Semantic Collaborative Tagging for Web APIs Sharing and Reuse

Devis Bianchini, Valeria De Antonellis, and Michele Melchiori

Dept. of Information Engineering University of Brescia
Via Branze, 38 - 25123 Brescia, Italy
{bianchin,deantone,melchior}@ing.unibs.it

Abstract. Sharing and reuse of Web APIs for fast development of Web applications require advanced searching facilities to enable Web designers to find the Web APIs they need. In this paper we describe a Web API semantic collaborative tagging system to be implemented on top of the public ProgrammableWeb Web API repository. The system is designed to be used in a social context: the designers can take actively part in the semantic tagging of Web APIs, thus sharing their experience in developing their own Web applications. Moreover, they can exploit new searching facilities to find out relevant Web APIs according to different search scenarios and reuse them for fast deployment of new applications. To this aim, they rely in an hybrid fashion on the semantic tags and on the collective knowledge derived from past designers' experiences. Proper matching and ranking metrics are defined and applied during Web API searching.

1 Introduction

Sharing and reuse of Web APIs is becoming a very popular way to quickly develop Web mashups, that is, low-cost, personalized Web applications, designed and implemented to be used for short periods of time (also referred as *situational applications*). To this aim, the ProgrammableWeb API public repository has been made available, where Web API providers share their own components and Web designers can look for Web APIs they need to compose new Web applications without implementing them from scratch. The repository registers almost 5,800 Web APIs (a number that is continuously growing[1]) and presents methods to programmatically retrieve the registered Web APIs and also to track all the mashups which have been developed starting from them. Currently, the repository contains more than 6,600 Web mashups.

In this context, enabling Web designers to effectively find Web APIs they need is becoming a more and more crucial asset. Some solutions propose the definition of component models [1,2] to support fast Web application development. Other solutions suggest the introduction of a Web API semantic characterization to

[1] See http://www.programmableweb.com/: the last access on April 30th, 2012 counts 5,792 Web APIs.

M. Brambilla, T. Tokuda, and R. Tolksdorf (Eds.): ICWE 2012, LNCS 7387, pp. 76–90, 2012.

face issues such as the heterogeneity across Web API descriptions [3]. However, the use of new Web API models on top of the ProgrammableWeb repository, although improves Web API retrieval and composition, introduces an additional learning effort for Web API providers who must adopt the models. Such an additional requirement is often not feasible in a social scenario, where the average skill of Web designers prevents from using complex models other than common Web programming technologies. In this paper we propose a lightweight semantic tagging system to be deployed on top of the ProgrammableWeb repository to be used in a social context. Web designers can take actively part in the semantic tagging of Web APIs, thus sharing their experience in developing their own Web applications, going beyond the limitations of traditional tags, which lack in facing ambiguities such as polisemy and omonyms. Moreover, Web designers can also exploit the search facilities of the system to find out relevant Web APIs according to different search scenarios and reuse them for fast deployment of new applications. To this aim, they rely in an hybrid fashion on the semantic tags and on the collective knowledge derived from past designers' experiences in developing mashups.

In the next section we describe the scenarios which motivated our work. Section 3 contains a formalization of proper matching and ranking metrics based on the Web API semantic tags and collective knowledge to be applied during search. In Section 4 we describe the implementation of the proposed system. Section 5 shows a preliminary validation of the system and a comparison with the state of the art. Section 6 closes the paper.

2 Motivations and Open Issues

Consider a simple example. Danielle is a designer who aims at building a Web application for her friends, to share pictures by posting photos on the Web site and showing with marks on a map the locations where photos have been taken. Danielle does not want to implement the Web application from scratch, since, for instance, the design of a Web interface for displaying points on a map is a time-consuming task and Danielle does not have time and the required advanced skills. Nevertheless, already available Web APIs, such as the Google Maps APIs, have been implemented for this purpose. She has to search for existing Web APIs to post pictures and personalize maps and has to properly wire them in the mashup. She inspects the ProgrammableWeb repository (see Figure 1), where she can: (i) find Web APIs by specifying keywords that will be matched against the descriptions associated with available Web APIs in the repository; (ii) filter available Web APIs according to their category, the company which proposed the APIs, the adopted protocols and data formats; (iii) find mashups composed of the available Web APIs. On ProgrammableWeb Danielle can not:

