A Social Network-based Framework for Data Services Selection in Modern Web Application Design

Devis Bianchini, Valeria De Antonellis, Michele Melchiori

Dept. of Information Engineering University of Brescia
Via Branze, 38 - 25123 Brescia (Italy)
{devis.bianchini|valeria.deantonellis|michele.melchiori}@unibs.it

Abstract. In recent years the design of enterprise Web applications is more and more based on the integration of resources delivered through data services from outside the organization boundaries. Searching and composing existing data services offer many advantages, namely, the availability of widespread solutions in the form of services and data shared over the Web, and reduced development costs. In this scenario, new methods for speeding up the design process are emerging and, in particular, developers' social networks have been established, where developers follow other developers to learn from their choices in selecting suitable services. In this paper, we propose a framework to support data service selection for modern Web application design, by also considering the developers' social network. The network of social relationships, properly weighted with the developers' credibility, is used to compute developers' rank. This rank qualifies developers' experience in selecting data services.

1 Introduction

Modern enterprise Web application design increasingly relies on the selection and composition of external services, that provide access to resources from outside the organization boundaries. Availability of data services, that meet these requirements, are becoming more and more important, as also witnessed by the growth of public Web API repositories (e.g., `ProgrammableWeb.com`, `Mashape.com`). In particular, the lightweight description of RESTful services (often referred to as Web APIs), compared to the standard description of SOAP-based service capabilities (e.g., WSDL), is the main reason of their increasing success.

Starting from RESTful lightweight description, recommendation of the most suitable services to be adopted within a Web application development process combines several factors, beyond functional and non-functional requirements [1, 2]. Increasing research effort is being devoted to: (i) the application of collaborative filtering techniques to recommend services based on users' ratings [3, 4]; (ii) the exploitation of API popularity (i.e., number of times a Web API has been

In: S. España, M. Ivanović, M. Savić (eds.): Proceedings of the CAiSE'16 Forum at the 28th International Conference on Advanced Information Systems Engineering, Ljubljana, Slovenia, 13-17.6.2016, published at http://ceur-ws.org

used) to weight API relevance [5]; (iii) the computation of API co-occurrence into existing mashups to rank APIs for selection purposes [6]; (iv) the exploitation of users' social relationships to suggest RESTful components to the user u considering other users who are similar to u in the social network [7]. These efforts demonstrate how the choice among different alternatives might be inspired by the experiences of other developers in using them, such as developers' ratings, comments, similar applications where APIs have been included. Indeed, it is frequent that a developer searches for advices based on design experiences of other known developers, rather than only relying on votes/choices of generic users over the Web [8].

In this paper, we describe an extension to WISeR [9], our Web API discovery and selection framework for modern Web application design. This extension exploits the developers' social network for improved service selection. We currently rely on "follower-of" relationships of `Mashape.com`, that is the largest repository where developers can follow each other for Web API discovery and selection purposes. In particular, given a set of data services that are candidate for selection, services are sorted based also on a ranking of developers, in the social network, who used these services in the past to develop their own applications. The proposed developers' rank metric is computed by considering: (i) the topology of social relationships between developers; (ii) the number of Web applications developed by each designer; (iii) the developers' *credibility*, estimated starting from their evaluations on services. Existing service recommendation techniques still remain valid and can be integrated within the framework as well.

The paper is organized as follows: Section 2 describes the social network of developers, underlying the framework, and the developers' credibility assessment; Section 3 explains techniques behind developers' ranking for improving data service selection process and discusses a preliminary validation of the framework; finally, Section 4 closes the paper.

2 Social network-based model for data service selection

To enable social network-based selection of data services, we extended an existing framework, WISeR (Web apI Search and Ranking) [9], with the developers' rank functionalities. WISeR is based on a multi-layered model, developed over different perspectives:

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

Formally, data services and aggregations are defined as follows.

Definition 1. *We define a data service s (hereafter, service) as an operation/method/query to access data of a web source, 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 $\langle n_s, \{t_s\}\rangle$, where: n_s is the service name; $\{t_s\}$ is a set of tags. We denote with $\mathcal{S}$ the overall set of available services.

Definition 2. *A service aggregation represents a set of services usable to deploy a 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.*

According to this vision, best representatives of data services are resource-oriented services (i.e., RESTful ones). For services that present a structured, operation-oriented description (e.g., WSDL), we consider only data they work on, represented through tags.

Developers, who used services to design their applications, are organized in a social network, defined as follows.

