standards and semantics, thus requiring some study of the documentation by the developer. And third, the services often need to be adapted for their use by the mashup platform in question.

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 seeked in the component, some registries would be more appropriate than others (e.g. some registries might show information about semantics of the service and others not). And again, according to the features seeked, it would be necessary to query external sources to fill up 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 trust).

The Web 2.0 phenomenon caused a vast amount of users collaborate and add contents to the current Web. The European project Open Mashup Enterprise Service Platform for Linked Data in the Telco Domain (OMELETTE)[1] attempts to foster this Web 2.0 phenomenon in web development, empowering users with few or no technical knowledge to create mashup applications. OMELETTE focuses its research on telco mashups, service-level and presentation-level composition, client-server mashups, and automation of component discovery and composition.

This paper describes the semantic service discovery process and description of existing service repositories, such as Programmable Web and Yahoo Pipes, which has been used

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]http://www.ict-omelette.eu

in the OMELETTE project. As said, Programmable Web and Yahoo Pipes are two service repositories which provide plenty of services that can be reused by developers to build new web applications. However, they provide information about services in a non-semantic fashion. The challenges behind integrating these repositories involved the problems of defining a common model, identifying relevant data and integrating and ranking extracted data.

Our research methodology consisted of four steps. First, we have harvested these repositories combining API access and semantic scraping techniques. Second, clustering algorithms have been applied to identify mappings between the different taxonomies present on each repository. Third, services have been automatically described based on Linked Mashups Ontology (LiMOn), a high-level service ontology. Finally, a service ranking algorithm has been defined. As a result, a metadirectory that has been populated with mashup components has been obtained. Because of the clustering process, this metadirectory can handle complex queries by using any of the taxonomies of the original directories. Additionally, it supports queries which combine external sources such as DBpedia thanks to the semantic nature of the resulting data.

The paper is structured as follows. First, we will describe LiMOn, the model used to describe and annotate the metadirectory's components. Afterwards, a metadirectory that makes use of LiMOn is described, as well as the approach that has been followed to populate the metadirectory with actual web components. The next section describes rOMking, the approach employed to rank the different components in the metadirectory. The metadirectory and the data obtained are then evaluated. Finally, related works are summarized and some conclusions of the research done and future works are detailed in the last section.

Linked Mashups Ontology (LiMOn)

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.

With these considerations in mind, we have defined the model presented in Figure 1. Regarding the technical aspect of services, a set of properties allow to cover interface aspects, such as linking to a lower-level component descriptions (e.g. Web Service Modeling Ontology (WSMO), World Wide Web Consortium (W3C) widgets or Web Service Definition Language (WSDL), depending on the nature of 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.

The trust aspect comprises popularity and company trust issues, and includes properties that allow representing users' *rating*, which reflect users' degree of satisfaction with the component. Other properties fall into the company trust by providing means to reference support facilities (i.e. forums and blogs) that the vendor provides to component users. Also, the property *provider* allows to identify the vendor of the component, for any company trust issues involved.

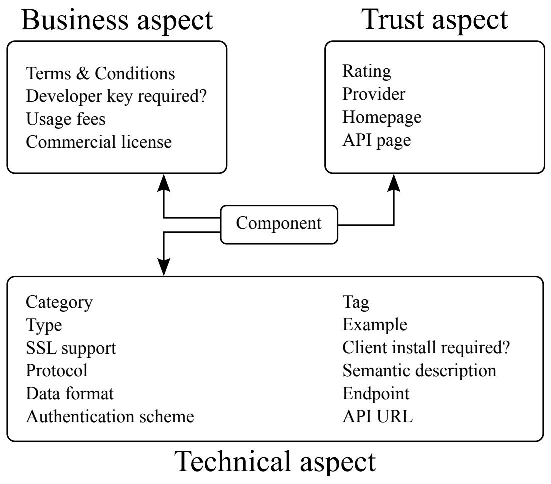

Figure 1: Linked Mashups Ontology (LiMOn)

The business aspect comprises costs, and legal and vendor issues, and is covered by the model through properties such as a cost-aspect property that links to any cost required to use the component. Other properties allow linking to a terms and conditions document, or to a commercial licenses for the usage of the component, etc.

