# The role of developers' social relationships in improving service selection

Devis Bianchini, Valeria De Antonellis and Michele Melchiori
*Department of Information Engineering, University of Brescia, Brescia, Italy*

Received 21 April 2016
Accepted 2 May 2016

## Abstract
**Purpose** – Modern Enterprise Web Application development can exploit third-party software components, both internal and external to the enterprise, that provide access to huge and valuable data sets, tested by millions of users and often available as Web application programming interfaces (APIs). In this context, the developers have to select the right data services and might rely, to this purpose, on advanced techniques, based on functional and non-functional data service descriptive features. This paper focuses on this selection task where data service selection may be difficult because the developer has no control on services, and source reputation could be only partially known.

**Design/methodology/approach** – The proposed framework and methodology are apt to provide advanced search and ranking techniques by considering: lightweight data service descriptions, in terms of (semantic) tags and technical aspects; previously developed aggregations of data services, to use in the selection process of a service the past experiences with the services when used in similar applications; social relationships between developers (social network) and their credibility evaluations. This paper also discusses some experimental results regarding the plan to expand other experiments to check how developers feel using the approach.

**Findings** – In this paper, a data service selection framework that extends and specializes an existing one for Web APIs selection is presented. The revised multi-layered model for data services is discussed and proper metrics relying on it, meant for supporting the selection of data services in a context of Web application design, are introduced. Model and metrics take into account the network of social relationships between developers, to exploit them for estimating the importance that a developer assigns to other developers' experience.

**Originality/value** – This research, with respect to the state of the art, focuses attention on developers' social networks in an enterprise context, integrating the developers' credibility assessment and implementing the social network-based data service selection on top of a rich framework based on a multi-perspective model for data services.

**Keywords** Collective knowledge, Data service, Developers social network, Service similarity, Web API, Web application design

**Paper type** Research paper

## 1. Introduction
Modern Enterprise Web Application development can exploit third-party software components, both internal and external to the enterprise, that provide access to huge and valuable data sets, as well as advanced functionalities tested by millions of users, often available as Web application programming interfaces (APIs) (e.g., Google Maps). Availability of data services meeting these requirements is becoming more and more important, as also witnessed by statistics of www.programmableweb.com one of the

International Journal of Web Information Systems
Vol. 12 No. 4, 2016
pp. 477-503
© Emerald Group Publishing Limited
1744-0084
DOI 10.1108/IJWIS-04-2016-0022

most popular Web API repository (Vitvar and Musser, 2010). Data services need to be discovered, selected and merged, but in a Web 2.0 context, their descriptions are often very simple, mainly expressed in terms of categories and (semantic) tags. As far as the enterprise operational environment evolves in terms of actors and their mutual social relationships, social structures, organizational entities and modern Web 2.0 technologies, the Web application design process should evolve coherently, being able to exploit new models and methodologies (Fuxman *et al.*, 2001).

Selection criteria relying on lightweight service descriptions (Blasch *et al.*, 2013; Ceri *et al.*, 2010) might be complemented by considering other perspectives. For instance, the relevance of a data service with respect to a given Web application design project might be high because the service has been already used in similar contexts (i.e., in applications designed with similar data services) and revealed to be useful. Moreover, the importance that a developer gives to past experiences of other developers, who have already used the services for designing their own Web applications, becomes relevant.

In this paper, we present a data service selection framework, apt to providing advanced search and ranking techniques that take into account:

- lightweight data service descriptions, in terms of (semantic) tags and technical aspects on which service implementations rely (e.g., protocols, formats for data exchange);
- previously developed aggregations of data services, to enhance selection by considering services already used in similar contexts (i.e., in applications based on similar data services); and
- a social network of developers, where social relationships represent explicit endorsements among developers concerning their skill in Web application development starting from third-party data service selection.

Moreover, developers can also express votes on data services as included in existing applications, and these votes are used to estimate developers' credibility according to a majority-based approach.

We introduced the idea of engaging developers' development experiences to improve component selection as per Bianchini *et al.* (2013), where we described a *multi-perspective model*, developed considering:

- a *component perspective*, focused on single components and their descriptions;
- an *application perspective*, focused on aggregations of components; and
- an *experience perspective*, focused on developers who used and voted components to build their own aggregations.

Major novel contributions of this paper are as follows:

- We specialized the multi-perspective model in terms of data services with lightweight descriptions (semantic tags and technical features) and their aggregations.
- We extended the experience perspective with the social network of developers.
- We integrated developers' ranking and credibility assessment in the service selection process.

The data service selection is based on the social network of developers and on the analysis of their social relationships. This analysis combined with credibility evaluation of each developer determines a ranking of developers. Specifically, the selection process is organized through the following steps. First, candidate services are selected using similarity metrics based on (semantic) tags, technical features and service co-occurrence in existing applications. Then, candidate services are sorted. Sorting takes into account both the rank of developers who used candidate services to develop their own Web applications and votes assigned to candidate services in the context of these development experiences.

The paper is organized as follows. First, we present in Section 2 an application scenario to make clear the problem of data service selection and provide motivations for our work. In Section 3, we provide some preliminary definitions, and we formalize the problem discussed in this paper. In Section 4, we describe the architecture of our data service selection framework derived from a previous prototype. Section 5 introduces the specialized multi-perspective model for data services. In Section 6, we explain how developer credibility and analysis of social relationships are performed and combined. In Section 7, we describe the metrics for data service selection, based upon the model, and service ranking. Experimental evaluation issues are discussed in Section 8. In Section 9, the state of the art on data service selection is discussed and the cutting-edge features of our approach are highlighted. Finally, conclusions and future work in Section 10 close the paper.

## 2. Application scenario

Let us consider a Web application developer, working for the marketing department of an enterprise that has to build an application that integrates information about potential markets, sales and demographic data. This application could be implemented by merging data coming from sources internal to the enterprise (e.g., information about the target clients) and external data sources (e.g., providing information about demographic data), made available as data services.

The developer's tasks can be intuitively organized into two main phases: first, the developer has to select the right data services and might rely, to this purpose, on advanced techniques, based on functional and non-functional data service descriptive features; therefore, data integration and querying over multiple sources have to be addressed, to deploy the application required by the marketing department.

In this paper, we focus on the former phase. Data service selection may be difficult because the developer has no control on services, and source reputation could be only partially known. These difficulties are mainly because of the high number and heterogeneity of available data services over the Web and to their often unknown origins. In this context, it is frequent that a developer:

- searches for advices from other developers (of the same enterprise or different ones); and
- looks for votes/ratings assigned by other developers to data services.

An approach that exclusively relies on the latter is error-prone, as no information is available on the credibility of people who rated the data services. Although some techniques for credibility assessment have been proposed in the literature (Bianchini *et al.*, 2013; Malik and Bouguettaya, 2009), a typical Web application design project prevents from exclusively using these techniques (Hertzum, 2014; Schafermeier and Paschke, 2011). It is reasonable to assume that developer's preference is for experts who can be contacted/engaged, because some mutual social or organizational relationships exist among them. These considerations are confirmed by studies, for example Hertzum (2014) conducted on employees of medium- and large-sized companies, that demonstrate how employees often prefer to talk to experts instead of using ratings or comments spread over the network. Therefore, social aspects play an important role by lowering the effort in the identification of candidate data services, that are suitable to be aggregated in the Web application to be designed. In the rest of the paper, we will assume the following preliminary definitions.

## 3. Preliminary definitions

Formally, we define the concepts data services and service aggregations as follows.

Definition 1. *We define a data service s (hereafter, also service) as an operation/method/query to access data of a source* $\Sigma$*, whose underlying data schema might be unknown to those who use the service. Within the scope of this paper, we model a service s as* $\langle n_s, URL_s, \{t_s\}, \{f_s\}\rangle$*, where:* $n_s$ *is the service name;* $URL_s$ *is the service endpoint;* $\{t_s\}$ *is a set of tags;* $\{f_s\}$ *is a set of technical features (e.g., programming languages and data formats compatible with the service). We denote with* $\mathcal{S}$ *the overall set of available services.*

Our definition of data services is suitable for describing resource-oriented services (i.e., RESTful ones). Concerning SOAP (Simple Object Access Protocol) services which present an operation-oriented description (e.g., WSDL [Web Service Definition Language]), we consider only data they work on, represented through tags as a simplification of service descriptors originally introduced in our first works on Web services' similarity analysis (De Antonellis *et al.*, 2003). We admit different ways for assigning tags to services:

- keywords automatically extracted from the service name, names of I/O parameters and related textual descriptions through the application of text-mining techniques (Gupta and Lehal, 2009); and
- tags manually assigned by developers who used the service to design their own applications.

Examples of data services are the methods of a Web API (for instance, the method of GeoData Demographics API[1] that provides demographic data for a given zone), queries formulated by using search-specific languages (such as Yahoo! Query Language, https://developer.yahoo.com/yql/), services delivering data in tabular format (e.g., Google Fusion Tables; http://tables.googlelabs.com/) or in row format (such as Factual, http://www.factual.com). Data services are usually wrapped as Web services, that can be implemented according to different styles (e.g., using REST or SOAP).

Definition 2. *A service aggregation describes a set of services that can be used to deploy an enterprise Web application. An aggregation g is modeled as a triple* $\langle n_g, \mathcal{S}(g), d\rangle$*, where:* $n_g$ *is the aggregation name;* $\mathcal{S}(g) = \{s_1,\ldots,s_n\}$ *is the set of data services used in g;* $d \in \mathcal{D}$ *is the developer who designed the Web application by composing services in g. We denote with* $\mathcal{G}$ *the overall set of service aggregations, that is,* $g \in \mathcal{G}$*, and with* $\mathcal{G}(s)$ *the set of aggregations where s has been included.*

As remarked in Section 2, the development process requires service selection and then service composition/integration. This is why we distinguish between applications and data service aggregations.

**Problem statement**. Given a developer $d \in \mathcal{D}$, who is designing a new Web application starting from a set of available data services $\mathcal{S}$, given the set $\mathcal{G}[d_k]$ ($\forall k = 1 \ldots n$) of data service aggregations abstracting the applications designed in the past by developer $d_k \in \mathcal{D}$. Our aim is supporting $d$ in performing data service selection, by proposing an ordered set of candidate data services $\mathcal{S}^* \subseteq \mathcal{S}$. Ordering takes into account the past experience in $\mathcal{G}[d_k]$ for each developer $d_k$ and weighs the experience based on social relationships established between developers to form a social network.

