Chapter 14
A Unified Conceptual Framework for Managing Services in the Web Oriented Architecture

Devis Bianchini, Valeria De Antonellis and Michele Melchiori

Abstract In recent years, there has been an increasing adoption of the agile paradigm for developing data-intensive web applications, relying on the selection and reuse of third party components. In parallel, the Web Oriented Architecture (WOA) has emerged, gathering together the notions underneath Service-Oriented Architecture (SOA), REpresentation State Transfer (REST) and web applications. In particular, WOA has promoted the success of: a) RESTful services for access to web data sources, and b) public repositories where these data providing services, in the form of Web APIs, are made available to the community of developers. In this context, it is more and more relevant to support the developers, even operating in community networks, to select from available repositories suitable APIs for their development needs. Nevertheless, recent selection approaches considered different features, complementary and only partially overlapping, among the ones used for service descriptions in the repositories. In this chapter a conceptual framework is defined that considers all the features to enable a flexible selection of data providing services over multiple repositories. To this aim, the framework provides: (i) a multi-perspective model for service description, that also includes a social-based perspective, focused on the community of developers, their mutual relationships and their estimated credibility in web application development; (ii) a collection of search and ranking techniques that rely on the model; (iii) a prototype system that implements the unified conceptual framework on top of service repositories.

Devis Bianchini
Dept. of Information Engineering University of Brescia, Via Branze, 38 - 25123 Brescia (Italy)
e-mail: bianchin@ing.unibs.it

Valeria De Antonellis
Dept. of Information Engineering University of Brescia, Via Branze, 38 - 25123 Brescia (Italy)
e-mail: deantone@ing.unibs.it

Michele Melchiori
Dept. of Information Engineering University of Brescia, Via Branze, 38 - 25123 Brescia (Italy)
e-mail: melchior@ing.unibs.it

J. Cabot et al. (eds.), *Conceptual Modeling Perspectives*,
https://doi.org/10.1007/978-3-319-67271-7_14

Key words: Web Oriented Architecture; RESTful service; developers' social network; collective knowledge; selection; search; ranking; similarity.

14.1 Introduction

In recent years, there has been an increasing adoption of the agile paradigm for developing data-intensive web applications, relying on the selection and reuse of third party components. In parallel, the increasing diffusion of the Web Oriented Architecture (WOA) paradigm has progressively shifted the technologies for web application development, gathering together the notions of Service-Oriented Architecture (SOA), REpresentation State Transfer (REST) and web applications. In particular, WOA has promoted the success of: a) RESTful services for access to web data sources; b) public repositories where these data providing services, in the form of Web APIs, are made available to the community of developers [1]. As a consequence, nowadays, it is more and more relevant to support the developers, even operating in community networks, to select from available repositories suitable data providing services for their needs. Service search and ranking techniques generally exploit different features in service descriptions. Beyond categories, tags and technical features, the following aspects are generally considered: (i) the co-occurrence of APIs in the same applications [2, 3]; (ii) the network traffic, e.g., number of visitors around APIs and applications (also denoted as mashups) [4, 5]; (iii) the ratings assigned by developers [6, 7]. Moreover, social relationships between developers, developers' experience and their credibility are considered relevant features, as already highlighted for traditional database systems [8]. Generally, in the approaches, subsets of features among the ones present in available repositories, such as `ProgrammableWeb` or `Mashape`, are considered. As of May 2017, `ProgrammableWeb` contains over 17,000 Web APIs, that have been used in more than 6,300 mashups (excluding the deprecated ones), while over 100,000 developers are registered in the repository. Web APIs are described through categories, tags and technical features, and the list of mashups that have been developed with the APIs. `Mashape`[1] is a cloud API hub, where each Web API is associated with the list of developers who adopted or declared their interest for it (denoted as *consumers* and *followers*, respectively) and where a developer can follow other developers (leveraging a twitter-like organization). Other public repositories, such as `apigee` or `Anypoint API Portal`[2], focus on a subset of these features.

As it has been proven that conceptual modeling plays a crucial role since the early stages of agile applications development [9, 10], the aim here is to demonstrate its effectiveness in enabling flexible data providing service selection over multiple repositories, by the definition of a unified model apt to consider all relevant features. To this purpose, a conceptual framework is defined to provide a reference model,

[1] `https://www.mashape.com/`

[2] `https://api-portal.anypoint.mulesoft.com`

capturing different service modeling perspectives, and a collection of techniques and methods for service selection in web application development. The conceptual framework is the basis of WISeR (Web apI Search and Ranking) a prototype system that has been developed to implement the service search and ranking facilities. Partial results of our work have been presented in [7, 11, 12], here the final overall framework is presented.