1. specify both the features of the Web APIs to search for and of the Web mashups which the Web APIs will be used in, to perform more advanced search; for instance, Danielle can not specify that she needs a mapping Web

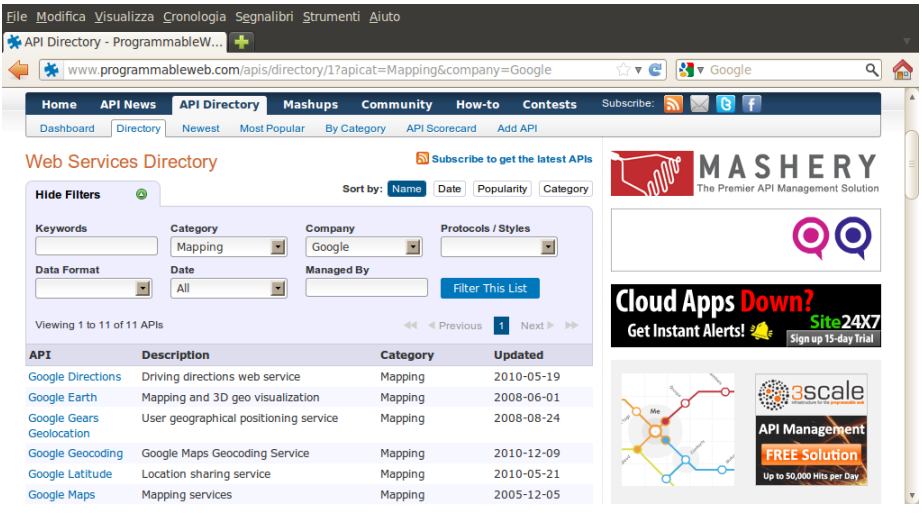

Fig. 1. An example of Web API search on ProgrammableWeb.com

API to be used in combination with a Web API for posting pictures; if she looks for a mapping Web API, the system returns about 210 results, which can be restricted to 11 results by selecting the most famous Google company (see Figure 1), but not all the returned APIs enable the definition of markers for displaying points on a map;

2. avoid limitations of traditional tag-based search, specifically false positives and false negatives due to the tag polisemy (that is, the same tag refers to different concepts) and tag omonyms (i.e., the same concept is pointed out using different tags); false positives are APIs incorrectly included among the search results, they are affected by polisemy; false negatives are APIs incorrectly excluded from search results, they are affected by omonyms;
3. be assisted in more articulated developing scenarios, where Danielle needs a proactive support for her search; for instance, let us suppose that Danielle already has at her disposal a Web mashup which combines a Web API for searching sightseeing locations and a Web API to display them on a map; ProgrammableWeb is neither able to proactively suggest different APIs that can be substituted to the sightseeing search API and properly wired with the map API (thus minimizing the efforts required to adapt the new Web API into the existing mashup) nor can suggest other Web APIs (e.g., a Twitter API) to be added to the mashup, because of many existing mashups where Web designers put together such kinds of Web APIs.

3 Semantic and Social Characterization of Web APIs

Given the open issues highlighted above, we inferred the need of both a semantic characterization of Web APIs, to face polisemy and omonyms problems,

and a social characterization of Web APIs, to rely on the past experiences of Web designers and enable the support for more articulated developing scenarios (see point 3 in Section 2). Formally, we denote a Web API description $\mathcal{W}$ as $\langle \mathcal{W}_{Sem}, \mathcal{W}_{Soc} \rangle$, where $\mathcal{W}_{Sem}$ and $\mathcal{W}_{Soc}$ are the semantic and social characterization of $\mathcal{W}$, respectively.