## 4. Architecture of the WISeR framework

WISeR (Web apI Search and Ranking) presented by Gupta and Lehal (2009) is a framework we originally implemented to provide advanced Web APIs' selection facilities by enabling application developers to actively take part in the semantic tagging and rating of Web APIs. The aim is allowing developers to share experiences obtained when producing their own Web APIs' aggregations and mashups. In particular, we adapt similarity metrics based on semantic tag matching, used by WISeR for Web API search, and we introduce similarity based on technical features. Then, we combine these metrics with developers' ranking metrics to implement the data service selection.

To enable this social network-based selection of data services, we extend the WISeR framework and the resulting architecture as shown in Figure 1.

First, the resulting WISeR is based on a multi-layered model, developed considering different description perspectives where components are data services in the extended version of WISeR:

- a component perspective, focused on single components and their descriptions;
- an application perspective, focused on aggregations of components; and
- an experience perspective, focused on developers who used and voted components to build their own aggregations.

The framework allows different kinds of search, either for single isolated services or for services needed to complete/update an existing application. Furthermore, two modalities of search are provided: simple and proactive. In the *simple search*, the user receives suggestions about relevant data services after explicitly specifying search features (e.g., tags, required technical features and so on). In the *proactive search*, that is the most explorative search modality, the developer does not have in mind the services of interest, or he/she just provides a partial specification of what he/she is looking for, and the framework proactively suggests candidate data services trying to complete the aggregation that is being developed. The framework is equipped with proper wizards that guide the developer in formulating the request depending on the desired kind of search. A set of similarity metrics, based on the three perspectives, has been defined in WISeR to quantify the compliance of available data services with respect to the specified request:

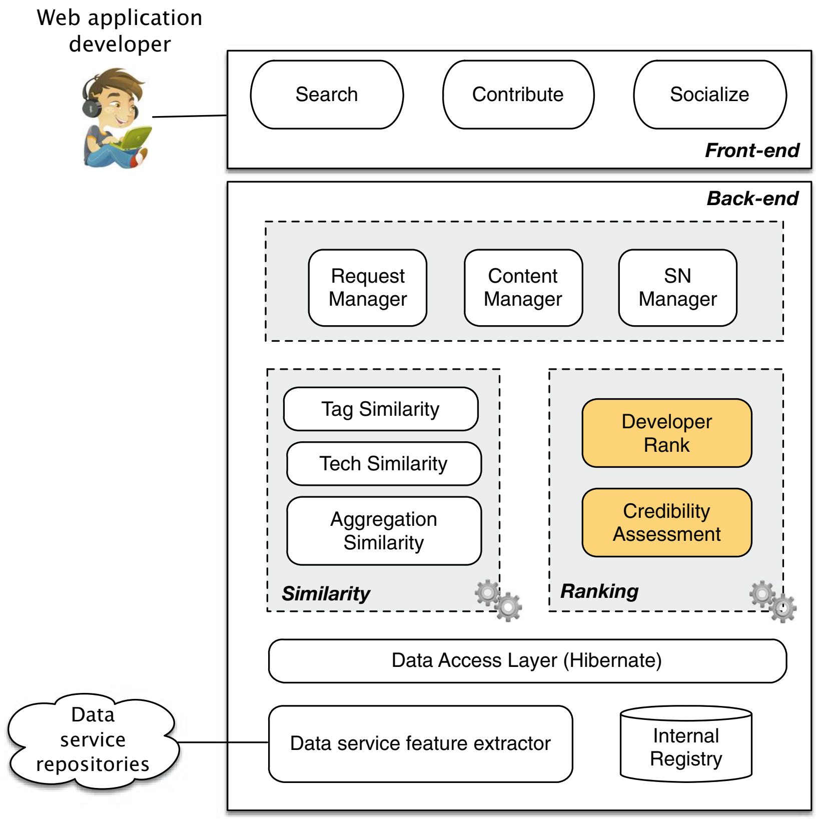


**Figure 1.**
The framework architecture

(1) the *tag similarity*, to denote the similarity between the request and each candidate service based on tags associated with both of them;
(2) the *technical feature similarity*, to denote the similarity between the request and each candidate service based on technical features; and
(3) the *aggregation similarity*, to denote the similarity between the request and each candidate service based on average similarity between the aggregation that is being developed and aggregations where candidate services have been used in the past.

An overall similarity between request and each candidate service is then defined as a combination of the above similarities. The adapted versions of these metrics and the way they are combined will be defined formally in the Section 7 of this paper.

Services are kept within public and enterprise repositories, where they are advertised. Proper wrappers have been designed to collect relevant service features (e.g., tags already assigned within such repositories) and to maintain a reference toward original service, thus ensuring full compatibility. In particular, we have currently implemented a wrapper that relies on the http://api.programmableweb.com methods for accessing the ProgrammableWeb API repository contents to make available data services implemented as Web APIs methods. Wrappers are invoked within the Data

Service Features Extractor (Figure 1). Service descriptions according to Definition (1) are stored within the Internal Registry shown in the figure.

Ratings of services, information about aggregations they belong to and developers' social relationships are saved within the Internal Registry as well. Developers interact with the framework through the front-end developed as Web application and that provides the following functionalities:

- *Search*, to search for services.
- *Contribute*, to tag and assign votes to services.
- *Socialize*, to manage social relationships between developers.

The Manager modules act as controllers for the front-end functionalities. The core modules we added to the existing WISeR framework, namely, Developer rank and Credibility Assessment, will be detailed in the next sections.

## 5. The three-layer model

The model we propose in this paper includes three layers, which specialize the perspectives presented in the previous section. Moreover, the model extends the experience perspective by also considering the social network of developers. The model is shown in Figure 2, and the layers are defined as follows:

(1) *A data service layer*: Where data services are described and tagged; data services are described according to Definition (1).

(2) *A service aggregation layer*: Where usage experiences of data services, in terms of *data service aggregations*, are described according to Definition (2).

(3) *A social layer*: Where developers are registered, together with their declared social relationships, to form a social network.

In particular, the model layers are formalized as follows.

Definition 3. *The data service layer is formed by the set $S = \{s_j\}$ of the available data services, where $s_j$ is a service defined according to Definition (1).*

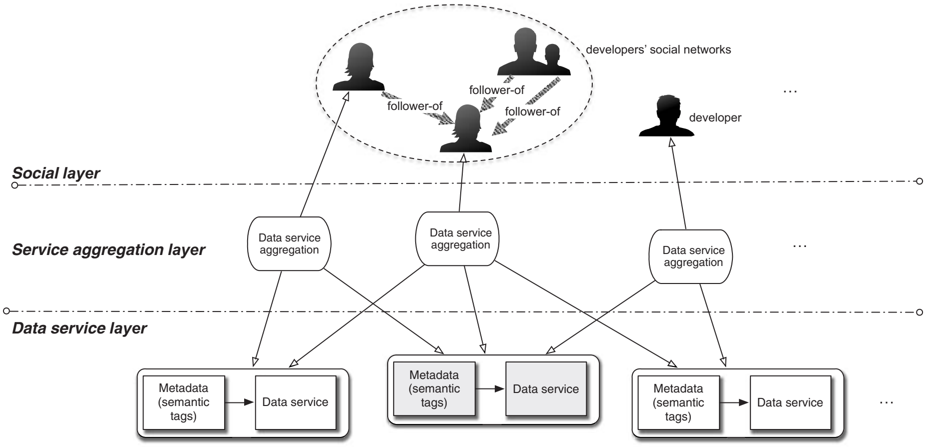


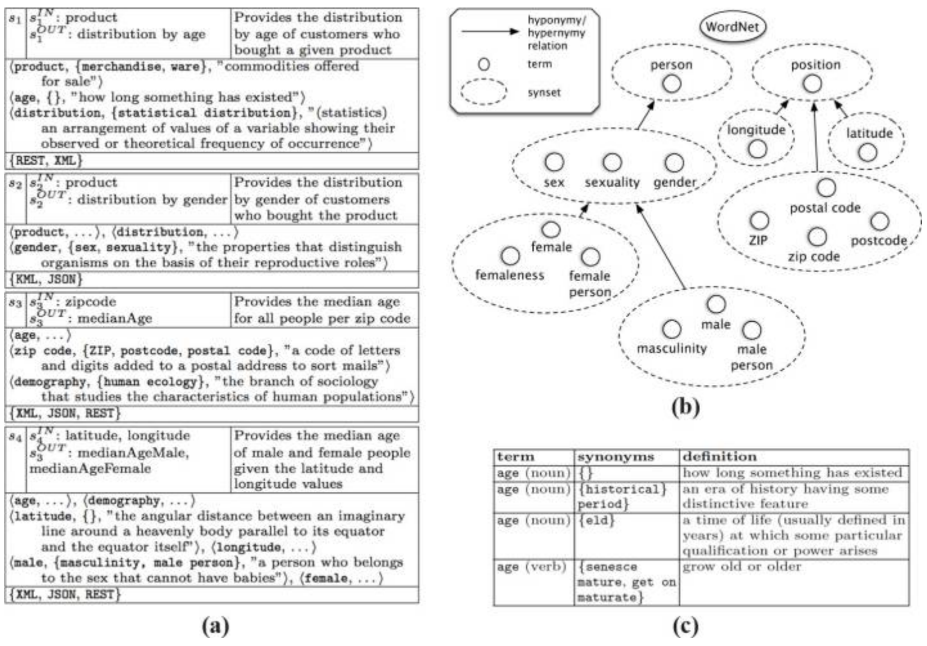
**Figure 2.** The multi-layered model for data service selection

In this layer, a data service $s$ is associated with some *metadata* that, in the current version of the model, contains (semantic) tags and technical features referring to the whole service, used to enable coarse-grained search of the services. Semantic tagging is supported within the WISeR framework through sense disambiguation facilities based on WordNet. In WordNet, the meaning of terms is defined by means of *synsets*. Each synset has a human-readable definition and a set of synonyms. Starting from a tag specified by the developer, WordNet is queried, all the synsets that contain the term are retrieved and the developer is asked to select the intended meaning. Each semantic tag is a triplet, composed of:

- the term itself, extracted from WordNet;
- the set of all the terms in the same synset selected by the developer (synonyms); and
- the human-readable definition associated with the synset.

In WordNet, synsets are related by *hyponymy/hypernymy relations*, used to represent the specialization/generalization relationship between two tags. Such a sense disambiguation process is supported by a wizard (Bianchini *et al.*, 2012).