The chapter is organized as follows: in Section 14.2 existing approaches in literature are presented and motivations for a unified conceptual framework are discussed; Section 14.3 describes the multi-perspective conceptual model; Section 14.4 details service search and ranking techniques, that take advantage of the unified model; in Section 14.5 the WISeR system is shortly described; finally, Section 14.6 closes the chapter.

14.2 Related Work

Several approaches in literature based Web API search and ranking strategies on lightweight descriptions. These approaches are referred to as *selection-oriented approaches*. They are conceived to select candidate Web APIs to feed *composition-oriented approaches*, mainly focused on providing support for properly combining available components [13]. Among selection-oriented approaches, there have been research efforts on service selection for mashup development based on API co-occurrence [14, 15], quality of components [16] and collaborative filtering [17].

Table 14.1 State of the art on Web API selection-oriented approaches.

	[18]	[3]	[6]	[4]	[5]	[2]	WISeR
Categories	✗	✗	✓	✗	✗	✗	✓
Tags/keywords	✓	✓	✓	✓	✓	✓	✓
Semantic tagging	✗	✗	✗	✗	✓	✗	✓
Mashup/API tagging	✓	✓	✓	✗	✓	✓	✓
Technical features	✗	✗	✓	✗	✗	✗	✓
Web API co-occurrence	✓	✓	✗	✓	✗	✓	✓
Web API rating	✗	✗	✓	✗	✗	✗	✓
Mashup-contextual rating	✗	✗	✗	✗	✗	✗	✓
Developers' experience	✗	✗	✗	✓	✗	✗	✓
Number of Web API uses	✗	✗	✓	✓	✓	✓	✓
Different search scenarios	✗	✗	✗	✗	✗	✓	✓

The focus here is on approaches that study the effects of taking into account multiple features for Web API selection. For these approaches, a summary of differences against the work described in this chapter is provided in Table 14.1, where all the considered features are reported.

In particular, the approaches in [3, 18] combine descriptive features based on tags with Web API popularity (number of mashups where APIs have been used and users' ratings). The system described in [18] firstly models user's interests as vectors of weighted tags, where tags are extracted by textual descriptions of the mashups the user has interacted with in the past. Similarly, vectors of weighted tags are extracted by textual descriptions of mashups and are used to represent them. Secondly, users' interests are used to recommend mashups based on a composite metrics considering: (i) similarity of vectors describing the user's interests and candidate mashups, (ii) similarity of both APIs and tags contained in the user' request for a mashup and in the candidate mashups. The approach has been extended into the CSCF (Content Similarity and Collaborative Filtering) Web API recommender system [3], where users' ratings have been also considered to refine API ranking. Other selection-oriented approaches include features related to social relationships among developers to discover and propose the best ranked Web APIs to mashup developers [4, 5]. In the SoCo (Social Composer) system [4], based on collaborative filtering, APIs are suggested to the user u considering other users who are similar to u in a social network. Social relationships may be: (a) explicit, that is, u can explicitly declare to share the same interests, in terms of APIs, of other users; (b) implicit, that is, inferred according to the activities of users, e.g., when an user adopts many of the APIs created by other users. A Web API is suggested to u depending on the number of times the API has been used by other users socially related to u and on the social proximity between users. In [5] tags used to annotate both APIs and mashups are classified into topics through a probabilistic distribution. Topics are used to add semantics on top of traditional tagging. In [2] authors distinguish between keywords assigned to mashups and keywords assigned to APIs, and the search takes into account this distinction. Moreover, number of mashups that include a Web API has been used to provide a Web API ranking. The Serviut Rank proposed in [6] has been combined with traditional tag-based or keyword-based search. The rank has been defined taking into account the number of times an API has been used in mashups, but also the popularity of mashups themselves, in terms of users' ratings and Internet traffic.

All the analysed approaches highlight useful features to perform service selection, although different approaches focus on complementary features, as shown in Table 14.1. To improve selection effectiveness and flexibility, we propose here a conceptual framework including a multi-perspective model that relies on all features present in available repositories.

14.3 Multi-Perspective Conceptual Model

14.3.1 Motivations

Different features, based on information available within service repositories, might help developers to select third party components for developing data-intensive applications: (i) the number of service followers and the number of mashups, where services have been used in, might help to identify widespread solutions, used by many developers to design their own applications; (ii) votes/ratings by developers might help to identify services shared by trustworthy providers; moreover, votes assigned to services while used in specific kinds of applications would be properly used to suggest the same service for developing similar applications; (iii) largely used and highly rated data providing services might have at their disposal valuable datasets, as well as functionalities tested by millions of users, so their re-use might offer advantages compared to their development from scratch, saving development costs and testing efforts. The combination of different features might have positive effects on service selection. In fact, service search and ranking focused on a single perspective may bring to misleading results. For example, as underlined in [2], service selection techniques that are based on descriptive features only heavily rely on the quality of information specified by service providers, which in public repositories cannot be always ensured. On the other hand, just considering number of service usages or developers' ratings suffers from the *cold start problem* and *preferential attachment* ("rich gets richer"); this means that the more used is a service, the more likely it will be selected as part of a new application, despite its compliance with requirements, while it is very difficult for new services to enter the market.