In total, the component repositories of Yahoo Pipes[2], Programmable Web[3], Opera Widgets[4], iGoogle Gadgets[5], AppStore[6], Android Market[7], and Ohloh[8] were analysed to identify relevant properties for the model.

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 considered components that will be included into the metadirectory because of the repositories that have been targeted, again Programmable Web and Yahoo Pipes.

In order to make the components addressable by developers, the metadirectory 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 metadirectory 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

[2]http://pipes.yahoo.com
[3]http://www.programmableweb.com
[4]http://widgets.opera.com
[5]http://www.google.com/ig/directory
[6]http://itunes.apple.com/de/genre/ios/id36
[7]https://market.android.com
[8]http://www.ohloh.net

or WSDL.

Data harvesting and integration

In this section, we will cover how the metadirectory has been populated with components from the targeted repositories 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 HyperText Markup Language (HTML) contents of their web resources and the Resource Description Framework (RDF) data they provide according to LiMOn. To define these mappings we have used the Scraping Ontology[9] (Fernández-Villamor et al. 2011). This approach lets the system to 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 into the metadirectory.

Once the metadirectory is populated with components from the web, a unified categorization scheme is seeked 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 that have been set by the users. 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.

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 said, Programmable Web provides some categorization schemes, with categories such as “Tools”, “Mapping”, or “Sports”. In the case of 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 the next section to provide a uniform interface for querying the metadirectory.

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

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

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 like 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:

$$\begin{aligned} d(bluevia, googlemaps) &\approx 0.1768 \\ d(foursquare, facebook) &\approx 0.3953 \\ d(facebook, bluevia) &\approx 0.4841 \end{aligned} \tag{6}$$

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 to perform a dimensionality reduction on the component space and map the n-dimensional space to a bidimensional 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.

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.

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

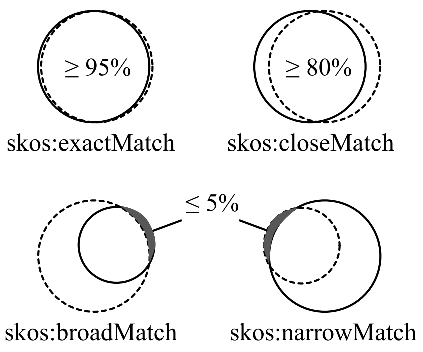

Figure 2: Mapping detection among categories

- 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 2. As shown, Simple Knowledge Organization System (SKOS) (Miles and Bechhofer 2008) 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.

Service ranking

In order to support the developer in choosing the appropriate web services for his mashup we investigated several ranking mechanisms. Any set of web services matched to a specific search term should be provided to the developer in an ordered list, descending in relevance. For that reason, a query interface called *rOMking* has been build in which different ranking approaches have been implemented and compared to each other in an empirical manner. Since we mainly focused on centralities the following definitions, inspired by FolkRank (Hotho et al. 2006), on how to represent our dataset as a graph have been made.

Let S be the set of web services whereas M is the set of mashups in our dataset. We reduced the intuitive hypergraph to an undirected and bipartite graph $G = (V, E)$ letting the set of vertices $V = S \cup M$ be the union of S and M. The set of edges was defined as $E = \{(s, m) \mid Mashup\, m\, uses\, the\, API\, of\, service\, s.\}$. Figure 3 illustrates the structure of this graph.