Example. In Figure 3(a), four sample data services are listed: the first two, namely, $s_1$ and $s_2$, are fictitious services for the marketing department of the enterprise in the running example described above; the others, namely, $s_3$ and $s_4$, are real data services provided within the GeoData Demographics API. For each data service, inputs, outputs, a short textual description and semantic tags disambiguated using WordNet and technical features are given. Note that some tags are obtained from the inputs $s_{IN}$ and outputs $s_{OUT}$ of services as explained in Section 3. In Figure 3(b), a portion of WordNet thesaurus is shown where terms that include also tags for the sample data services are



|  |  |  |
|---|---|---|
| $s_1$ | $s_1^{IN}$: product<br>$s_1^{OUT}$: distribution by age | Provides the distribution by age of customers who bought a given product |
| ⟨product, {merchandise, ware}, "commodities offered for sale"⟩<br>⟨age, {}, "how long something has existed"⟩<br>⟨distribution, {statistical distribution}, "(statistics) an arrangement of values of a variable showing their observed or theoretical frequency of occurrence"⟩ | | |
| {REST, XML} | | |
| $s_2$ | $s_2^{IN}$: product<br>$s_2^{OUT}$: distribution by gender | Provides the distribution by gender of customers who bought the product |
| ⟨product, ...⟩, ⟨distribution, ...⟩<br>⟨gender, {sex, sexuality}, "the properties that distinguish organisms on the basis of their reproductive roles"⟩ | | |
| {KML, JSON} | | |
| $s_3$ | $s_3^{IN}$: zipcode<br>$s_3^{OUT}$: medianAge | Provides the median age for all people per zip code |
| ⟨age, ...⟩<br>⟨zip code, {ZIP, postcode, postal code}, "a code of letters and digits added to a postal address to sort mails"⟩<br>⟨demography, {human ecology}, "the branch of sociology that studies the characteristics of human populations"⟩ | | |
| {XML, JSON, REST} | | |
| $s_4$ | $s_4^{IN}$: latitude, longitude<br>$s_3^{OUT}$: medianAgeMale, medianAgeFemale | Provides the median age of male and female people given the latitude and longitude values |
| ⟨age, ...⟩, ⟨demography, ...⟩<br>⟨latitude, {}, "the angular distance between an imaginary line around a heavenly body parallel to its equator and the equator itself"⟩, ⟨longitude, ...⟩<br>⟨male, {masculinity, male person}, "a person who belongs to the sex that cannot have babies"⟩, ⟨female, ...⟩ | | |
| {XML, JSON, REST} | | |

(a)

| term | synonyms | definition |
|---|---|---|
| age (noun) | {} | how long something has existed |
| age (noun) | {historical period} | an era of history having some distinctive feature |
| age (noun) | {eld} | a time of life (usually defined in years) at which some particular qualification or power arises |
| age (verb) | {senesce mature, get on maturate} | grow old or older |

(c)

**Figure 3.**
(a) Sample data services for the running example; (b) a portion of WordNet thesaurus used for semantic disambiguation of tags; (c) WordNet synsets and corresponding definitions for the age term

related by hyponymy/hypernymy relations. Finally, in Figure 3(c), examples of WordNet definitions for the Age term are listed.

Definition 4. *The data service aggregation layer is formed by the set* $\mathcal{G} = \{g_i\}$ *of the available aggregations, where* $g_i$ *is a service aggregation defined according to Definition (2).*

In this layer, past usage experiences of data service selection are stored, in the form of *data service aggregations*.

The social layer includes the social network of developers defined as follows.

Definition 5. *The social network of developers is a pair* $SN = \langle \mathcal{D}, \varepsilon \rangle$, *where: (a)* $\mathcal{D}$ *is the set of developers; (b)* $\varepsilon$ *is a set of follower-of relationships between developers, defined as* $\varepsilon = \{d_i \xrightarrow{f} d_j | d_i, d_j \in \mathcal{D}\}$, *where* $d_i \xrightarrow{f} d_j$ *indicates that* $d_i$ *explicitly declares to be inclined to learn from the choices made in the past by* $d_j$ *for web application design purposes.*

Definition 6. *Each developer* $d_i \in \mathcal{D}$ *in the network is modeled as* $\langle \mathcal{G}(d_i), \mathcal{D}^* \rangle$, *where* $\mathcal{G}(d_i) \subseteq \mathcal{G}$ *is the set of aggregations designed by* $d_i$ *in the past,* $\mathcal{D}^* \subseteq \mathcal{D}$ *is the set of other developers, whom* $d_i$ *declares to be inclined to learn from to design Web applications, that is,* $\mathcal{D}^* = \{d_k | d_i \xrightarrow{f} d_k \in \varepsilon\}$.

The set of *follower-of* relationships determines the network topology. This kind of relationship, for example, is used within the ProgrammableWeb and Mashape repositories. The same viewpoint is assumed by our framework that, in the current version, is based on the ProgrammableWeb contents. Other kinds of social applications for developers, such as GitHub, are more focused on code sharing and collaborative coding. In the current version of our approach, the *follower-of* relationships are set after an explicit endorsement of developers. An overview of the network of social relationships between developers might reveal different kinds of topologies of social relationships that can be recognized (dos Santos *et al.*, 2010) and representing different design scenarios. In fact, the developers' social network can be represented as one or more directed graphs, as shown in Figure 4, where a graph can assume different topologies. For example, it can be restricted to a hierarchy. On the other hand, a peer-based network has a topology where a hierarchy is not present; there can be pairs of developers that mutually follow each other, and this is typical of modern enterprises in a totally collaborative and open context. An example is the network in Figure 4(a). A third kind of topology, Figure 4(b), represents a hybrid case, where a developer is or has been involved in different Web application design projects and, may be depending on

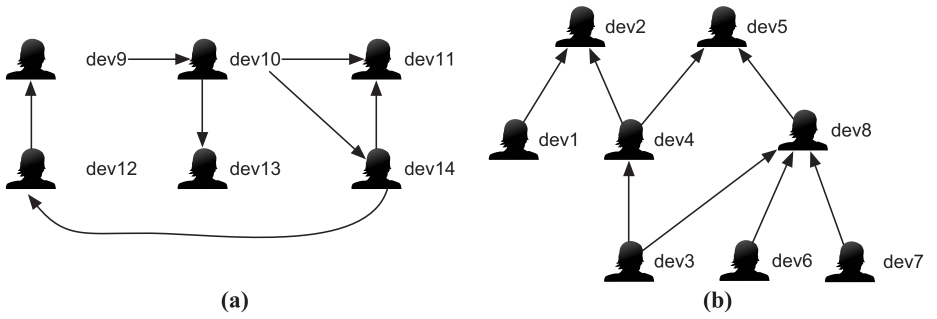


**Figure 4.** Sample social network of developers, that present a peer-based (a) and hybrid (b) topology

the particular application domain, can follow different reference developers (consider, as an example, dev3, who declares to follow both dev4 and dev8).

## 6. Social-based evaluation of developers

### *6.1 Service request*

A developer, who is responsible for a project based on data services, hereafter denoted as the *requester* $d^r$, formulates the request, denoted with $s^r$, that is matched against the set $\mathcal{S}$ of available data services. The aim is at finding data services to complete/expand the application that is being designed. Therefore, the request $s^r$ is formulated as a set of desired (semantic) tags, a set of data services, that have been already included in the application and a set of desired technical features. Formally, $s^r = \langle \{t^r\}, g^r, \{f^r\} \rangle$, where $\{t^r\}$ is the set of tags, $g^r = \{s_1, s_2, \ldots s_n\}$ is the set of already selected data service descriptions and $\{f^r\}$ is a set of technical features.

For example, the following request $s^r$ is formulated to find a demography data service, annotated with a postal code, to produce a distribution of people by sex. The service will be used in a Web application, that is being designed and already contains data services $s_1$ and $s_2$ (Figure 3). The application will be written in PHP and a REST service is looked for:

$$\begin{aligned}
s^r = \langle t_1^r &= \{\langle \text{postal code}, \{\text{zip code, ZIP, postcode}\}\text{”, a code of} \\
&\quad \text{letters and digits added to a postal address to sort mails”}\rangle\}; \\
t_2^r &= \{\langle \text{sex}, \{\text{gender, sexuality}\},\text{” the properties that distinguish} \\
&\quad \text{organisms on the basis of their reproductive roles”}\rangle\}; \\
t_3^r &= \{\langle \text{demography}, \{\text{human ecology}\},\text{” the branch of sociology that} \\
&\quad \text{studies the characteristics of human populations”}\rangle\}; \\
g^r &= \{s_1, s_2\}\rangle \\
f^r &= \{\text{PHP, REST}\}\rangle
\end{aligned}$$

It is worth noting that the search for a single data service (or for the first data service to be included in the new Web application that is being designed) is a particular case of the same definition, that is, $s^r = \langle \{t^r\}, g^r, \{f^r\} \rangle$, where $g^r = \phi$.

Answering the service request $s^r$ is based in our approach on the following phases:

- developers’ credibility evaluation;
- developers’ ranking;
- service selection; and
- service ranking.

In the following, we detail these phases, and Figure 5 provides a short summary, where each phase uses the output of previous one, and references to relevant sections.

### *6.2 Developers’ credibility evaluation*

In WISeR, a developer may assign votes to services used in the applications. In particular, because developers exchange their experiences in using services, votes become an enabling feature to this purpose. Following this perception, for example, all of the most popular Web API repositories (and, among them, ProgrammableWeb) include a rating system. Votes are assigned by the developers with the adoption of the NIH nine-point Scoring System[2]. This scoring system has few rating options (only nine) to increase potential reliability and consistency and with enough range and appropriate anchors to encourage developers to use the full scale (from poor, to denote completely useless and wrong services, to exceptional, to denote services with very good performances and functionalities and easy to use). During the rating, the developer is provided with the set of options and corresponding meaning (such as, for example, the ones associated with poor (9) and exceptional (1) options, as mentioned above). These options are then uniformly distributed over the [0,1] interval so that the highest vote is corresponding to 1 and the lowest to 0. Our approach introduces an important distinction for service rating, compared to existing systems, because it takes into account the aggregation in which services have been evaluated according to the following definition.

Definition 7. *Given a service $s_j \in \mathcal{S}$, we denote with $v(s_j, g_k, d_i) \in [0,1]$ the vote assigned to the service $s_j$ by a developer $d_i \in \mathcal{D}$ with reference to the aggregation $g_k \in \mathcal{G}$ in which $s_j$ has been used. Votes are assigned according to the NIH nine-point Scoring System and mapped to the* [0,1] *interval.*

Aggregation-contextual rating helps in properly weighing votes assigned to services. For instance, different votes might be assigned to the Amazon APIs, depending on the aggregations where they have been used. When a developer is looking for the average of votes assigned to a service, relevant votes to be considered are those that have been assigned with reference to aggregations that are similar to the ones that is being developed (according to the aggregation similarity, *AggSim*(), mentioned in Section 4).