These considerations motivate the need of a comprehensive conceptual model that merges together multiple perspectives on service descriptions, in terms of different features.

14.3.2 Representation of data providing services

The unified conceptual model here proposed to describe data providing services brings together multiple features and is divided into three parts for Service Description, Service Annotation and Service Experience, as shown in Figure 14.1.

14.3.2.1 Service Description

Services are represented at two levels of abstraction:

- a *component perspective*, focused on categories, technical features and tags in service descriptions;
- an *application perspective*, focused on service aggregations in mashups.

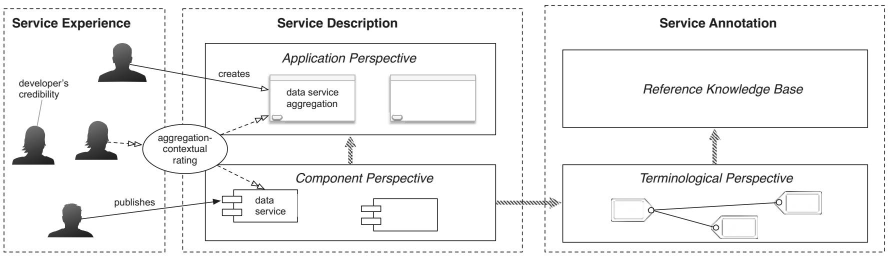

Fig. 14.1 Overview of multi-perspective conceptual model for data providing services.

Definition 14.1. A service s is an operation/method/query to access data of a web source, whose underlying data schema might be unknown to those who use the service. $\mathscr{S}$ denotes the overall set of available services. A service $s \in \mathscr{S}$ is modeled as $\langle n_s, descr_s, URI_s, \mathscr{F}_s, \mathscr{T}_s \rangle$, where:

- n_s is the service name;
- $descr_s$ is a human-readable, textual description of the service;
- URI_s is the unique resource identifier for the service;
- $\mathscr{F}_s$ is an array of elements, where each element $\mathscr{F}_s^X$ represents a technical feature X (e.g., protocols, data formats, authentication mechanisms, to mention features used in `ProgrammableWeb.com`); each technical feature is modeled as a set of allowed values for that feature (e.g., XML or JSON as data formats);
- $\mathscr{T}_s$ is a set of terms used to provide a terminological description of the service (terminological equipment).

The set of terms $\mathscr{T}_s$ is defined for tagging purposes as explained in the following *Service Annotation* description. In Figure 14.2, examples of services taken from `ProgrammableWeb.com` are listed, where URIs and textual descriptions have been omitted.

Application Perspective. Concerning modern application development, to implement a web application starting from available services, developer has to search the set of available services, select the most suitable ones, integrate and compose them, in order to deploy the final application. Within the scope of this chapter, the focus is on the first step, i.e., service selection. Service aggregations are mentioned, instead of web applications, that are the final product of the development process. An aggregation is defined as follows.

Service	Service name	Technical features	Tags
s_1	HotWire	$\mathscr{F}_{s_1}^{DataFormat} = \{\texttt{XML,JSON}\}$ $\mathscr{F}_{s_1}^{Protocol} = \{\texttt{RSS, Atom, REST}\}$	{`City`, `Star`, `Hotel`, `Travel`}
s_2	EasyToBook	$\mathscr{F}_{s_2}^{DataFormat} = \{\texttt{XML}\}$ $\mathscr{F}_{s_2}^{Protocol} = \{\texttt{SOAP}\}$	{`City`, `Hotel`, `Travel`}
s_3	MyAgentDeals	$\mathscr{F}_{s_4}^{DataFormat} = \{\texttt{XML,JSON}\}$ $\mathscr{F}_{s_4}^{Protocol} = \{\texttt{HTTP}\}$	{`City`, `Star`, `Near`, `Hotel`, `Travel`}

Fig. 14.2 Examples of service descriptions.

Definition 14.2. An aggregation represents a set of services that will be mashed-up to deploy a web application. We denote with $\mathscr{G}$ the overall set of aggregations. An aggregation g is modeled as $\langle n_g, descr_g, URI_g, \mathscr{S}_g, d_g \rangle$, where:

- n_g is the aggregation name;
- $descr_g$ is a human-readable, textual description of the aggregation;
- URI_g is the unique resource identifier for the aggregation;
- $\mathscr{S}_g = \{s_g^1, \dots s_g^n | s_g^i \in \mathscr{S}\}$ is the set of services aggregated in g;
- d_g is the developer of the aggregation.