3.1 Semantic Characterization

The semantic characterization of a Web API $\mathcal{W}$ is defined as follows:

$$\mathcal{W}_{Sem} = \langle c_{\mathcal{W}}, \{t_{\mathcal{W}}\}, \{m_{\mathcal{W}}\} \rangle \tag{1}$$

The elements of such a characterization are discussed in the following and are extracted both from the contents of the ProgrammableWeb repository and from the information provided by the designers.

Web API category $c_{\mathcal{W}}$. It is used to provide a high level classification of the Web API. We rely on the classification of Web APIs provided by ProgrammableWeb, composed of 67 categories such as `mapping`, `payment`, `search`.

Web API semantic tags $t_{\mathcal{W}}$. Tags are used to provide a fine-grained semantic characterization of the Web API. To this purpose, we rely on tags defined by the designers who adopted the Web API. During the assignment of such tags, sense disambiguation techniques based on the WordNet [4] lexical system are applied. In WordNet the meaning of terms is defined by means of *synsets*. Each synset has a human readable definition and a set of synonyms. In our model, a Web API semantic tag $t_{\mathcal{W}}$ is a triple, composed of: (i) the term itself (namely, $t^0_{\mathcal{W}}$) extracted from WordNet; (ii) the set $t^{syn}_{\mathcal{W}}$ of all the terms in the same synset of $t^0_{\mathcal{W}}$; (iii) the human readable definition $t^d_{\mathcal{W}}$ associated with the synset. The designer is supported in the selection of the synset to better specify the meaning of a tag as shown in Section 4.

Web mashup semantic tags $m_{\mathcal{W}}$. When the Web API is tagged, the designer is also required to add a set of semantic tags $m_{\mathcal{W}}$ that describe the Web mashup where the Web API has been used. The semantic tags $m_{\mathcal{W}}$ has the same structure of semantic tags in $\{t_{\mathcal{W}}\}$.

An example of semantic characterization of the `Flickr` Web API to be used in a file sharing application could be the following:

$Flickr_{Sem}$ = ⟨`Photos`, {⟨`photo`, {`photograph`, `exposure`, `picture`, `pic`}, "a representation of a person or scene in the form of a print or transparent slide or in digital format"⟩},
{⟨`file`, {`date_file`}, "a set of related records (either written or electronic) kept together"⟩,
⟨`sharing`, {}, "using or enjoying something jointly with others"⟩}

Web API category, semantic tags in $\{t_{\mathcal{W}}\}$ and Web mashup semantic tags in $\{m_{\mathcal{W}}\}$ will be used to empower the Web API search by applying advanced matching techniques described in Section 3.4.

3.2 Social Characterization

In a Web 2.0 context, the suggestion of a Web API to be used in a mashup should also consider as relevant past experiences of the designers who adopted the Web APIs in their mashups. We denote these aspects as the social characterization of the Web API $\mathcal{W}$:

$$\mathcal{W}_{Soc} = \{d \in \mathcal{D}_{\mathcal{W}} | d = \langle \sigma, \mu, \{\mathcal{W}_k\} \rangle\} \quad (2)$$

where, for each designer $d \in \mathcal{D}_{\mathcal{W}}$, who used the Web API $\mathcal{W}$, σ is the designer's skill for developing Web applications and μ is a quantitative rating (within the range [0,1]) given by the designer to the Web API $\mathcal{W}$. Social characterization of the Web API $\mathcal{W}$ is further refined by asking the designer, during semantic tagging, to specify other Web APIs $\mathcal{W}_k$ that he/she used together with $\mathcal{W}$ in the same Web mashup. It is worth mentioning that if a Web API has been adopted by designers with high skill, the system should better rank such Web API with respect to the other ones among the search results. The skill σ is collected during designer's registration to the system (see Section 4). A set of options are proposed to be selected by the designer, ranging from `unexperienced` to `expert`, and are uniformly mapped into the [0,1] range, with `unexperienced`=0 and `expert`=1. The capability of the system to automatically update the designers' skills on the basis of the number of developed mashups and their complexity (for instance, based on the size of developed mashups as the number of included APIs) will be investigated as future work. Also the set $\{\mathcal{W}_k\}$ could be loaded directly from the mashups stored on ProgrammableWeb if the Web designer is registered on the repository Web site.