It is important to estimate the credibility of a developer, who expresses votes or judgments. Therefore, we include credibility evaluation techniques with respect to which we introduce the notion of aggregation-contextual rating. Votes are used in the Credibility Assessment Module (Figure 1). The basic idea is that, if the reported vote does not agree with the majority opinion, the developer's credibility is decreased, otherwise it is increased. Suppose the developer $d_i$ assigned some votes to the service $s_j$ with reference to the aggregations $g_1, g_2, \cdots, g_t$, respectively. For each $g_m$ in these aggregations, we consider the set $\mathcal{A}g_m$ of aggregations $g_o \in \mathcal{G}$ that have similarity $AggSim(g_o, g_m)$ above a given threshold.

A k-mean clustering algorithm is then applied to the set of votes assigned by other developers to the service $s_j$ in the context of aggregations in $\mathcal{A}g_m$. By grouping similar ratings together, we define the majority opinion in the context of specific aggregations. The rationale for clustering votes can be explained with the help of an example: if a service receives votes 1,1,1,2,9,9,8 (considered before their normalization to [0,1]) and we adopt an average-based model, we obtain an overall rating of 4.4 out of 9. Actually, the

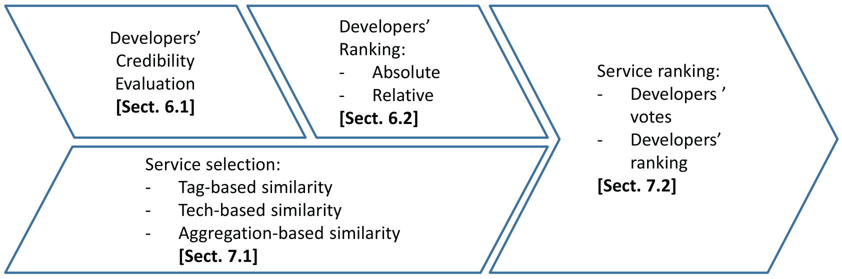


**Figure 5.** Services selection and ranking phases in the extended version of WISeR

average rating does not well describe the depicted situation, where most of the voters give very high votes. So we choose a majority opinion approach inspired by the Malik and Bouguettaya (2009). The majority opinion on $s_j$ is hence represented by the most densely populated cluster, whose centroid is taken as the majority vote:

$$M(s_j) = centroid\,(max_{i=1}^{k}(C_i)) \tag{1}$$

where $C_i$ is the i-th cluster, $k = |\{C_i\}|$ is the total number of clusters, $max()$ returns the cluster with the largest membership and $centroid()$ computes the centroid of the cluster. The number $k$ of desired clusters is set to $\lceil \sqrt{N/2} \rceil$, where $N$ is the number of considered votes and $\lceil x \rceil$ is the smallest integer not less than $x$. Therefore, considering a developer $d_i$, having a credibility $c_n(d_i)$ after he already assigned $n$ votes. Considering also a new vote $v(s_j, g_m, d_i)$ assigned by $d_i$ to a service $s_j$ when used within an aggregation $g_m$, then the new credibility value for $d_i$ is computed as follows:

$$c_{n+1}(d_i) = \frac{c_n(d_i) \cdot n + (1 - |M(s_j) - v\,(s_j, g_m, d_i)|)}{n + 1} \in [0, 1] \tag{2}$$

According to equation (2), if the vote $v(s_j, g_m, d_i) \in [0,1]$ differs from the centroid $M(s_j) \in [0,1]$, then the term $1 - |M(s_j) - v(s_j, g_m, d_i)|$ tends to be zero; therefore, $c_{n+1}(d_i) < c_n(d_i)$ (the decrement is controlled by denominator $n + 1$, to avoid the case in which a designer loses too quickly his/her credibility for few assigned votes that are not aligned with majority opinion). Vice versa, if the vote $v(s_j,g_m,d_i)$ is close to $M(s_j)$, then the term $1 - |M(s_j) - v(s_j, g_m, d_i)|$ tends to 1 and $c_{n+1}(d_i) > c_n(d_i)$ until $c_{n+1}(d_i)$ reaches 1 (maximum credibility). Initial values $c_0(d_i)$ are set to 0.5. However, note that credibility of a developer with a high number \em{n} of votes is quite stable according to Equation (2). That is, even if his vote is different from the majority, his credibility according to Equation (2) decreases of a small amount. In fact, this type of vote is not necessarily describing an incoherent behavior of the developer and could be the result of a recent change in the service conditions or quality perceived by the voter. Generally speaking, defining credibility in a more general way and performing its assessment is a complex problem (Malik and Bouguettaya, 2009) and here we give a definition of credibility based on the information available in the context we are considering.

### *6.3 Ranking of developers*

Let us suppose that $d^r$ has formulated the request $s^r$. Consider two candidate services $s_1$ and $s_2$, used by two developers $d_1$ and $d_2$, respectively, in aggregations that are similar to the one in $s^r$. If $s_1$ and $s_2$ are equally relevant with respect to $s^r$, then $s_1$ will be ranked better than $s_2$ if the experience of $d_1$, who used $s_1$, is ranked better than the experience of $d_2$, who used $s_2$. The point here is at ranking the experience of developers $d_1$ and $d_2$.

Rank of a developer $d_i \in \mathcal{D}$ is computed as the product of two different rankings, according to the following formula:

$$dr\,(d_i) = \rho_{rel}^{d^r}(d_i) \cdot \rho_{abs}(d_i) \in [0, 1] \tag{3}$$

where:

- a *relative ranking* $\rho^{d}_{rel}(d_i) \in [0,1]$ ranks developer $d_i$ based on the *follower-of* relationships between $d_i$ and $d^r$ (this rank is introduced to take into account the viewpoint of $d^r$, who explicitly declared to learn from other developers to select the right service); and
- an *absolute ranking* $\rho_{abs}(d_i)$ is based on the overall network of developers, to take into account the authority of $d_i$ in the network independently of the developer $d^r$, who issued the request.

*6.3.1 Relative ranking.* In particular, the relative ranking $\rho^{d^r}_{rel}(d_i)$ is inversely proportional to the distance $\ell(d^r, d_i)$ between $d^r$ and $d_i$, in terms of *follower-of* relationships, that is:

$$\rho^{d^r}_{rel}(d_i) = \frac{1}{\ell(d^r, d_i)} \in [0,1] \tag{4}$$

If there is no a path from $d^r$ to $d_i$, $\ell(d^r, d_i)$ is set to the length of the longest path of *follower-of* relationships that relate $d^r$ to the other developers, incremented by 1, to denote that $d_i$ is far from $d^r$ more than all the developers within the $d^r$ sub-network. Consider, for example, the network shown in Figure 4, where the developer dev3 is the requester and has to choose among services that have been used in the past by the developers dev4, dev5, dev6, dev8 and dev11, whose *follower-of* relationships are depicted in the figure. In the example, $\ell$(dev3, dev4) = $\ell$(dev3, dev8) = 1, $\ell$(dev3, dev5) = 2 and $\ell$(dev3, dev6) = $\ell$(dev3, dev11) = 4 + 1 = 5.

*6.3.2 Absolute ranking.* The absolute ranking $\rho_{abs}(d_i) \in [0,1]$ is evaluated no matter the viewpoint of the requester $d^r$. This ranking is composed of two different parts. The first one depends on the number of aggregations designed by $d_i$ and the second one depends on the topology of the network of other developers who declared their interest for $d_i$ past experiences, that is:

$$\rho_{abs}(d_i) = \frac{1-\alpha}{|\mathcal{D}|} \cdot |\mathcal{G}(d_i)| + \alpha \cdot \sum_{j=1}^{n} \frac{c(d_j) \cdot \rho_{abs}(d_j)}{F(d_j)} \tag{5}$$

This expression is an adaptation of the known PageRank metrics for Web search engines to the context we are considering. The original PageRank calculates an authority degree for Web pages based on the incoming links to pages. The value $\rho_{abs}(d_i)$ represents the probability that a developer will consider the example given by $d_i$ in using a service for designing an enterprise Web application. Therefore, $\Sigma_i \rho_{abs}(d_i) = 1$. Initially, all developers are assigned with the same probability, that is, $\rho_{abs}(d_i) = 1/|\mathcal{D}|$. Furthermore, at each iteration of the absolute ranking computation, the absolute rank of a developer $d_j$, such that $d_j \xrightarrow{f} d_i$, is "transferred" to $d_i$ according to the following criteria:

(1) if $d_j$ follows more developers, his/her rank is distributed over all these developers, properly weighted considering the credibility $c(d_j)$ of $d_j$ [see the second term in equation (5), where $F(d_j)$ is the number of developers followed by $d_j$]; and

(2) a contribution to $\rho_{abs}(d_i)$ is given by the experience of $d_i$ and is therefore proportional to the number of aggregations designed by $d_i$ [see the first term in equation (5)].

A damping factor $\alpha \in [0,1]$ is used to balance contributions explained in Steps (1) and (2) above. At each step, a normalization procedure is applied to ensure that $\Sigma_i \rho_{abs}(d_i) = 1$.

The algorithm actually used to recursively compute equation (5) is similar to the one applied for PageRank. In particular, denoting with $\rho_{abs}(d_i,\tau_N)$ the N-th iteration in computing $\rho_{abs}(d_i)$, with $\mathbf{DR}(\tau_N)$ the column vector whose elements are $\rho_{abs}(d_i,\tau_N)$, we have:

$$\mathbf{DR}\,(\tau_{N+1}) = \frac{1-\alpha}{|\mathcal{D}|}\cdot\begin{bmatrix}|\mathcal{G}(d_1)|\\|\mathcal{G}(d_2)|\\\vdots\\|\mathcal{G}(d_n)|\end{bmatrix} + \alpha\cdot\mathbf{M}\cdot\mathbf{DR}(\tau_N) \quad (6)$$

where $\mathbf{M}$ denotes the adjacency matrix properly modified to consider credibility, that is, $M_{ij} = c(d_j)/F(d_j)$ if $d_j \xrightarrow{f} d_i$, zero otherwise. As demonstrated in PageRank, computation formulated in equation (6) reaches a high degree of accuracy within only a few iterations.