Fictious examples of aggregations are listed in the following, where URIs and textual descriptions have been omitted.

$g_1 \Rightarrow \langle \texttt{TravelPlan}, \mathscr{S}_{g_1} = \{s_1, s_3\}, d_{g_1} \rangle$
$g_2 \Rightarrow \langle \texttt{Stay\&Fun}, \mathscr{S}_{g_2} = \{s_2, s_3\}, d_{g_2} \rangle$

14.3.2.2 Service Annotation

Services are associated with a terminological equipment, composed of terms, that are used for tagging purposes in order to improve search and ranking.

For semantic characterization, a term can be related to an ontological concept or to a WordNet term and a set of other terms (denoted as *bag of words*) can be associated with it. In particular, given a term t^i: (i) if t^i is related to a term in WordNet, its bag of words coincides with the list of synonyms of the term; (ii) if t^i is related to a concept in an ontology, its bag of words is composed of the names of other concepts related to t^i by semantic relationships in the ontology (to this aim, in the current version of the approach presented here, OWL/RDF equivalence relationship is considered); (iii) finally, if t^i is an unrelated term, its bag of words is empty. Starting from the tag specified by the developer, who is performing tagging, a proper wizard is used to support the developer for selecting the intended meaning. The tagging procedure has been extensively described in [7] by using WordNet. When based on ontologies, it is performed in a similar way. The WISeR system is compliant with WordNet and any OWL ontology a developer might choose for semantic disambiguation of terms. The approach here discussed is neutral with respect to the adopted ontologies.

14.3.2.3 Service Experience

The focus is on the set $\mathscr{D}_s$ of developers, who used the service s to develop their own mashups. In particular, a developer $d_i \in \mathscr{D}_s$ can express votes represented by $v(s_j, g_k, d_i) = \mu_{jk} \in [0, 1]$ to denote that d_i assigned a quantitative rating μ_{jk} to the service s_j when used within the aggregation g_k (*aggregation-contextual rating*). Votes are assigned according to the NIH 9-point Scoring System[3]. This scoring

[3] `http://enhancing-peer-review.nih.gov/scoring%26reviewchanges.html.`

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). These options are uniformly distributed over the $[0,1]$ interval so that the highest vote to a service corresponds to 1 and the lowest to 0. The possibility of assigning votes in the context of a specific aggregation is a distinguishing feature of the approach compared to the most popular repositories (and, among them, `ProgrammableWeb`), where votes are assigned to Web APIs regardless the mashups where they have been used. This distinction relies on the fact that a service could be suitable to be used only in specific aggregations.

A social-based perspective focused on the community of developers is also part of the model. In particular, as detailed in the following Section 14.4.3, the service selection phase takes advantage of the aforementioned votes and weights a vote proportionally to the rank of developer who expressed the vote. This rank summarizes the importance of the developer in the social network: high rank indicates high importance in the network, as discussed in the following Section 14.4.2.

Definition 14.3. A social network of developers is a pair $SN = \langle \mathscr{D}, \mathscr{E} \rangle$, where: (a) $\mathscr{D}$ is the set of developers; (b) $\mathscr{E}$ is a set of *follower-of relationships* between developers, defined as $\mathscr{E} = \{d_i \xrightarrow{f} d_j | d_i, d_j \in \mathscr{D}\}$, 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.

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

The organization of the *follower-of* relationships determines the network structure. The developers' social network can be represented as one or more directed graphs, as shown in Figure 14.3, where a graph can assume different topologies. It can be restricted to a hierarchy or can be a peer-based network where developers can mutually follow each other in collaborative and open contexts. An example is the network in Figure 14.3(a). A third kind of topology, see Figure 14.3(b), represents a hybrid case, where a developer is or has been involved in different web application design projects and, maybe depending on the particular application domain, can follow different reference developers (consider, as an example, `dev3`, who declares to follow both `dev4` and `dev8`).