The value of μ is selected by the designer according to the 9-point Scoring System[2]. This scoring system has few rating options (only nine) to increase potential reliability and consistency and with sufficient range and appropriate anchors to encourage designers to use the full scale. During the rating, we provided the designer with the set of options that are mapped into the [0,1] range, as shown in Table 1.

Table 1. The 9-point Scoring System for the classification of designers' Web API rating

Rating (additional guidance on strengths/weaknesses)	Score
Poor (*completely useless and wrong*)	0.2
Marginal (*several problems during execution*)	0.3
Fair (*slow and cumbersome*)	0.4
Satisfactory (*small performance penalty*)	0.5
Good (*minimum application requirements are satisfied*)	0.6
Very Good (*good performance and minimum application requirements are satisfied*)	0.7
Excellent (*discreet performance and satisfying functionalities*)	0.8
Outstanding (*very good performances and functionalities*)	0.9
Exceptional (*very good performances and functionalities and easy to use*)	1.0

[2] `http://www.nhlbi.nih.gov/funding/policies/nine_point_scoring_system_and_program_project_review.htm`.

An example of social characterization of two photo sharing Web APIs, `Flickr` and `23hq.com`, used in Web applications like the one described in the motivating scenarios, is the following:

`Flickr`	$d_1 = \langle 1$ (`expert`), 0.7 (`excellent`), {`GoogleMaps`, `del.icio.us`}$\rangle$
	$d_2 = \langle 1$ (`expert`), 0.6 (`very good`), {`GoogleMaps`}$\rangle$
`23hq.com`	$d_3 = \langle 1$ (`expert`), 0.4 (`satisfactory`), {`GoogleMaps`, `SilverlightStreaming`, `YouTube`}$\rangle$
	$d_4 = \langle 0.5$ (`medium`), 0.6 (`very good`), {`YouTube`, `Twitter`, `GoogleMaps`, `del.icio.us`, `Amazon`}$\rangle$

In the example, `Flickr` has been rated as excellent and very good by two experts, while `23hq.com` as satisfactory by an expert and very good by a medium-skilled designer. All the designers used these Web APIs in past applications together with the Google Maps API. Social characterization of the available Web APIs in the ProgrammableWeb repository is exploited for ranking purposes after search. Given a set of Web APIs among search results, they are ranked according to the ratings of other designers that used such APIs in the past, taking into account their skills. Ranking metrics will be detailed in Section 3.4.

3.3 Web APIs Search Scenarios

The semantic and social characterization of Web APIs described above enable to match a request $\mathcal{W}^r$ against the $\mathcal{W}_{Sem}$ and $\mathcal{W}_{Soc}$ of available Web APIs in the repository and to rank the search results with respect to designers' skills and ratings. Formally, we define a request $\mathcal{W}^r$ for a Web API as follows:

$$\mathcal{W}^r = \langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{m^r_{\mathcal{W}}\}, \{\mathcal{W}_h\}\rangle \tag{3}$$

where $c^r_{\mathcal{W}}$ is the requested category, $\{t^r_{\mathcal{W}}\}$ is a set of semantic tags specified for the Web API to search for, $\{m^r_{\mathcal{W}}\}$ is a set of semantic tags featuring the Web mashup in which the Web API to search for should be used, if any, $\{\mathcal{W}_h\}$ is the set of Web APIs already included in such a mashup, if any, which the Web API to search for should be wired with. We distinguish two different search scenarios:

- in the first scenario, Danielle is looking for a Web API to start the development of a new Web mashup; to this aim, she specifies a category $c^r_{\mathcal{W}}$ and a set of semantic tags $\{t^r_{\mathcal{W}}\}$; Danielle has not in mind any mashup where the Web API to search for should be used; the request is formalized as $\mathcal{W}^r_1 = \langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}\rangle$; we denote this scenario as *simple search*; a variant of this scenario is the one where Danielle has already in mind a set of semantic tags $\{m^r_{\mathcal{W}}\}$ which denote the mashup where the Web API to search for should be used; the request is formalized as $\mathcal{W}^r_2 = \langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{m^r_{\mathcal{W}}\}\rangle$ and we denote this variant as *advanced search*;
- in a second scenario, Danielle has already built or started the construction of a Web mashup, composed of a set of Web APIs $\{\mathcal{W}_h\}$, but she has to complete it and the system could suggest the best Web API that can be wired with the other Web APIs already within the mashup; the request is formalized as $\mathcal{W}^r_3 = \langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{m^r_{\mathcal{W}}\}, \{\mathcal{W}_h\}\rangle$ and we denote it as *completion search*; in a variant of this scenario, Danielle has no preferences on the Web

API to search for (i.e., $\{t^r_{\mathcal{W}}\}$ is empty) and she totally relies on the system that should proactively suggest to Danielle which APIs could be added on the basis of the semantic tags $\{m^r_{\mathcal{W}}\}$ on the mashup that is being developed, the set $\{\mathcal{W}_h\}$ of Web APIs already included in the mashup and past experiences of mashups, registered within the repository; in this case, the request is formalized as $\mathcal{W}^r_4 = \langle\{m^r_{\mathcal{W}}\}, \{\mathcal{W}_h\}\rangle$ and we denote it as *proactive completion search*.

For instance, an example of request formulated in the *completion search* scenario to find a Web API in the category `Photos` to be used for picture sharing together with the Google Maps Web API can be represented as follows:

$\mathcal{W}^r$ = ⟨`Photos`, {⟨`picture`, {`photograph`, `photo`, `exposure`, `pic`}, "a representation of a person or scene in the form of a print or transparent slide or in digital format"⟩}, {⟨`picture`, {`photograph`, `photo`, `exposure`, `pic`}, "a representation of a person or scene in the form of a print or transparent slide or in digital format"⟩, ⟨`sharing`, {}, "using or enjoying something jointly with others"⟩}, {`GoogleMaps`}⟩

where $\{t^r_{\mathcal{W}}\}$ = {`picture`} and $\{m^r_{\mathcal{W}}\}$ = {`picture`, `sharing`}. The designer is supported in the formulation of the request by the same sense disambiguation techniques used during semantic tagging, as explained in Section 4.

3.4 Web APIs Matching and Ranking

The search scenarios introduced above can be satisfied by applying a set of metrics that are used to compare the affinity between categories, semantic tags and mashups in which the Web APIs must be included. The matching and ranking model we adopted in our system is defined by the following elements:

$$\Gamma = \langle \mathcal{W}^r, \{\mathcal{W}\}, \texttt{target}, Sim(), \rho \rangle \tag{4}$$

where $\mathcal{W}^r$ is the request, $\{\mathcal{W}\}$ is the set of semantic and social characterization of available Web APIs $\mathcal{W}$ in the repository ($\mathcal{W} = \langle \mathcal{W}_{Sem}, \mathcal{W}_{Soc} \rangle$), `target` is the kind of search (*simple*, *advanced*, *completion*, *proactive completion*), $Sim(\mathcal{W}^r, \mathcal{W})$ is the similarity measure used to evaluate candidate Web APIs as search results and ρ is the ranking function for search results. The **matching** measure $Sim(\mathcal{W}^r, \mathcal{W})$ is based on the semantic characterization $\mathcal{W}_{Sem}$ of $\mathcal{W}$ and is composed of the following elements.