For example, let us consider Table I, that lists developers' features considered for the running example and approach validation described in Section 8. In particular, we set $\alpha = 0.6$. At time $\tau_0$, we set $\rho_{abs}(d_i) = 1/|\mathcal{D}| = 0.0714$ for all $d_i$. During the next iteration:

$$\rho_{abs}(\text{dev4},\tau_1) = [\frac{1-0.6}{14}\cdot 4 + 0.6\cdot\frac{1.0\cdot 0.0714}{2}] = 0.1357$$

Similarly, $\rho_{abs}(\text{dev8},\tau_1) = 0.1299$. After each iteration, normalization is applied to have $\Sigma_i \rho_{abs}(d_i) = 1$. In the example, after five iterations, the error measured as a Euclidean norm of the vector $\mathbf{DR}(\tau_5) - \mathbf{DR}(\tau_4)$ is less than 0.001. And we obtain $\rho_{abs}(\text{dev4}) = 0.0997$ and $\rho_{abs}(\text{dev8}) = 0.0801$.

| *Developer ($d_i$)* | $|\mathcal{G}(d_i)|$ | *Credibility $c$ ($d_i$)* |
|---|---|---|
| dev1 | 5 | 1.0 |
| dev2 | 3 | 0.7 |
| dev3 | 2 | 1.0 |
| dev4 | 4 | 0.1 |
| dev5 | 3 | 0.7 |
| dev6 | 2 | 0.2 |
| dev7 | 2 | 1.0 |
| dev8 | 2 | 0.2 |
| dev9 | 2 | 0.7 |
| dev10 | 3 | 0.6 |
| dev11 | 3 | 0.7 |
| dev12 | 2 | 0.9 |
| dev13 | 1 | 0.5 |
| dev14 | 2 | 0.7 |

**Table I.**
Details about developers' features, considered for the running example

## 7. Data service selection

### *7.1 Data service selection metrics*

Different metrics are applied and properly combined to find the set of candidate data services, given the request $s^r$ formulated as shown above. The overall similarity between $s^r$ and each available data service $s \in \mathcal{S}$, denoted with $Sim(s^r, s) \in [0,1]$, is used to filter out irrelevant data services and is computed by combining two matching techniques, based on (semantic) tags and available data service aggregations, adapted from Bianchini *et al.* (2013).

*7.1.1 Tag matching.* The closeness between the request and each available data service according to their associated (semantic) tags, denoted with $Sim_{tag}(\{t^r\}, \{t\})$, is computed by evaluating the affinity between pairs of (semantic) tags, one from the set $\{t^r\}$ assigned to the request and one from the set $\{t\}$ of tags assigned to the data service $s \in \mathcal{S}$, and by combining them through the Dice formula, that is:

$$Sim_{tag}(\{t^r\}, \{t\}) = \frac{2 \cdot \sum_{t^r,t} TagAff(t^r, t)}{|\{t^r\}| + |\{t\}|} \in [0, 1] \tag{7}$$

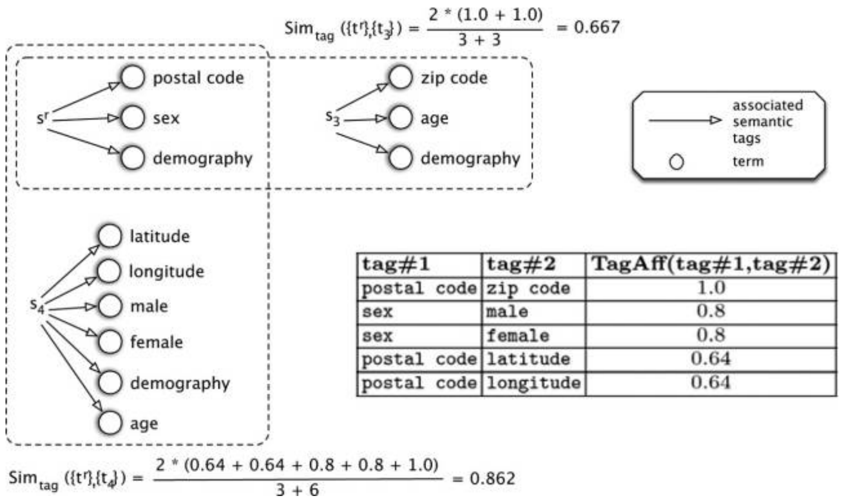
where $|\{t^r\}|$ (resp., $|\{t\}|$) denotes the number of tags in $\{t^r\}$ (resp., $\{t\}$). Pairs of tags to be considered for the $Sim_{tag}$ computation are selected according to a maximization function that relies on the assignment in bipartite graphs. The point here is how to compute $TagAff(t^r, t) \in [0,1]$. First, we distinguish the case in which $t^r$ and $t$ have been semantically disambiguated using WordNet. In this case, the tag affinity between $t^r$ and $t$ is computed as extensively described by Bianchini *et al.* (2012). Tag affinity is equal to 1.0 if the two tags belong to the same synset or coincide; it decreases as long as the path of hyponymy/hypernymy relations between the two synsets of the tags increases. In particular, tag affinity is equal to $0.8^L$, where there is a path of $L$ hyponymy/hypernymy relations between the synsets which the two tags belong to. The value of 0.8 has been proven to be optimal in our experiments on WordNet-based affinity.

In cases where either $t^r$ or $t$ has not a disambiguation based on WordNet, the $TagAff(t^r, t)$ computation is performed as follows:

- if both $t^r$ and $t$ have no disambiguation, then tag affinity $TagAff(t^r,t) = StringSim(t^r,t)$ (or vice versa), where $StringSim(\cdot) = [1 - NLevDist(\cdot)] \in [0,1]$, where $NLevDist(\cdot)$ is the Levenshtein distance normalized within the [0,1] range[3].
- if $t^r$ has not been disambiguated, while $t$ presents a sense disambiguation using WordNet, let's denote with $\mathcal{W}$ the set of synonyms of $t$, then $TagAff(t^r,t) = max_{t^i \in \mathrm{W}} \{StringSim(t^r,t^i)\}$ (for symmetry, the same case applies if $t^r$ has been disambiguated and $t$ has not).

Example. For instance, $Sim_{tag}()$ computation for data services of the running example is summarized in Figure 6, that also shows the table containing $TagAff()$ values according to the portions of WordNet reported in Figure 3.

*7.1.2 Data service aggregation matching.* To check how much an available data service $s \in \mathcal{S}$ fits the request $s^r = \langle\{t^r\}, g^r\rangle, f^r$, the similarity between $g^r$ and the aggregations in $\mathcal{G}[s]$, corresponding to the applications where $s$ has been used in the past, also plays an important role. The more aggregations in $\mathcal{G}[s]$ are similar to $g^r$, the better $s$ is considered


Sim_tag ({t^r},{t_3}) = 2 * (1.0 + 1.0) / 3 + 3 = 0.667
s^r
postal code
sex
demography
s_3
zip code
age
demography
associated semantic tags
term
s_4
latitude
longitude
male
female
demography
age
Sim_tag ({t^r},{t_4}) = 2 * (0.64 + 0.64 + 0.8 + 0.8 + 1.0) / 3 + 6 = 0.862

| tag#1 | tag#2 | TagAff(tag#1,tag#2) |
|---|---|---|
| postal code | zip code | 1.0 |
| sex | male | 0.8 |
| sex | female | 0.8 |
| postal code | latitude | 0.64 |
| postal code | longitude | 0.64 |

**Figure 6.**
Sim$_{tag}$ computation for the data services considered in the running example

as candidate search result for $s^r$. The metric $AggSim(g^r, g)$ computes the similarity between two aggregations, that is, the one that is being designed, $g^r$, and an existing aggregation $g \in \mathcal{G}[s]$. Such similarity is based on the similarity between data services included within the two compared aggregations and relies on the Dice formula, that is:

$$AggSim(g^r, g) = \frac{2 \cdot \sum_{i,j} Sim_{tag}(\{t_i\}, \{t_j\})}{|g^r| + |\mathcal{S}[g]|} \in [0, 1] \tag{8}$$

where $\mathcal{S}[g]$ is the set of services used in $g \in \mathcal{G}[s]$, $s_i \in g^r$, $s_j \in \mathcal{S}[g]$, $\{t_i\}$ is the set of tags used to annotate $s_i$, $\{t_j\}$ is the set of tags used to annotate $s_j$, $|g^r|$ (resp., $|\mathcal{S}[g]|$) is the number of data services in $g^r$ (resp., $\mathcal{S}[g]$). Pairs of data services to be considered for the $Sim_{tag}$ computation are selected according to the maximization function on which computation of equation (7) is based too. The rationale here is that, the more data services associated with the two compared aggregations are similar according to their tag similarity, the more the two aggregations are similar as well.

Example. Let us suppose that data services $s_3$ and $s_4$ of Figure 3(a) are associated with the two aggregations $g_1 = \langle\{s_1, s_2, s_3\}, d_e\rangle$ by a developer $d_e$ and $g_2 = \langle\{s_1, s_4\}, d_f\rangle$ by a developer $d_f$, respectively. Values of $Sim_{tag}(\{t^r\}, \{t_3\}) = 0.667$ and $Sim_{tag}(\{t^r\}, \{t_4\}) = 0.862$ have been computed in Figure 6. Because $g^r$ already contains $s_1$ and $s_2$, we have:

$$AggSim(g^r, g_1) = \frac{2 \times (1.0 + 1.0 + 0.667)}{3 + 3} = 0.889 \tag{9}$$

$$AggSim(g^r, g_2) = \frac{2 \times (1.0 + 0.862)}{3 + 2} = 0.745 \tag{10}$$

*7.1.3 Overall similarity.* The overall similarity $Sim(s^r,s) \in [0,1]$ between $s^r$ and each available data service $s \in \mathcal{S}$ is computed as follows:

$$\omega_s \cdot Sim_{tag}(\{t^r\},\{t\}) + (1-\omega_s)\cdot\frac{1}{|\mathcal{G}[s]|}\cdot\sum_{g\in\mathcal{G}[s]} AggSim(g^r,g) \tag{11}$$

where $|\cdot|$ denotes the set cardinality; the second term in equation (11) represents the average *AggSim*() value over all the aggregations where $s$ has been used in the past, $\omega_s \in [0,1]$ is used to balance the impact of similarity based on (semantic) tags and similarity based on data service aggregations. In Section 8, we will provide some hints about the procedure we used to choose $\omega_s$ value. If $g^r = \phi$, then $\omega_s = 1$. In the running example, $Sim(s^r,s_3) = 0.778$ and $Sim(s^r,s_4) = 0.804$. Data services included in the search results (that we denote with $\mathcal{S}' \subseteq \mathcal{S}$) are those whose overall similarity $Sim(s^r,s) \geq \tau$, where $\tau$ is set by the requester. In our experiments (see Section 8), we set $\tau = 0.4$.