Definition 3. *The social network of developers is a pair $SN = \langle \mathcal{D}, \mathcal{E}\rangle$, where: (a) $\mathcal{D}$ is the set of developers; (b) $\mathcal{E}$ is a set of follower-of relationships between developers, defined as $\mathcal{E} = \{d_i \xrightarrow{f} d_j | d_i, d_j \in \mathcal{D}\}$, 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.*

Each developer $d_i \in \mathcal{D}$ 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, in order to design web applications, that is, $\mathcal{D}^* = \{d_k | d_i \xrightarrow{f} d_k \in \mathcal{E}\}$. The organization of the *follower-of* relationships determines the network structure as extracted from `Mashape` repository. The developers' social network can be represented as one or more directed graphs, as shown in Figure 1.

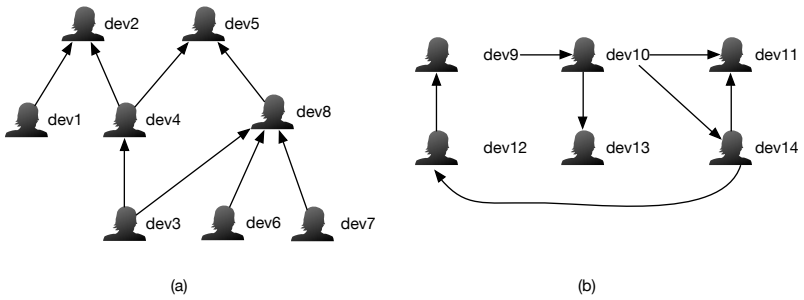

Fig. 1. Sample social networks of developers, that present a hierarchical (a) and a peer-based structure (b).

2.1 Developers' credibility

In our approach, a developer may assign votes to services used in the applications. In particular, since developers exchange their experiences in using services, votes become an enabling feature to this purpose. Following this vision, for example, all the most popular Web API repositories include a rating system. 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 (*aggregation-contextual rating*), according to the following definition.

Definition 4. *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.*

Aggregation-contextual rating helps in properly weighting votes assigned to services. 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 one that is being developed, according to the aggregation similarity, $AggSim()$, introduced in [9]. Moreover, we include credibility evaluation techniques in the approach, inspired by the ones defined in [10], with respect to which we introduced the notion of aggregation-contextual rating. 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, set by d_i. The majority opinion on s_j is hence represented by the most densely populated cluster, whose centroid is considered as the majority rating:

$$M(s_j) = centroid(max_{i=1}^{k}(\mathcal{C}_i)) \tag{1}$$

where $\mathcal{C}_i$ is the i-th cluster, $k = |\{\mathcal{C}_i)\}|$ 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 $\left\lceil \sqrt{N/2} \right\rceil$ 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/she already assigned n votes, and given 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, the new credibility value for the developer 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 term $1 - |M(s_j) - v(s_j, g_m, d_i)|$ tends to 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 looses too quickly his/her credibility for few assigned votes that are not aligned with the majority opinion). Viceversa, if the vote $v(s_j, g_m, d_i)$ is close to $M(s_j)$, then 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 (max credibility). Initial values $c_0(d_i)$ are set to 0.5. Note that credibility of a developer with a high value of n and who is assigning a vote different from the ones expressed by the majority, is reduced of a low 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. The rationale for clustering votes can be explained with the help if an example: if a service receives votes 1,1,1,2,9,9,8, and we adopt an average-based model, we obtain an overall rating of 4.4. Actually, this rating is not representative of the depicted situation, where the majority of voters gives a low vote.

3 Ranking of developers

Let's consider a developer d^r, who is searching for a service 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 that is being developed. If s_1 and s_2 are equally relevant for the developer d^r according to the overall similarity function $Sim(s_1, s_2)$ (defined in [9]), 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:

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

where: (a) 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); (b) an *absolute ranking* $\rho_{abs}(d_i)$ is based on the overall network of developers, to take into account the centrality of d_i in the network independently of the developer d^r, who issued the request.

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 topology shown in Figure 1, 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`)=2 + 1=3.

The absolute ranking $\rho_{abs}(d_i) \in [0, 1]$ is evaluated no matter is 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, 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 PageRank metrics to the context we are considering. 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, $\rho_{abs}(d_i) = 1/|\mathcal{D}|$. 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: (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 (5), 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 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 (i) and in (ii). At each step, a normalization procedure is applied in order to ensure that $\sum_i \rho_{abs}(d_i) = 1$.