In particular, the following ranking functions have been investigated:

- C_D: *Degree centrality*, i.e. in our scenario the number of mashups that use a certain service, which directly reflects the popularity.
- C_B: *Betweenness centrality* is a more complex approach that considers the number of shortest paths between two vertices $v \neq u$ a web service s lies on. This metric is an important measure in social network analysis.
- C_C: *Closeness centrality*, implemented with an extension as (Opsahl, Agneessens, and Skvoretz 2010) propose. A vertex v is ranked higher the shorter the geodesic distances between itself and other vertices are, i.e. the *closer* it is to other vertices. Closeness centrality is also an important technique in social network analysis.
- C_E: *Eigenvector centrality* is a very established and successful approach to rank documents in other domains, e.g. PageRank for web resources. The idea behind it is that an API gets ranked higher the more important the mashups that use it are and vice versa.
- PUR: The score, ranging from 0 to 5, represents the rating that each API has on Programmable Web. It which measures the degree of satisfaction the users had when working with a specific API and thus represents a measure of quality to rank components.
- GSO: The amount of hits the Google Search Engine[10] returned querying it for the web service's name and limiting the results to the domain of *StackOverflow*[11], a question-and-answer website specialized on programming topics, is an indicator of how widespread an API is among developers.

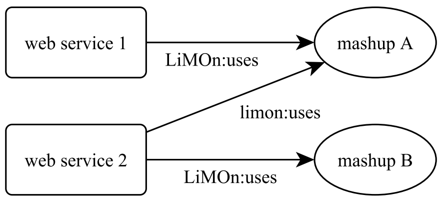

Figure 3: Illustration of the dataset's representation as a graph.

Results and evaluation

The metadirectory contains around 10,000 services and 7,000 mashups, as of a crawling performed in July 2011 on the mentioned repositories of Yahoo Pipes and Programmable Web.

The previously described automatic categorization technique allowed identifying a set of mappings among the different taxonomies. Figure 4 shows some of the mappings. As it can be seen, some categories are defined as sub- or supercategories of others, whilst others are defined as close or exact matches. In the case of Yahoo Pipes repository, the previously described method for automatically building

[10]http://www.google.com

[11]http://www.stackoverflow.com

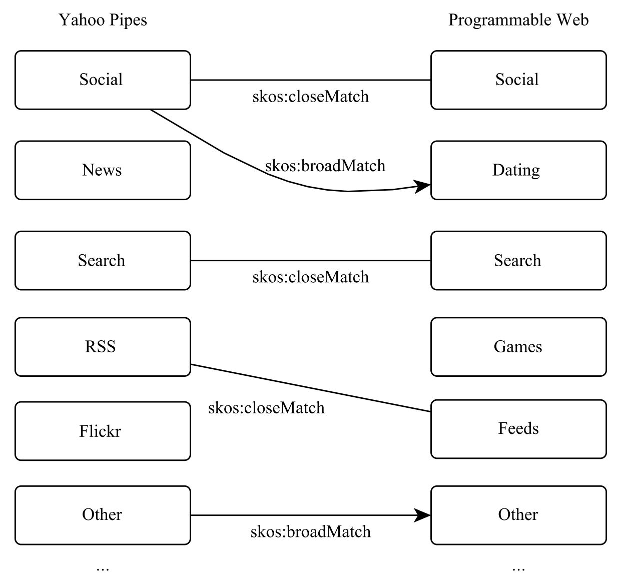

Figure 4: Examples of mappings identified among the different categorization schemes

a taxonomy was used. We executed a clustering algorithm to obtain nine different categories. Then, the resulting categories were applied to Programmable Web's data. The resulting sets were matched to the ones that Programmable Web already provides to identify possible mappings, which resulted in the identified relations among the categories of the different taxonomies.