*7.1.4 Technical features matching.* Similarity based on technical features is used to filter out services that have been included into $S^*$ because their overall similarity fulfills $Sim(s^r,s) \geq \tau$ but they are not compliant with $s^r$ based on technical features. In this way, we improve the precision of selection process. This similarity, denoted with $Sim_{tech}(\{f^r\},\{f\})$, is computed by evaluating total similarity between pairs of technical features, one from the set $\{f^r\}$ specified in the request and one from the set $\{t\}$ featuring the data service $s \in \mathcal{S}$, and by combining them through the Dice formula. Actually, we use here a similarity definition that deals asymmetrically with the request and the candidate service. In fact, $Sim_{tech}(\{f^r\},\{f\}) = 1$ when $\{f^r\} \subseteq \{f\}$. In other words, when the requested technical features are offered by a candidate service, the similarity is set to the maximum does not matter if the service offers more technical features. Formally:

$$Sim_{tech}(\{f^r\},\{f\}) = \frac{|\{f^r\} \cap \{f\}|}{|\{f^r\}|} \in [0,1] \tag{12}$$

where $|\{f^r\} \cap \{f\}|$ (resp., $|\{f^r\}|$) denotes the number of technical features in $|\{f^r\} \cap \{f\}|$ (resp., $\{t^r\}$). Pairs of tags to be considered for the $Sim_{tech}$ computation are selected according to a maximization function like in the discussed computation of $Sim_{tag}$. A candidate services $s \in S^*$ is filtered out if it does not fulfill $Sim_{tech}(\{f^r\},\{f\}) \geq \rho$. In our experiments, we usually set $\rho = 0.5$.

Example. In our example of request $\{f^r\} = \{$PHP, REST$\}$. The data service $s_4$ of Figure 3(a) has technical features $\{$XML, JSON, REST$\}$. Their similarity is therefore:

$$Sim_{tech}(\{f^r\},\{f\}) = \frac{\{PHP, REST\} \cap \{XML, JSON, REST\}}{|\{PHP, REST\}|} = \frac{1.0}{2} = 0.5 \tag{13}$$

## *7.2 Data service ranking*

Candidate data services $\mathcal{S}^*$ are then ranked by combining the overall similarity value with a ranking function $\rho_{serv}: \mathcal{S}^* \mapsto [0,1]$, that is based on:

- the ranking of developers who used $s \in \mathcal{S}^*$; and
- the votes $v(s, g_i, d_k)$ assigned to $s$ by each developer $d_k$ that used $s$ in an aggregation $g_i$;

In particular, the better the ranking of developers who used the service $s$ and the higher the votes assigned to $s$, the closer the value $\rho_{serv}(s)$ to 1.0 (maximum value). The value $\rho_{serv}(s)$ is therefore computed as follows:

$$\rho_{serv}(s) = \frac{\sum_{k=1}^{n} \sum_{i=1}^{m_k} dr(d_k) \cdot v(s, g_i, d_k)}{N} \in [0, 1] \quad (14)$$

where $d_k \in \mathcal{D}, \forall k$ are the developers who used the service $s$ in their own $m_k$ Web application design projects, the vote $v(s, g_i, d_k)$ is weighted by $dr(d_k)$ that is the ranking of developer $d_k$ according to equation (3). Moreover, $N$ is the number of times the service $s$ has been selected (under the hypothesis that a developer might use a data service $s$ in $m \geq 1$ projects, then $dr(d_k)$ is considered $m$ times), thus $N = \Sigma_{k=1}^{n} m_k$ The $Sim(s^r, s)$ and $\rho(s)$ elements are finally combined in the following harmonic mean:

$$rank(s) = \frac{2 \cdot \rho(s) \cdot Sim(s^r, s)}{\rho(s) + Sim(s^r, s)} \in [0, 1] \quad (15)$$

## 8. Experimental evaluation

We performed experiments to evaluate the extended WISeR according two main parts concerning: service selection and developers' ranking.

### *8.1 Experiments on service selection*

First, we tried to consider benchmarks for comparing Web source selection approaches which are described in The UIUC Web Integration Repository (2003). Nevertheless, available data sets within this repository list Web sources (e.g., Deep Web sources), but do not provide any information about past experiences in using such sources, namely, data about the Web applications designed with them and details about the social network of developers. Therefore, in the laboratory experiments we performed to test the effectiveness of our approach, we created a proper data set that is compliant with the model depicted in Figure 2. To this purpose, we considered Web APIs and data services from ProgrammableWeb. We started from this repository to:

- collect a set of data services for the *data service layer*, in the form of 1,000 methods of about 250 Web APIs in the application domain of the motivating example, where in particular selected Web APIs are classified within the eCommerce, Marketing, Advertising, CRM, Business, Enterprise and Localization categories of ProgrammableWeb;
- collect a set of Web applications designed with such methods for the *Web application layer*, in the form of about 400 mashups, and extract from them the corresponding aggregations, as sets of Web APIs in each mashup; and
- collect a set of developers for the *social layer*, in the form of mashup owners.

Within the ProgrammableWeb repository, no information is provided about social relationships between developers. Nevertheless, developers may follow/track mashups designed by other developers. Therefore, if $d_i$ follows at least a mashup developed by $d_j$, a candidate *follower-of relationship* is identified from $d_i$ to $d_j$; such relationship is then confirmed ($d_i \xrightarrow{f} d_j$) if and only if there exists at least a pair of data services used in the same application by $d_i$ that have been used together in the past also by $d_j$. Finally, we performed WordNet-based semantic tagging of data services starting from the keywords extracted from the Web APIs and mashups' descriptions. Semantic tagging has been performed with the support of the wizard described by Bianchini *et al.* (2012).

We ran the experiments on a 2.8 GHz Intel Core i7 processor, equipped with 16 GB RAM. Specifically, we randomly selected a Web application $g \in \mathcal{G}$, we chose a data service $s \in \mathcal{S}[g]$ and we issued a request $s^r = \langle \{t^r\}, g^r, \{\} \rangle$, where $\{t^r\}$ is the list of tags assigned to $s$ and $g^r = \mathcal{S}[g]/s$. We carefully manually browsed available data services, selecting and ranking each candidate service $s_i$ according to:

- the compliance of $s_i$ with $s$, based on their descriptions and assigned tags; and
- the popularity of $s_i$ in terms of its use in past Web application design projects.

We used the Kendall's tau coefficient to compare the manually performed ranking with three different configurations of our system:

(1) without considering the semantic disambiguation of tags and without considering the social layer (*prototype #1*);

(2) considering the semantic disambiguation of tags, but without considering the social layer (*prototype #2*); and

(3) considering both the semantic disambiguation of tags and the social layer (*prototype #3*).

If the agreement between two rankings is complete (i.e., the two rankings are the same), the coefficient has a value that is equal to 1. Results are shown in Figure 7, where we also tuned the $\omega_s$ weight using the following values: $\omega_s = 1$ (only $Sim_{tag}()$ is considered), $\omega_s = 0.7$ ($Sim_{tag}()$ is considered as more important than aggregation similarity), $\omega_s = 0.5$ ($Sim_{tag}()$ and aggregation similarity are considered as equally relevant) and $\omega_s = 0.3$ ($Sim_{tag}()$ is considered as less important than aggregation similarity). The experiment has been performed ten times using different requests and setting $\tau = 0.4$. Figure 7 shows the average results.

Figure 7 shows how Prototype #3 presents better performances than the others, if compared to the ideal ranking manually prepared. Specifically, semantic disambiguation of tags enables to discard irrelevant data services, thus avoiding

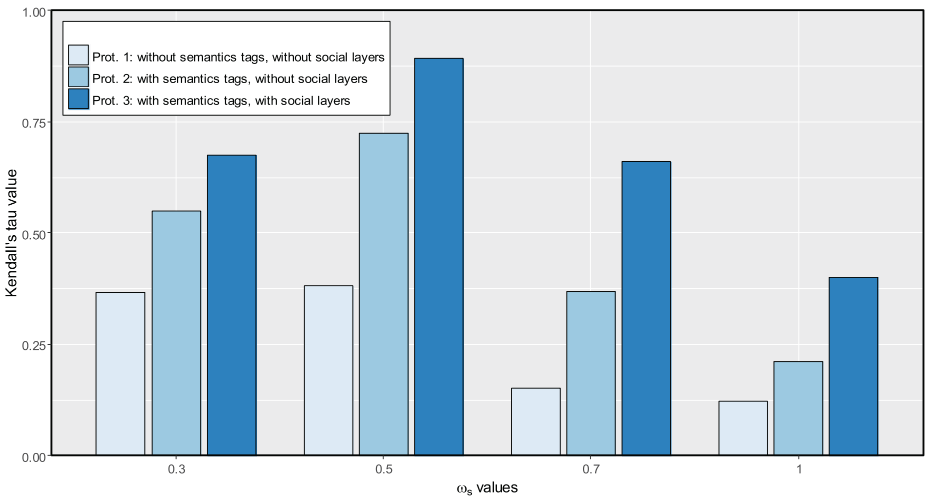


**Figure 7.**
Experimental results: Kendall's tau coefficient values for different $\omega_s$ configurations

limitations imposed by tag polysemy and synonymy. As expected, considering only $Sim_{tag}()$ values for selecting relevant data services is not effective, as the use of a specific data service for a particular Web application cannot be inferred by only inspecting tags used to classify the service; instead, aggregation similarity values and ranking based on social network of developers should also be exploited to this purpose. This has been investigated by varying the $\omega_s$ parameter. Kendall's tau values are greater for $\omega_s = 0.3$ than for $\omega_s = 0.7$: this means that in the data set considered in the experiments, the importance of past developers' experiences in using candidate data services is even greater than the information coming from data service (semantic) tagging.

### *8.2 Experiments on developers' ranking*

In this part of the experimentation, we focus on determining the effectiveness of our developers' ranking procedure and metrics. To this purpose, we perform some experiments based on 14 developers organized according to the social network depicted in Figure 4, whose features are summarized in Table I. This data set has been built with the aim of differentiating the features of the involved developers, namely:

- developers with high credibility ($0.8 \leq c(d_i) \leq 1.0$), who designed many aggregations ($|\mathcal{G}(d_i)| \geq 3$), that is, dev1, and who designed few aggregations ($|\mathcal{G}(d_i)| < 3$), that is, {dev3, dev7, dev12};
- developers who present medium credibility ($0.4 \leq c(d_i) < 0.8$) and designed many aggregations ($|\mathcal{G}(d_i)| \geq 3$), that is, {dev2, dev5, dev10, dev11}, and who designed few aggregations ($|\mathcal{G}(d_i)| < 3$), that is, {dev9, dev13, dev14}; and
- developers with low credibility ($0 \leq c(d_i) < 0.4$), who designed many aggregations ($|\mathcal{G}(d_i)| \geq 3$), that is, dev4, and who designed few aggregations ($|\mathcal{G}(d_i)| < 3$), that is {dev6, dev8}.

In the following, we focus on ranking of developers and we compare manual rankings with results of automated ranking as produced by the extended WISeR framework.