14.4 Model-based service search and ranking

According to application development needs, developers can look for single services or for more services apt to complete existing aggregations. Two search modalities

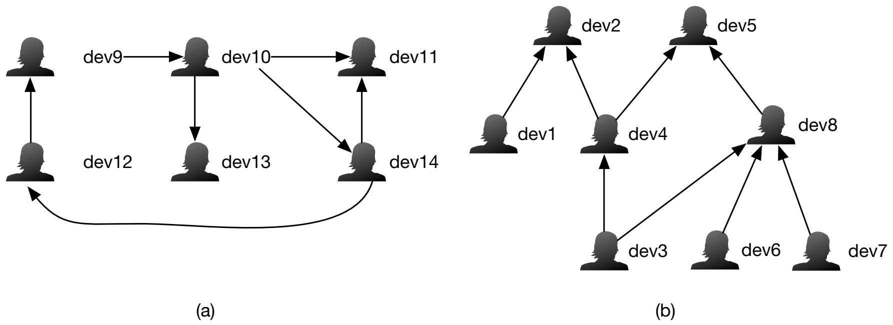

Fig. 14.3 Sample social networks of developers, which present peer-based (a) and hybrid (b) topologies.

can be defined: (i) simple search, and (ii) proactive search. In the *simple search*, the developer receives suggestions about relevant services after explicitly specifying the requested features (e.g., tags, required values for each kind of technical features, and so on). In particular, answering a single request in the context of the simple search modality is based on the component perspective and on the terminological one (see Figure 14.1). In the *proactive search*, the developer does not specify features for the services of interest, because he/she has just a partial idea of what he/she is looking for, and the framework proactively suggests candidate services according to the aggregation that is being developed. Answering to requests according to this modality, in order to complete an existing aggregation, requires using the whole type of knowledge depicted in Figure 14.1, as discussed in the following.

14.4.1 Service request

A service request is formulated according to the following definition.

Definition 14.4. A service request s^r is formally represented as $\langle \mathscr{T}_r, \mathscr{F}_r, g_r \rangle$, where:

- $\mathscr{T}_r$ is a set of terms used to specify what the requester is looking for;
- $\mathscr{F}_r$ is the set of required technical features, that the developer who issues the request can specify for further refining the search constraints; as for the specification of technical features within service description, according to Definition (14.1), $\mathscr{F}_r$ is defined as an array of elements, where each element $\mathscr{F}_r^X$ contains required values for a technical feature X;
- g_r is a set of services, representing the current composition of the aggregation that is being designed; the g_r element is optional.

The presence of the g_r in the request s^r depends on the search target. In particular, in case of searching for a single service (e.g., to search for the first service to be included in a new web application that is being designed) the service request is expressed as $s^r = \langle \mathscr{T}_r, \mathscr{F}_r, g_r \rangle$, where $g_r = \emptyset$.

Answering a service request s^r is based on the following phases: (i) developers' credibility evaluation and ranking; (ii) service search and ranking. In the following, these phases are detailed

14.4.2 Developers' credibility evaluation and ranking

To model the service experience perspective as described in Section 14.3.2, it becomes relevant to estimate the credibility of a developer, who expresses votes. To this purpose, credibility can be assessed based on a majority-based criteria. The basic idea is that, if a given vote on a service does not agree with the majority opinion on that service, the developer's credibility score is decreased, otherwise it is increased. The details of the credibility assessment are given in previous work [12].

Both the credibility scores and the way the social network of developers is organized are used to determine the developer's rank. This type of rank is considered to answer a request, as described in the next Section 14.4.3, in particular to assign a weight to the developer's votes.

Let's suppose d^r be a developer who has submitted a request.The overall rank of a developer $d_i \in \mathscr{D}$, denoted with $dr(d_i)$, is computed as the product of two different ranks, according to the following formula:

$$dr(d_i) = \rho_{rel}^{d^r}(d_i) \cdot \rho_{abs}(d_i) \in [0,1] \tag{14.1}$$

where: (a) a *relative rank* $\rho_{rel}^{d^r}(d_i) \in [0,1]$ 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 services); (b) an *absolute rank* $\rho_{abs}(d_i)$ is based on the overall network of developers and it takes into account the authority degree of d_i in the network independently of the developer d^r, who issued the request. In particular, the authority degree of d_i can be computed by adapting the PageRank metrics (that calculates the authority degree for Web pages based on the incoming links) to the context considered here.

Relative rank.

The relative rank $\rho_{rel}^{d^r}(d_i)$ 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:

$$\rho_{rel}^{d^r}(d_i) = \frac{1}{\ell(d^r, d_i)} \in [0,1] \tag{14.2}$$

If there is no 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 14.3, 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, ℓ(`dev3`,`dev4`)=ℓ(`dev3`,`dev8`)=1, ℓ(`dev3`,`dev5`)=2, and ℓ(`dev3`,`dev6`)=ℓ(`dev3`, `dev11`)=$4+1$=5.

Absolute rank.