Evaluation has been performed with a group of three relevance judges, i.e. experienced mashup developers, adapting the methodology of (Küster and König-Ries 2009). The relevance judges independently rated web services in three different dimensions, i.e. functional offer, technological aspects and trust. Afterwards, conflicting judgements have been discussed and the relevance judges agreed on a uniform rating for each web service. Then, the three-dimensional rating was reduced to a one-dimensional one which served as the gain quantifications needed for the *Normalized Discounted Cumulated Gain* (nDCG) (Järvelin and Kekäläinen 2002) measure. The nDCG measure gives a very intuitive sight on how close a ranking is to a specific, supposedly ideal one, in our case the one the relevance judges produced, since $nDCG(n) \in \{0, 1\}$ denotes a score of similarity for an evaluated ranking function until the n-th position in the ranking. In order to limit the amount of relevance judgements we used subsets of S filtering all web services in the metadirectory by means of their names, descriptions and tags, e.g. the subset S_{image} contains all web services that have the term "image" in their name, description or tags.

Table 1 shows the results of our evaluation done in the subset S_{image}. We chose sharp gain quantifications, i.e. powers of 2, as well as a discounting factor of 2. Since the results get less valuable to the developer the higher their position in the ranking is, we also decided to compare the top 5, 10 and 15 ranked results. As can be seen, the ranking functions produce results of considerably similar quality except the Programmable Web user rating PUR. An explanation for PUR's lack of quality may be the lack of votes and therefore missing reliability. Moreover, the reason for the similarity between the centrality measures is their strongly-related nature and the structure of our dataset's graphical representation. For example, the more mashups use a certain web service (C_D) the higher is the probability of being part of a shortest path in G (C_B) and the higher the number of mashups or APIs *close* to it (C_C).

	C_D	C_B	C_C	C_E	PUR	GSO
nDCG(5)	.870	.841	.870	.870	.438	.803
nDCG(10)	.888	.881	.887	.823	.561	.814
nDCG(15)	.878	.918	.930	.918	.634	.851

Table 1: Results of the evaluation in S_{image}

Although runtime performance has not yet been taken into consideration, our experiments showed that degree centrality as well as eigenvector centrality deliver the best cost-benefit ratios among the analysed ranking approaches. While betweenness and closeness centrality suffer from their algorithmic complexity, the traffic caused by GSO does not imply a practical use.

In a next step, we will consider linearly combining the ranking functions in order to maximize the effectiveness of the rankings.

Related work

In this paper, a high-level model for describing the mashup components has been defined. Several approaches deal with web components of different kinds, from services to widgets. There are many initiatives to describe services' interface to allow automation of certain tasks, in the Web Service field (Christensen et al. 2001) (Roman et al. 2005), or in the Representational Stateless Transfer (REST) service area with heavy-weight approaches such as Web Application Description Language (WADL) (Hadley 2006), or more light-weight approaches such as Microservices (Fernández-Villamor, Iglesias, and Garijo 2010), WSMO-Lite (Vitvar, Kopecky, and Fensel 2007), Semantically-Annotated REST (SA-REST) (Sheth, Gomadam, and Lathem 2007) or hRESTS (Wright State University 2008). W3C widgets (Alario-Hoyos and Wilson 2010) define 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 our model, thus existing different standards for each kind of component. Therefore, we make use of them at our model by allowing linking a LiMOn description to a WSDL/WSMO/W3C widget description.

Also, a ranking function is employed to measure components' relevance when querying the metadirectory. (Wang, Chen, and Zhang 2009) use degree, betweenness and closeness centrality in order to analyse the network of Programmable Web and define the importance of a web service with the help of a user-api-network and the degree centrality of the service's neighbourhood. (Ranabahu et al. 2008) present a composite ranking functionality for web services that makes use of user popularity scores. Moreover, they use Alexa traffic rankings in order to determine the popularity of a web service. (Elmeleegy et al. 2008) use estimations of conditional probabilities that a certain concept is added to a given mashup input as basis for their ranking component. Also, exploiting the structure of folksonomies (Hotho et al. 2006) adapted the idea behind PageRank demonstrating their results in the social bookmarking domain.