For each developer in the data set, the ideal ranking of the other developers has been considered, based on their features (Figure 4 and Table I). In particular, to manually determine the ideal ranking of developers from the viewpoint of a requester $d^r$, we performed pairwise comparisons of developers $\langle d_i, d_j \rangle$ with respect to the following criteria:

- relevant differences in their credibility ($c(d_i) \gg c(d_j)$ or vice versa, see for example, dev1 and dev4);
- the existence of a direct *follower-of* relationship between $d^r$ and $d_i$ and/or $d_j$; and
- relevant difference in the number of designed applications (see, for example, $|\mathcal{G}(\text{dev1})| = 5$ and $|\mathcal{G}(\text{dev13})| = 1$).

If an overall preference between $d_i$ and $d_j$ cannot be identified considering these criteria (see, for example, dev2 and dev11), the credibility of their followers and the presence and length of a path of *follower-of* relationships between $d^r$ and $d_i$ and/or $d_j$ are further considered. For instance, considering dev3 as the requester, developers in the data set should be ideally ranked as: dev1 – dev11 – dev2 – dev7 – dev9 – dev12 – dev14 – dev5 – dev4 – dev8 – dev10 – dev13 – dev6.

Then, we run our system, using different values for the damping factor $\alpha \in [0,1]$ and we compared the obtained ranking against the ideal one using the *Kendall tau distance*

$k \in [0,1]$, where $k = 0$ means that the compared rankings fully agree (Kendall and Gibbons, 1990). We repeated the experiments for all the 14 developers in the data set and we calculated the average value of the Kendall tau distance. Figure 8 shows average values of the Kendall tau distance with respect to the values of the damping factor $\alpha \in [0,1]$.

Finally, we used the ideal ranking and the Kendall tau distance to compare:

- our approach, setting $\alpha = 0.6$;
- an optimistic situation, where maximum credibility is assigned to all developers (i.e., $c(d_i) = 1.0, \forall i$), setting $\alpha = 0.6$; and
- a situation biased on the requester, where only the relative ranking $\rho_{rel}^{d^r}(d_i)$ has been considered.

We repeated the experiments for all the 14 developers in the data set and we calculated the average value of the Kendall tau distance. The resulting values of the average Kendall tau distance are given in Figure 9, showing the accuracy of our ranking solution.

## 9. Related work

In literature, data service (or data source) selection and their integration or composition to enable querying over them, are as we already remarked in this paper clearly distinguished into two distinct phases (Ceri *et al.*, 2010). According to this viewpoint, various approaches consider as separate the activities of selection and integration when applied to Deep Web data querying (Calì and Martinenghi, 2010), multi-domain queries issued over multiple data services (Campi *et al.*, 2010; Quarteroni *et al.*, 2013), as well as Web application development driven by design patterns (Brambilla and Tziviskou, 2014). For example, the approach described by Quarteroni *et al.* (2013), which shares with our many preliminary definitions, focuses on data service integration and querying, considering the data service selection results as given. In our paper, we have dealt with data service selection issues, which have to be solved before beginning integration of data and metadata coming from the selected sources. Concerning selection on data service or Web data source, recent trends focused their attention on search over Deep Web data sources (Li *et al.*, 2013), search computing (Bozzon *et al.*, 2013; Cafarella *et al.*, 2011), quality-driven ranking (Barbagallo *et al.*, 2012; Dillon *et al.*, 2013; Xian *et al.*, 2009), collaborative filtering (Cao *et al.*, 2014), users' experience (Liu *et al.*, 2015) and social-based (Maaradji *et al.*, 2011) selection.

### *9.1 Deep Web search*

Querying Deep Web contents can be hampered by access constraints that are superimposed by Web forms used to query data from hidden databases (Calì and Martinenghi, 2010). Li *et al.* (2013) propose a divide-and-conquer strategy, that properly combines a page classifier, a link scoring algorithm and novel crawling policies to discover and recognize entry points of domain-specific Deep Web databases, that is, Web forms.Malki *et al.* (2016) propose an approach to describe data services with semantic-based uncertain descriptions. On top of these descriptions, a method is defined to evaluate user queries over data services that show correlated semantics (i.e., when data services share the same data sources).

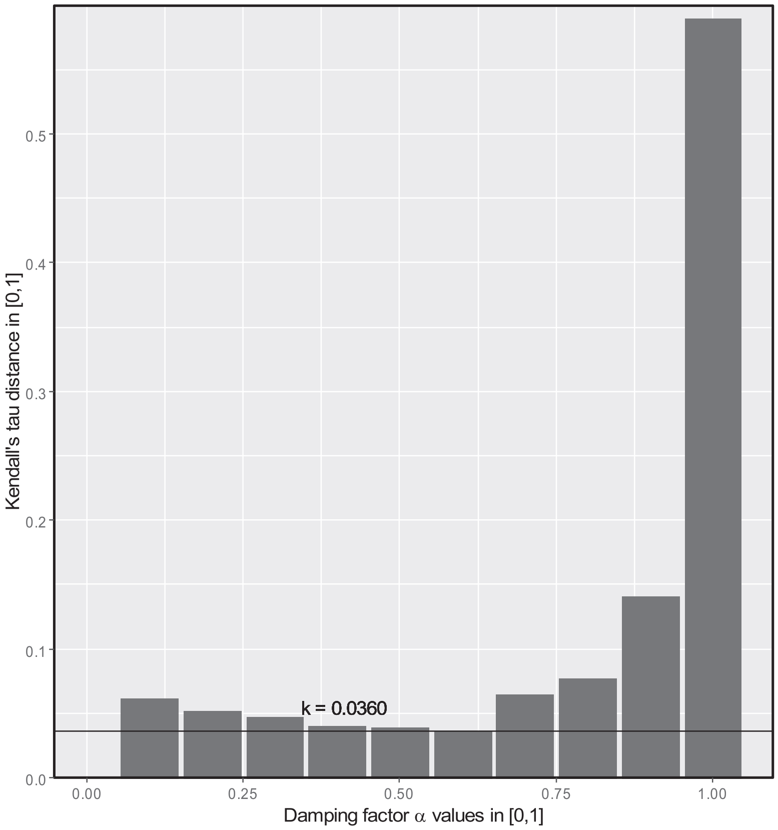


**Figure 8.**
Average Kendal tau distances computed on developers' ranking with respect to the damping factor $\alpha$

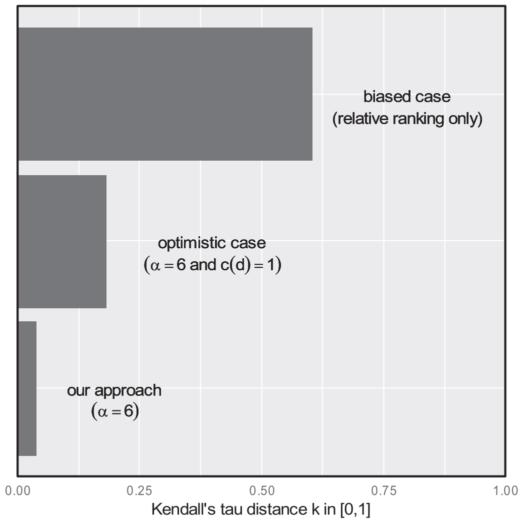


**Figure 9.**
Average Kendal tau distances computed for the approach validation

### *9.2 Search computing*

The Search Computing Project Campi *et al.* (2010) focuses on search services, properly identified and registered within a search service integration platform; search services are combined to define vertical applications through conceptual modeling of the domain of interest. The Search Computing approach aims at formulating *exploratory queries*, that is, queries that enable the users to follow the links across related concepts and to iteratively refine the query on the basis of previous search results (Bozzon *et al.*, 2013). Cafarella *et al.* (2011) apply the concept to all structured data over the Web using Google's Web Tables and a Deep Web Crawler.

### *9.3 Quality-based ranking*

Source ranking, based on quality criteria, is addressed by Xian *et al.* (2009) and Barbagallo *et al.* (2012). Xian *et al.* (2009) propose a data source selection approach based on the quality of Deep Web sources, assessed by evaluating quality dimensions that represent the features of the sources. Barbagallo *et al.* (2012) present a platform for data mashup, supporting quality-driven filtering and composition of Web 2.0 sources. The approach used by Dillon *et al.* (2013) considers as very important the quality of data sources as well. To ensure the selection of Web sources, authors present a framework that is based on a subject-specific database, curated by domain experts and dynamically generated through a large user input. The intervention of domain experts could become the bottleneck of the overall system. Functional requirements and quality requirements can be combined for service discovery and search, as we discussed by Bianchini *et al.* (2004).

### *9.4 Collaborative filtering*

Other quite recent approaches are based on techniques of collaborative filtering applied to Web APIs recommendation. For example, in the CSCF (Content Similarity and Collaborative Filtering) Web API recommender system, described by Cao *et al.* (2014), users' ratings have also been considered to refine service ranking by applying collaborative filtering techniques. In Li *et al.*'s (2014) study, tags used to annotate both RESTful data services and mashups are classified into topics through a probabilistic distribution. Topics are used to add semantics on top of traditional tagging. The API popularity is taken into account as experience dimension, computed as the number of times a Web API has been used in the past. API popularity is used during search to weigh API relevance (with respect to the issues request), based on topics. The work described by Tapia *et al.* (2011) is similar, because it combines the API popularity and API co-occurrence in mashups with traditional keyword search techniques. With respect to these approaches, we exploit the network of social relationships between developers, integrating the credibility assessment techniques and the concept of aggregation-contextual rating and adapting them to the enterprise context, considering the requirements presented in the introduction.

### *9.5 Users' experience for service selection*

Recent trends on (data) service selection integrated users' experience aspects (ratings/votes, co-usage of the same services in several applications, service popularity, etc.) with traditional recommendation techniques, mainly based on functional and non-functional features. For example, matrix factorization techniques have been used in dynamic service exploration and recommendation, to identify user-/topic-/service-related latent factors that influence users to make service selections (Liu and Fulia, 2015). For service recommendation, emerging techniques based on user behavior similarity and service usage can be interesting too (Liu *et al.*, 2015). Al-Sharawneh *et al.*'s (2011) and Malik and Bouguettaya's (2009) approaches study service selection based on votes/ratings and introduced techniques to evaluate the reputation and credibility of raters. They are mainly focused on these aspects, compared to our approach, where credibility assessment is integrated in a complex framework for data service selection. This enables a finer and more effective selection of data services. The system described by Cao *et al.* (2013) first extracts users' interest as a set of tags weighted based on description documents of mashups the user has used in the past. Second, users' interests are used to recommend RESTful services and mashups described by tags similar to the user's interest.