Category similarity. The similarity between the category $c^r_{\mathcal{W}}$ of $\mathcal{W}^r$ and the category $c_{\mathcal{W}}$ of $\mathcal{W}_{Sem}$ is inferred from the ProgrammableWeb repository; since no hierarchies are defined among the available categories, advanced semantic-driven techniques (such as category subsumption checking) can not be used; nevertheless, we consider the two categories as more similar as the number of Web APIs that are categorized in both the categories, denoted with $|c^r_{\mathcal{W}} \cap c_{\mathcal{W}}|$, increases with respect to the overall number of Web APIs classified in $c^r_{\mathcal{W}}$,

denoted with $|c^r_{\mathcal{W}}|$, and in $c_{\mathcal{W}}$, denoted with $|c_{\mathcal{W}}|$; formally, the category similarity is defined as follows:

$$Sim_c(c^r_{\mathcal{W}}, c_{\mathcal{W}}) = \frac{2 \cdot |c^r_{\mathcal{W}} \cap c_{\mathcal{W}}|}{|c^r_{\mathcal{W}}| + |c_{\mathcal{W}}|} \tag{5}$$

Semantic tag affinity. Semantic tag affinity applied between two tags t_1 and t_2, denoted with $TagSim(t_1, t_2) \in [0, 1]$, is used to state how much similar they are with respect to the WordNet lexical system. In WordNet the synsets used to define the meaning of terms are related by eighteen different kinds of relationships. Some relationships have been designed to refine search capabilities and enhance the navigation of the net of terms in the lexical system. In particular, *hyponymy/hypernymy relations* are used to represent the specialization/generalization relationship between two terms: for instance, `station wagon` is a more specific term with respect to `automobile`; this means that there is a semantic affinity between `station wagon` and `automobile`, that is, if a user is looking for an automobile, also those resources that have been tagged with the station wagon term can be considered relevant. According to this viewpoint, we state that the affinity between two tags t_1 and t_2 is maximum if the tags belong to the same synset; otherwise, if they belong to different synsets, a path of hyponymy/hypernymy relations which connects the two synsets is searched: the highest the number of relationships in this path, the lowest is semantic tag affinity, that is:

$$TagSim(t_1, t_2) = \begin{cases} 1 & \textit{if } t_1 \textit{ and } t_2 \textit{ belong to the same synset} \\ 0.8^L & \textit{if there are } L \textit{ hyponymy/hypernymy relations between } t_1 \textit{ and } t_2 \\ 0 & \textit{otherwise} \end{cases} \tag{6}$$

The value 0.8 has been proved to be optimal in our experiments on WordNet terms affinity [5]. Nevertheless, it can be parameterized (within the range [0,1]) and set by the designer. The affinity between two semantic tags t_1 and t_2 is evaluated considering t^0_1 and t^0_2, while the list of synonyms and the human readable description in the t_1 and t_2 definitions are used to speed up $TagSim$ evaluation and to enable the identification of the synsets within WordNet. The tag affinity between two sets of semantic tags $\mathcal{T}_1$ and $\mathcal{T}_2$ is evaluated by computing the semantic tag affinity between each pairs of tags, one from $\mathcal{T}_1$ and one from $\mathcal{T}_2$ by applying the Dice formula [6]:

$$Sim_t(\mathcal{T}_1, \mathcal{T}_2) = \frac{2 \cdot \sum_{t_1 \in \mathcal{T}_1, t_2 \in \mathcal{T}_2} TagSim(t_1, t_2)}{|\mathcal{T}_1| + |\mathcal{T}_2|} \tag{7}$$

This is used to evaluate the total affinity $Sim_t(\{t^r_{\mathcal{W}}\}, \{t_{\mathcal{W}}\})$ between semantic tags used to annotate the Web APIs and $Sim_t(\{m^r_{\mathcal{W}}\}, \{m_{\mathcal{W}}\})$ between semantic tags used to annotate the mashups. Pairs to be considered for the Sim_t computation are selected according to a maximization function that relies on the assignment in bipartite graphs. For instance, there exists a path of length $L = 2$ between terms `picture` and `file` in WordNet, therefore $Sim_t(\{$`picture`, `sharing`$\}, \{$`file`, `sharing`$\})$ is evaluated as $[2 \cdot (0.8^2 + 1.0)]/4 = 0.82$.