Furthermore, a metadirectory that integrates several repositories has been built. Some research works that perform similar tasks are available in the current literature. (Wang et al. 2011) mine Programmable Web and build a domain ontology out of the keywords available in the textual descriptions of the services. It helps to validate Programmable Web's categorization scheme. (Blake and Nowlan 2011) perform an automatic categorization of services using the internals of WSDL descriptions, and not just the keywords available in the textual descriptions. However, both works do not explore mappings with other categorizations or service repositories. (Elmeleegy et al. 2008) describe a mashup advisor, which also builds a catalogue of mashup components to exploit in recommendations for mashup development. Unlike our work, they do not consider the integration with different heterogeneous component repositories.

Conclusions and future work

Through this paper, the different challenges that developers face when selecting components for building a mashup have been addressed by an integrated metadirectory of mashup components. Linked Mashups Ontology (LiMOn), a unified model for components, has been defined, and several component repositories have been mined and loaded onto the 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. Also, the components are ranked using rOMking, an index of relevance for querying the metadirectory.

It has been shown how the metadirectory exploits social aspects such as tags, ratings and even activity in forums to improve the experience of accessing the stored data. The metadirectory offers a unified query interface that allows retrieving relevant components through complex queries, involving components of different nature, and allowing integration with the Linked Data cloud.

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

Acknowledgements

This research project was funded by the European Commission under the R&D project OMELETTE (FP7-ICT-2009-5).

References

Alario-Hoyos, C., and Wilson, S. 2010. Comparison of the main alternatives to the integration of external tools in different platforms. In *Proc. International Conference of Education, Research and Innovation, ICERI*, 3466–3476.

Blake, M., and Nowlan, M. 2011. Knowledge discovery in services (kds): Aggregating software services to discover enterprise mashups. *IEEE Transactions on Knowledge and Data Engineering* 23(6):889–901.

Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S.; et al. 2001. Web services description language (wsdl) 1.1.

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 International Conference on*, 337–344. IEEE.

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 International Conference on Agents and Artificial Intelligence*.

Fernández-Villamor, J. I.; Iglesias, C. A.; and Garijo, M. 2010. A vocabulary for the modelling of image search microservices. In *Proceedings of the Fifth International Conference on Evaluation of Novel Approaches to Software Engineering*.

Hadley, M. J. 2006. Web application description language. https://wadl.dev.java.net/wadl20061109.pdf.

Hotho, A.; Jäschke, R.; Schmitz, C.; and Stumme, G. 2006. Folkrank: A ranking algorithm for folksonomies. *Proceedings of FGIR 2006* 2006:2–5.

Järvelin, K., and Kekäläinen, J. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems* 20(4):422–446.

Küster, U., and König-Ries, B. 2009. Relevance judgments for web services retrieval - a methodology and test collection for sws discovery evaluation. In *Proceedings of the 7th IEEE European Conference on Web Services (ECOWS09)*.

Miles, A., and Bechhofer, S. 2008. Skos simple knowledge organization system reference. *W3C Recommendation*.

Opsahl, T.; Agneessens, F.; and Skvoretz, J. 2010. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks 32 (3)* 245–251.

Ranabahu, A.; Nagarajan, M.; Sheth, A. P.; and Verma, K. 2008. A Faceted Classification Based Approach to Search and Rank Web APIs. *2008 IEEE International Conference on Web Services* 177–184.

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

Sheth, A. P.; Gomadam, K.; and Lathem, J. 2007. SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. In *IEEE Computer Society*.

Vitvar, T.; Kopecky, J.; and Fensel, D. 2007. Wsmo-lite: Lightweight semantic descriptions for services on the web. In *Proceedings of the Fifth European Conference on Web Services*, 77–86. Citeseer.

Wang, J.; Zhang, J.; Hung, P.; 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 International Conference on*, 687–694. IEEE.

Wang, J.; Chen, H.; and Zhang, Y. 2009. Mining user behavior pattern in mashup community. *2009 IEEE International Conference on Information Reuse Integration* 126–131.

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