### *9.6 Social-based service selection*

An attempt to introduce social network exploitation for Web API/service selection is described by Maaradji *et al.* (2011). In the SoCo (Social Composer) system (Maaradji *et al.*, 2011), based on collaborative filtering, components are suggested to the user *u* considering other users that are similar to *u* in a social network. Social relationships may be explicit (*u* can explicitly declare to share the same interests, in terms of components, of other users) or implicit (that is, inferred according to the activities of users, e.g., when a person uses many of the components created by other persons). Specifically, a component is suggested to *u* depending on the number of times the component has been used by other users socially related to *u* and on the social proximity between such users and *u*. The notion of *social network* in the previously cited paper by Cao *et al.* (2013) presents a different meaning compared to our definition. Indeed, it is not a social network of developers, social relationships are defined between mashups, if their tag-based similarity and the number of common services are over a pre-defined threshold.

Nevertheless, these approaches do not take into account the topology of the developers' social network to support service selection. Moreover, we focused our attention on developers' social networks in an enterprise context, integrating the developers' credibility assessment and implementing the social network-based data service selection on top of a more complex framework based on a multi-perspective model.

## 10. Conclusions

In this paper, we presented a data service selection framework that extends and specializes existing ones for Web APIs selection. We discussed the revised multi-layered model for data services and introduced proper metrics relying on it, meant for supporting the selection of data services in a context of Web application design. Model and metrics take into account the network of social relationships between developers, to exploit them for estimating the importance that a developer assigns to past experience of other developers. In summary, the framework is apt to provide advanced search and ranking techniques by considering:

- lightweight data service descriptions, in terms of (semantic) tags and technical aspects;
- previously developed aggregations of data services, to use in the selection process of a service the past experiences with the services when used in similar applications;
- social relationships between developers' developers (social network); and
- developers' credibility evaluations.

We also discussed some experimental results that we are planning to expand with other experiments to check how developers feel themselves using the approach. An interesting issue to be investigated concerns the time sensitivity of social relationships, that is, relationships that have been established later could have a different impact for data service ranking.

Finally, further studies are ongoing for extending the social network model: specifically, other aspects such as the maturity of the use of data services (estimated through their publishing data and the number and quality of aggregations including the services) and specificity of the searched services (i.e., general purpose or domain-specific) may be investigated with respect to a possible influence in the search and ranking process.

## Notes

1. http://geodataservice.net
2. http://enhancing-peer-review.nih.gov/scoring%26reviewchanges.html
3. Note that the Levenshtein distance does not belong to the [0,1] range, as it is the number of single-character edits to change one string into the other; however, by definition, it is at most equal to the length of the longer string; therefore, this upper bound is used to normalize the measure within [0,1] range.

## References

Al-Sharawneh, J., Williams, M., Wang, X. and Goldbaum, D. (2011), "Mitigating risk in web-based social network service selection: follow the leader", *Proceedings of Sixth International Conference on Internet and Web Applications and Services*, *St. Maarten*, pp. 156-164.

Balakrishnan, R., Kambhampati, S. and Manishkumar, J. (2013), "Assessing relevance and trust of the deep web sources and results based on inter-source agreement", *ACM Transactions on the Web*, Vol. 7 No. 2, p. 32.

Barbagallo, D., Cappiello, C., Francalanci, C., Matera, M. and Picozzi, M. (2012), "Informing Observers: Quality-driven Filtering and Composition of Web 2.0 Sources", *Proceedings of International Workshop on Business Intelligence and the WEB*, *Berlin*.

Bianchini, D., De Antonellis, V. and Melchiori, M. (2004), "QoS in ontology-based service classification and discovery", *IEEE Proceedings of International Workshops on Database and Expert Systems Applications - DEXA*, *Zaragoza, Spain*, pp. 145-150.

Bianchini, D., De Antonellis, V. and Melchiori, M. (2012), "Semantic collaborative tagging for web APIs sharing and reuse", *Proceedings of the 12th International Conference on Web Engineering (ICWE)*, *Berlin*, pp. 76-90.

Bianchini, D., De Antonellis, V. and Melchiori, M. (2013), "A multi-perspective Framework for Web API search in enterprise mashup design (best paper)", *Proceedings of 25th International Conference on Advanced Information Systems Engineering (CAiSE)*, *Valencia*, pp. 353-368.

Blasch, E., Chen, Y., Chen, G., Shen, D. and Kohler, R. (2013), "Information fusion in a cloud-enabled environment", *High Performance Cloud Auditing and Applications*, Springer-Verlag, NY.

Bozzon, A., Brambilla, M., Ceri, S. and Mazza, D. (2013), "Exploratory search framework for web data sources", *VLDB Journal*, Vol. 22 No. 5, pp. 641-663.

Brambilla, M. and Tziviskou, C. (2014), "Model-driven design frameworks for semantic web applications", *Semantic Web Enabled Software Engineering*, IOS Press, Amsterdam, pp. 179-204.

Cafarella, M.J., Halevy, A. and Madhavan, J. (2011), "Structured data on the web", *Communications of the ACM*, Vol. 54 No. 2, pp. 75-79.

Calì, A. and Martinenghi, D. (2010), "Querying the deep web", *Proceedings of 13th International Conference on Extending Database Technology (EDBT2010)*, *Lausanne*, pp. 724-727.

Campi, S., Ceri, S., Gottlob, G., Maesani, A. and Ronchi, S. (2010), "Chapter 9: service marts", *Search Computing*, Springer, Heidelberg, pp. 163-187.

Cao, B., Liu, J., Tang, M., Zheng, Z. and Wang, G. (2013), "Mashup service recommendation based on user interest and social network", *Proceedings of International Conference on Web Services (ICWS)*, *Santa Clara, CA*.

Cao, B., Tang, M. and Huang, X. (2014), "Cscf: A mashup service recommendation approach based on content similarity and collaborative filtering", *International Journal of Grid and Distributed Computing*, Vol. 7 No. 2, pp. 163-172.

Ceri, S., Braga, D., Corcoglioniti, F., Grossniklaus, M. and Vadacca, S. (2010), "Search computing challenges and directions", *Objects and Databases, Lecture Notes in Computer Sciences*, Vol. 6348, Springer-Verlag, Berlin, Heidelberg, pp. 1-5.

De Antonellis, V., Melchiori, M. and Plebani, P. (2003), "An approach to web service compatibility in cooperative processes", *Proceedings of 2003 IEEE Symposium on Applications and the Internet Workshops, SAINT 2003*, *Orlando, FL*, pp 95-100.

Dillon, S., Stahl, F. and Vossen, G. (2013), "Towards the web in your pocket: curated data as a service", *Advanced Methods for Computing Collective Intelligence*, Springer-Verlag, Berlin, pp. 25-34.

dos Santos, T.A, de Araujo, R.M. and Magdaleno, A.M. (2010), "Identifying collaboration patterns in software development social networks", *Journal of Computer Science*, Vol. 9 No. 1, pp. 51-60.

Fuxman, A., Giorgini, P., Kolp, M. and Mylopoulos, J. (2001), "Information systems as social structures", *Formal Ontology in Information Systems*, ACM, New York, pp. 12-21.

Gupta, V. and Lehal, G.S. (2009), "A survey of text mining techniques and applications", *Journal of Emerging Technologies in Web Intelligence*, Vol. 1 No. 1, pp. 60-76.

Hertzum, M. (2014), "Expertise seeking: a review", *Information Processing and Management*, Vol. 50 No. 5, pp. 775-795.

Kendall, M. and Gibbons, J.D. (1990), *Rank Correlation Methods*, Edward Arnold, London, UK.

Li, C., Zhang, R., Huai, J. and Sun, H. (2014), "A novel approach for API recommendation in mashup development", *Proceedings of International Conference on Web Services (ICWS)*, *Anchorage, AK*, pp. 289-296.

Li, Y., Wang, Y. and Du, J. (2013), "E-FFC: an enhanced form-focused crawler for domain-specific deep web databases", *Journal of Intelligent Information Systems*, Vol. 40 No. 1, pp. 159-184.

Liu, R., Xu, X. and Wang, Z. (2015), "Service recommendation using customer similarity and service usage pattern", *Proceedings of IEEE International Conference on Web Services (ICWS 2015)*, *New York*, pp. 408-415.

Liu, X. and Fulia, I. (2015), "Incorporating user, topic, and service related latent factors into web service recommendation", *Proceedings of IEEE International Conference on Web Services (ICWS 2015)*, *New York*, pp. 185-192.

Maaradji, A., Hacid, H., Skraba, R., Lateef, A., Daigremont, J. and Crespi, N. (2011), "Social-based web services discovery and composition for step-by-step mashup completion", *Proceedings of International Conference on Web Services (ICWS 2011)*, *Washington Marriott, DC*.

Malik, Z. and Bouguettaya, A. (2009), "RATEWeb: reputation assessment for trust establishment among web services", *VLBD Journal*, Vol. 18 No. 4, pp. 885-911.

Malki, A., Benslimane, D., Benslimane, S.M., Barhamgi, M., Malki, M., Ghodous, P. and Drira, K. (2016), "Data services with uncertain and correlated semantics", *World Wide Web*, Vol. 19 No. 1, pp. 157-175.

Quarteroni, S., Brambilla, M. and Ceri, S. (2013), "A bottom-up, knowledge-aware approach to integrating and querying web data services", *ACM Transactions on the Web*, Vol. 7 No. 4, 33 pages.

Schafermeier, R. and Paschke, A. (2011), "Using domain ontologies for finding experts in corporate wikis", *I-SEMANTICS*, *Graz, Austria*, pp. 63-70.

Tapia, B., Torres, R. and Astudillo, H. (2011), "Simplifying mashup component selection with a combined similarity- and social-based technique", *Proceedings of the 5th International Workshop on Web APIs and Service Mashups*, *Lugano, Switzerland*, pp. 1-8.

The UIUC Web Integration Repository (2003), Computer Science Department, University of Illinois at Urbana-Champaign, available at: http://metaquerier.cs.uiuc.edu/repository

Vitvar, T. and Musser, J. (2010), "ProgrammableWeb.com: statistics, trends, and best practices", *Keynote of the Web APIs and Service Mashups Workshop at the European Conference on Web Services (ECOWS)*, *Ayia Napa, Cyprus*.

Xian, X., Zhao, P., Fang, W., Xin, J. and Cui, Z. (2009), "Quality-based data source selection for web-scale deep web data integration", *Proceedings of the Eighth International Conference on Machine Learning and Cybernetics*, *Baoding, Hebei*, pp. 427-432.

**Corresponding author**
Michele Melchiori can be contacted at: michele.melchiori@unibs.it