The absolute rank $\rho_{abs}(d_i) \in [0,1]$ is evaluated independently of the requester d^r. This rank is composed of two different parts. The first one depends on the number of aggregations designed by d_i, the second one depends on the topology of the network of other developers who declared their interest for past experiences of d_i, that is:

$$\rho_{abs}(d_i) = \frac{1-\alpha}{|\mathscr{D}|} \cdot |\mathscr{G}(d_i)| + \alpha \cdot \sum_{j=1}^{n} \frac{c(d_j) \cdot \rho_{abs}(d_j)}{F(d_j)} \tag{14.3}$$

This expression is an adaptation of the PageRank metrics to the context considered in this chapter. 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 a web application. Therefore, $\sum_i \rho_{abs}(d_i) = 1$. Initially, all developers are assigned with the same probability, that is, $\rho_{abs}(d_i) = 1/|\mathscr{D}|$. Furthermore, at each iteration of the 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: (i) 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 (14.3), where $F(d_j)$ is the number of developers followed by d_j); (ii) a contribution to $\rho_{abs}(d_i)$ is given by the experience of d_i and is therefore proportional to the number $|\mathscr{G}(d_i)|$ of aggregations designed by d_i(see the first term in Equation (14.3)). A damping factor $\alpha \in [0,1]$ is used to balance the two contributions. At each step, a normalization procedure is applied in order to ensure that $\sum_i \rho_{abs}(d_i) = 1$.

The algorithm actually used to compute recursively Equation (14.3) is similar to the one applied for PageRank. In particular, denoting with $\rho_{abs}(d_i, \tau_N)$ the N-th iteration in computing $\rho_{abs}(d_i)$ and with $\mathbf{DR}(\tau_N)$ the column vector whose elements are $\rho_{abs}(d_i, \tau_N)$, it follows that:

$$\mathbf{DR}(\tau_{N+1}) = \frac{1-\alpha}{|\mathscr{D}|} \cdot \begin{bmatrix} |\mathscr{G}(d_1)| \\ |\mathscr{G}(d_2)| \\ \vdots \\ |\mathscr{G}(d_n)| \end{bmatrix} + \alpha \cdot \mathbf{M} \cdot \mathbf{DR}(\tau_N) \tag{14.4}$$

where $\mathbf{M}$ denotes the adjacency matrix properly modified to consider credibility, that is, $M_{ij} = \frac{c(d_j)}{F(d_j)}$ if $d_j \xrightarrow{f} d_i$, zero otherwise. As demonstrated in PageRank, computation formulated in Equation (14.4) reaches a high degree of accuracy within only a few iterations.

Developer (d_i)	$\lvert\mathscr{G}(d_i)\rvert$	*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 14.2 Example of values for developers' features, i.e., number of developed aggregations $\lvert\mathscr{G}(d_i)\rvert$ and credibility $c(d_i)$.

Let's consider Table 14.2, that lists an example with values for developers' features (i.e., number of developed aggregations, credibility). In particular, $\alpha = 0.6$. At time τ_0 $\rho_{abs}(d_i) = 1/|\mathscr{D}| = 0.0714$ for all d_i. During the next iteration:

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

Similarly, $\rho_{abs}(\texttt{dev8}, \tau_1) = 0.1299$. After each iteration, normalization is applied to have $\sum_i \rho_{abs}(d_i) = 1$. In the example, after 5 iterations, the error measured as Euclidean norm of the vector $\mathbf{DR}(\tau_5) - \mathbf{DR}(\tau_4)$ is less than 0.001. At the end, $\rho_{abs}(\texttt{dev4}) = 0.0997$ and $\rho_{abs}(\texttt{dev8}) = 0.0801$.

14.4.3 Service selection and ranking

Service selection is performed by exploiting: (a) tags, used for service semantic characterisation, based on the terminological perspective; (b) past use of services matching the request, based on the aggregation perspective, and (c) technical features, based on the component perspective. All the defined perspectives contribute to quantify the matching between a service $s \in \mathscr{S}$ and a request s^r. In particular, in order to answer service requests, similarity metrics, based on the multi-perspective model, have been defined to quantify service-request matching:

- the **tag similarity**, to evaluate the similarity between the request and each service based on tags, either semantically disambiguated or not; tag similarity is denoted as $TagAff(\{t_{s_i}\}, \{t_{s_j}\}) \in [0,1]$, where $\{t_{s_i}\}$ and $\{t_{s_j}\}$ are compared sets of tags;
- the **aggregation similarity**, to evaluate the similarity between the request and each service based on average similarity between the aggregation that is being developed and aggregations where the service s has been used in the past, re-

spectively; this similarity is denoted as $AggSim(g_o, g_p) \in [0,1]$, where g_o and g_p are compared aggregations; the rationale here is that the more similar the services used in the two compared aggregations according to their similarity, the more similar the two aggregations;

- the **technical feature similarity**, to evaluate the similarity between the request and each service based on technical features; similarity for a technical feature X is denoted as $TechSim^X(\{f_{s_i}\}, \{f_{s_j}\}) \in [0,1]$, where $\{f_{s_i}\}$ and $\{f_{s_j}\}$ are compared sets of values allowed for feature X.