The computation algorithm used for 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:

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

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 (6) reaches a high degree of accuracy within only a few iterations.

Framework validation. Since there are no benchmarks to compare our approach with similar efforts, we built a dataset to perform a validation on the framework. We used wrappers to extract from `Mashape` repository service descriptions and the developers who follow/consume those services, including the

network of their *follower-of* relationships. We completed the dataset construction by adding the number of developed aggregations and developers' credibility values in order to obtain the following classes of developers: (i) developers with a high number of followers, who in turn have several followers as well, with high credibility ($0.7 \leq c(d_i) \leq 1.0$) and several designed aggregations ($|\mathcal{G}(d_i)| \geq 3$), like `dev5` in Figure 1; (ii) developers who present few followers, medium credibility ($0.4 \leq c(d_i) < 0.7$) and several designed aggregations ($|\mathcal{G}(d_i)| \geq 3$); (iii) developers who present few followers, medium credibility and few designed aggregations ($|\mathcal{G}(d_i)| < 3$); (iv) developers who present few followers, low credibility ($0 \leq c(d_i) < 0.4$) and who designed few aggregations. Intuitively, the above mentioned classes are ordered according to an increasing rank of developers, where developers in class (i) are top ranked. This rank will be considered as reference for setup of the damping factor α and approach validation.

We run the system randomly selecting a subset of developers as requesters, we computed the Kendall tau distance k for each run with respect to the reference developers' rank described above and we computed the average value for k. We also repeated the same experiment considering two different configurations of the approach, namely: (a) an optimistic configuration, where we considered all developers having maximum credibility (i.e., $c(d_i) = 1.0$, $\forall i$); (b) a configuration biased on the requester, where only the relative ranking $\rho_{rel}^{d^r}(d_i)$ has been considered. We kept the value $\alpha = 0.6$ for the damping factor. The results of this validation are shown in Table 1. The average value of the Kendall tau distance shows the accuracy of our ranking solution compared to an intuitive sorting of developers described above considering the four classes of developers (i)-(iv). Validation results also demonstrate the impact of considering developers' credibility and the topology of social relationships between developers through the computation of the absolute ranking $\rho_{abs}(d_i)$. Indeed, if we do not consider different levels of credibility (optimistic case), the quality of ranking slightly decreases. However, decrement of ranking quality is even more evident if we do not consider the absolute ranking (biased case), thus demonstrating that the relative viewpoint of the requester is not able to correctly identify the centrality of other developers in the social network.

	Credibility $c(d_i)$	**Dumping factor** α	**Kendall tau distance** $k \in [0, 1]$
Biased case	-	$\alpha = 0.6$	0.6026
Optimistic case	$c(d_i) = 1$, $\forall d_i$	$\alpha = 0.6$	0.1795
Our approach	$c(d_i) \in [0, 1]$, $\forall d_i$	$\alpha = 0.6$	0.0360

Table 1. Average Kendal tau distances computed for the approach validation

4 Conclusions

In this paper, we discussed how developers' social relationships, as well as their credibility, can be properly exploited to support data service selection. In particular, we proposed a framework for ranking developers by considering both a relative and an absolute perspective. The framework interacts with the `Mashape` repository, the largest service repository where developers can follow each other in a social network. Other developers' social networks, such as `GitHub`, are not specifically meant for data service selection, while other highly populated Web API repositories (e.g., `ProgrammableWeb`) do not present social relationships between developers. Further studies are on going 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.

References

1. W. Xu, J. Cao, L. Hu, J. Wang, M. Li, A social-aware service recommendation approach for mashup creation, in: IEEE International Conference on Web Services, 2013.
2. L. Yao, S. Zheng, A. Segev, J. Yu, Recommending web services via combining collaborative filtering with content-based features, in: IEEE International Conference on Web Services, 2013.
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. R. Balakrishnan, S. Kambhampati, J. Manishkumar, Assessing Relevance and Trust of the Deep Web Sources and Results Based on Inter-Source Agreement, ACM Transactions on the Web 7 (2) (2013) 32 pages.
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. 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.
7. 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.
8. A. Fuxman, P. Giorgini, M. Kolp, J. Mylopoulos, Information Systems as Social Structures, Formal Ontology in Information Systems (2001) 12–21.
9. 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.
10. Z. Malik, A. Bouguettaya, RATEWeb: Reputation Assessment for Trust Establishment among Web Services, VLBD Journal 18 (2009) 885–911.