The overall similarity between two services, computed as a linear combination of the above three similarities, is denoted as $Sim(s_i, s_j) \in [0,1]$. Overall testing and setup of weights, to proper balance tag, technical feature and aggregation similarity, have been discussed in [7].

The aim is to combine this overall similarity value with a ranking function $\rho_{serv} : \mathscr{S} \mapsto [0,1]$, that is based on: (i) the ranking of developers who used $s \in \mathscr{S}$; (ii) the votes $v(s, g_i, d_k)$ assigned to s by each developer d_k who 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:

$$\rho_{serv}(s) = \frac{\sum_{k=1}^{n} \sum_{i=1}^{m_k} dr(d_k) \cdot v(s, g_i, d_k)}{N} \in [0,1] \tag{14.5}$$

where $d_k \in \mathscr{D}$, for each 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 overall rank of developer d_k with respect to the request s^r, as discussed in the previous section. 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 $N = \sum_{k=1}^{n} m_k$. The overall service similarity $Sim(s^r, s)$ and $\rho_{serv}(s)$ elements are finally combined in the following harmonic mean in order to rank service s:

$$rank(s) = \frac{2 \cdot \rho_{serv}(s) \cdot Sim(s^r, s)}{\rho_{serv}(s) + Sim(s^r, s)} \in [0,1] \tag{14.6}$$

14.5 The WISeR system for service selection

The WISeR system (Web apI Search and Ranking) has been developed as web application and it implements the framework and the multi-perspective model described in the previous sections. The system functional architecture is shown in Figure 14.4. The WISeR core module is the *Matching and Ranking Engine*, that embeds the similarity metrics presented in previous section and is invoked through the *Search GUI*. Given a service published within a repository, proper wrappers (implemented within the *Web API Features Extractor*) are used to extract service features and

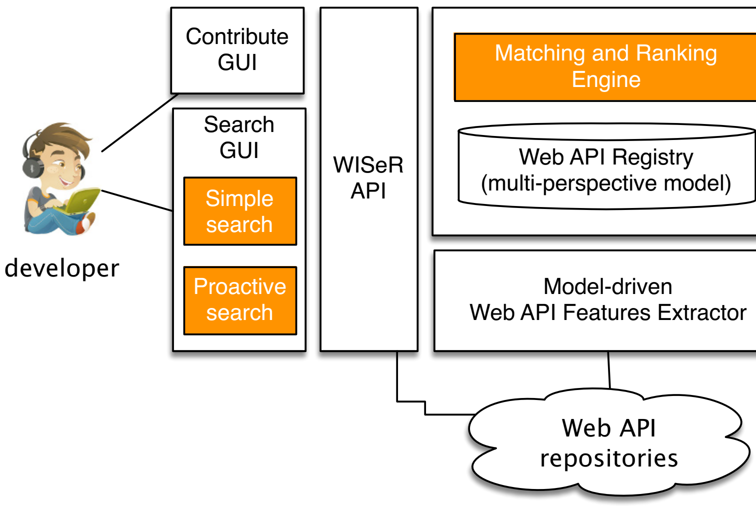
Contribute GUI
Search GUI
Simple search
Proactive search
WISeR API
Matching and Ranking Engine
Web API Registry (multi-perspective model)
Model-driven Web API Features Extractor
Web API repositories
developer

Fig. 14.4 WISeR functional architecture.

store them within the internal *Web API Registry*. The current implementation of WISeR is built upon the `ProgrammableWeb` and `Mashape` repositories. The *Web API Registry* stores categories, technical features, terminological equipment, used for service search and ranking. Note that in WISeR, by means of a specific interface, *Contribute GUI*, developers can add features that are not present in the original repositories, but are exploited by the system matching and ranking techniques. It is the case, for example, of aggregation-contextual votes. To add information related to service experience, developers registration is required. The search interface, *Search GUI*, permits to use both the WISeR service selection modalities and the original keyword-based search mechanisms available in the repositories. The *Search GUI* also embeds a ranking function based on the service publication date: all services are listed starting from the most recently published one.

The WISeR system has been used for experiments aimed to evaluate the effectiveness in service selection and developer's ranking [12]. In particular, these experiments have confirmed the positive contribution and importance of using the multi-perspective model to improve the selection precision.

14.6 Conclusions and Future Work

The diffusion of Web Oriented Architecture and data intensive web application development, relying on the selection and reuse of third party components, called for new data providing service search and ranking approaches. A conceptual framework that merges different Web data service features becomes crucial to build applications starting from ready-to-use components. Beyond descriptive features like categories, tags and technical features, the choice among different alternatives might be inspired by the experiences of other developers in using them, such as developers' ratings and similar applications where services have been included. In this chapter, a conceptual framework is described to provide: (i) a multi-perspective model for service description, that also includes a social-based perspective, focused on the community of developers, their mutual relationships and their estimated credibility in web application development; (ii) a collection of search and ranking techniques that rely on the model; (iii) a prototype system that implements the unified conceptual framework on top of service repositories. Future work will focus on advanced service search and ranking techniques to enable dynamic exploration and access on data of interest, also considering application domains where Internet of Things (IoT) and Internet of Services (IoS) technologies enable sharing and integration of huge quantity of heterogeneous data.

References

1. W. Tan, Y. Fan, A. Ghoneim, M. Hossain, S. Dustdar, From the Service-Oriented Architecture to the Web API Economy, IEEE Internet Computing 20 (4) (2016) 64–68.
2. B. Tapia, R. Torres, H. Astudillo, Simplifying mashup component selection with a combined similarity- and social-based technique, in: Proceedings of the 5th International Workshop on Web APIs and Service Mashups, 2011, pp. 1–8.
3. B. Cao, M. Tang, X. Huang, Cscf: A mashup service recommendation approach based on content similarity and collaborative filtering, International Journal of Grid and Distributed Computing 7 (2) (2014) 163–172.
4. A. Maaradji, H. Hacid, R. Skraba, A. Lateef, J. Daigremont, N. Crespi, Social-based Web Services Discovery and Composition for Step-by-Step Mashup Completion, in: Proc. of Int. Conference on Web Services (ICWS), 2011.
5. C. Li, R. Z. Z. Huai, H. Sun, A novel approach for api recommendation in mashup development, in: Proc. of Int. Conference on Web Services (ICWS), 2014, pp. 289–296.
6. K. Gomadam, A. Ranabahu, M. Nagarajan, A. Sheth, K. Verma, A Faceted Classification Based Approach to Search and Rank Web APIs, in: Proc. of International Conference on Web Services (ICWS), 2008, pp. 177–184.
7. D. Bianchini, V. De Antonellis, M. Melchiori, A Multi-perspective Framework for Web API Search in Enterprise Mashup Design (Best Paper), in: Proc. of 25th Int. Conference on Advanced Information Systems Engineering (CAiSE), Vol. LNCS 7908, 2013, pp. 353–368.
8. D. Archer, L. Delcambre, D. Maier, User Trust and Judgments in a Curated Database with Explicit Provenance, Search of Elegance in the Theory and Practice of Computation (2013) 89–111.
9. A. Olivé, Conceptual Modeling in Agile Information Systems Development, in: Proc. of the 16th International Conference on Enterprise Information Systems (ICEIS14), 2014.
10. M. González, L. Cernuzzi, N. Aquino, O. Pastor, Developing web applications for different architectures: The MoWebA approach, in: Proc. of IEEE International Conference on Research Challenges in Information Science (RCIS2016), 2016, pp. 1–11.
11. D. Bianchini, V. D. Antonellis, M. Melchiori, WISeR: A Multi-dimensional Framework for Searching and Ranking Web APIs, ACM Transactions on the Web, (in press).
12. D. Bianchini, V. D. Antonellis, M. Melchiori, The role of developers' social relationships in improving service selection, International Journal of Web Information Systems 12 (4) (2016) 477–503.
13. O. Díaz, I. Aldalur, C. Arellano, H. Medina, S. Firmenich, Web Mashups with WebMakeup, in: Proc. of ICWE Rapid Mashup Challenge workshop (RMC2015), 2015, pp. 82–97.

14. A. Riabov, E. Boillet, M. Feblowitz, Z. Liu, A. Ranganathan, Wishful search: interactive composition of data mashups, in: Proc. of the 19th Int. World Wide Web Conference (WWW'08), Beijin, China, 2008, pp. 775–784.
15. O. Greenshpan, T. Milo, N. Polyzotis, Autocompletion for Mashups, in: Proc. of the 35th Int. Conference on Very Large DataBases (VLDB), Lyon, France, 2009, pp. 538–549.
16. M. Picozzi, M. Rodolfi, C. Cappiello, M. Matera, Quality-based recommendations for mashup composition, in: Proceedings of the 10th international conference on Current trends in web engineering (ICWE), 2010, pp. 360–371.
17. M. Kayaalp, T. Ozyer, S. T. Ozyer, A mash-up application utilizing hybridized filtering techniques for recommending events at a social networking site, Social Network Analysis and Mining 1 (3) (2011) 231–239.
18. B. Cao, J. Liu, M. Tang, Z. Zheng, G. Wang, Mashup Service Recommendation based on User Interest and Social Network, in: Proc. of Int. Conference on Web Services (ICWS), 2013.