The final matching measure $Sim(\mathcal{W}^r, \mathcal{W})$ is computed as:

$$Sim(\mathcal{W}^r, \mathcal{W}) = \omega_1 \cdot Sim_c(c^r_{\mathcal{W}}, c_{\mathcal{W}}) + \omega_2 \cdot Sim_t(\{t^r_{\mathcal{W}}\}, \{t_{\mathcal{W}}\}) + \omega_3 \cdot Sim_t(\{m^r_{\mathcal{W}}\}, \{m_{\mathcal{W}}\}) \in [0,1] \quad (8)$$

where $0 \leq \omega_i \leq 1$, with $i = 1, 2, 3$, and $\sum_{i=1}^{3} \omega_i = 1$ are weights set according to the `target`, as shown in Table 2. Setup experiments showed that the category is only a coarse-grained entry point to look for Web APIs in the repository (this explains the low values for ω_1 weight). For instance, the $Sim(\mathcal{W}^r, \texttt{Flickr})$ between the sample request $\mathcal{W}^r$ shown in Section 3.3 and the $Flickr_{Sem}$ characterization shown in Section 3.1 is computed as follows:

$$Sim(\mathcal{W}^r, \texttt{Flickr}) = 0.2 \cdot 1.0 + 0.4 \cdot 0.82 + 0.4 \cdot \frac{2 \cdot 1.0}{2} = 0.928 \quad (9)$$

Table 2. The setup of ω_i weights for computation of matching measure $Sim(\mathcal{W}^r, \mathcal{W})$

`Target`	**Request** $\mathcal{W}^r$	**Weights setup**
Simple	$\mathcal{W}^r_1 = \langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}\rangle$	$\omega_1 = 0.4$, $\omega_2 = 0.6$, $\omega_3 = 0.0$
Advanced	$\mathcal{W}^r_2 = \langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{m^r_{\mathcal{W}}\}\rangle$	$\omega_1 = 0.2$, $\omega_2 = \omega_3 = 0.4$
Completion	$\mathcal{W}^r_3 = \langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{m^r_{\mathcal{W}}\}, \{\mathcal{W}_h\}\rangle$	$\omega_1 = 0.2$, $\omega_2 = \omega_3 = 0.4$
Proactive completion	$\mathcal{W}^r_4 = \langle \{m^r_{\mathcal{W}}\}, \{\mathcal{W}_h\}\rangle$	$\omega_1 = \omega_2 = 0.0$, $\omega_3 = 1.0$

The Web APIs included in the search results (which we denote with $\{\mathcal{W}'\} \subseteq \{\mathcal{W}\}$) are those whose overall similarity is equal or greater than a threshold γ experimentally set. The Web APIs $\{\mathcal{W}'\}$ are ranked according to the social characterization of each $\mathcal{W}'$. In particular, the **ranking** function $\rho : \{\mathcal{W}'\} \mapsto [0,1]$ takes into account the past experiences of designers in using the $\mathcal{W}'$ Web API ($\rho_1(\mathcal{W}')$) and the ratings given by the designers to $\mathcal{W}'$ ($\rho_2(\mathcal{W}')$). Depending on the declared skills, past experiences and ratings of more expert designers are considered as more relevant for ranking search results. In particular, we define $\rho(\mathcal{W}') = \alpha \cdot \rho_1(\mathcal{W}') + \beta \cdot \rho_2(\mathcal{W}')$. In the performed preliminary experiments, the weights α and β are both set to 0.5 to give the same relevance to the two aspects.

The computation of $\rho_1(\mathcal{W}')$ is different if the request $\mathcal{W}^r$ contains the set $\{\mathcal{W}_h\}$ of the Web APIs already included in the mashup in which the Web API to search for should be used (completion or proactive completion search scenarios) or not (simple/advanced search scenarios). In the first case, let be $\{\mathcal{W}^i_k\}$ the Web APIs included by the i-th designer $d^i \in \mathcal{D}_{\mathcal{W}'}$ in the same mashup where he/she used $\mathcal{W}'$. We use the degree of overlapping between $\{\mathcal{W}_h\}$ and $\{\mathcal{W}^i_k\}$, to quantify the closeness of mashup in which $\mathcal{W}'$ has been used and the mashup where $\mathcal{W}'$ will be used, through the same rationale applied to category similarity in formula (5), that is:

$$Sim_m(\{\mathcal{W}_h\}, \{\mathcal{W}^i_k\}) = \frac{2 \cdot |\{\mathcal{W}_h\} \cap \{\mathcal{W}^i_k\}|}{|\{\mathcal{W}_h\}| + |\{\mathcal{W}^i_k\}|} \in [0,1] \quad (10)$$

where $|\cdot|$ denotes the number of Web APIs in the set and $|\{\mathcal{W}_h\} \cap \{\mathcal{W}_k^i\}|$ denotes the number of common Web APIs in the two sets. The computation of $\rho_1(\mathcal{W}')$ is then performed as follows:

$$\rho_1(\mathcal{W}') = 1 - \frac{\sum_i (1 - \sigma_i \cdot Sim_m(\{\mathcal{W}_h\}, \{\mathcal{W}_k^i\}))}{|\mathcal{D}_{\mathcal{W}'}|} \in [0, 1] \tag{11}$$

where σ_i is the declared skill of designer $d^i \in \mathcal{D}_{\mathcal{W}'}$. The formula above ensures that the past experiences of more expert designers have a higher impact on the $\rho_1(\mathcal{W}')$ computation. Intuitively, the closest the σ_i and $Sim_m(\{\mathcal{W}_h\}, \{\mathcal{W}_k^i\})$ values to 1 (maximum value) for all the designers d^i, the closest the second member in formula (11) to zero, that is, the ranking $\rho_1(\mathcal{W}')$ assumes the best value. For instance:

$$\rho_1(\texttt{Flickr}) = 1 - \frac{(1 - 1.0 \cdot \frac{2 \cdot 1.0}{3}) + (1 - 1.0 \cdot \frac{2 \cdot 1.0}{2})}{2} = 0.833 \tag{12}$$

The value $\rho_1(\texttt{23hq.com})$ is computed in the same way and is equal to 0.334. If we consider the simple or advanced search scenario, where $\{\mathcal{W}_h\} = \emptyset$, we simplify formula (11) by putting $Sim_m(\{\mathcal{W}_h\}, \{\mathcal{W}_k^i\})$ to 1, that is, $\rho_1(\mathcal{W}') = 1 - [\sum_i (1 - \sigma_i)]/|\mathcal{D}_{\mathcal{W}'}|$. The Web API $\mathcal{W}'$ is ranked better if all the designers who adopted it have high development skill.

The computation of $\rho_2(\mathcal{W}')$ follows the same rationale, considering in this case the rating μ_i given by the designer d^i to the Web API $\mathcal{W}'$, that is:

$$\rho_2(\mathcal{W}') = 1 - \frac{\sum_i (1 - \sigma_i \cdot \mu_i)}{|\mathcal{D}_{\mathcal{W}'}|} \in [0, 1] \tag{13}$$

For instance, $\rho_2(\texttt{Flickr}) = 1 - [(1 - 0.7) + (1 - 0.6)]/2 = 0.65$.

4 The System Implementation

A designer can access our system using the *search facilities* to find out relevant Web APIs according to the search scenarios introduced above. Alternatively, he/she can *register himself/herself* in the system and he/she can take actively part in the semantic and social characterization of Web APIs he/she used. The architectural overview of our system is shown in Figure 2. The *Web API Semantic Tagging* and *Search interfaces* are PHP pages accessed through the Web browser and interact with: (i) the *ProgrammableWeb APIs* for retrieving basic information on Web APIs and Web mashups from the repository[3]; (ii) the *Web API Storage module*, where semantic and social characterization of Web APIs are stored together with designers' development skills and ratings and matching and ranking routines are implemented; (iii) the WordNet lexical system for sense disambiguation. We rely on a WordNet version made available for J2EE

[3] `api.programmableweb.com/`.

platform, therefore we implemented the sense disambiguation module as a Web service to enable communication between PHP pages and Java classes. Interactions with the other modules are managed with the AJAX technology to ensure good performances of the Web interfaces.

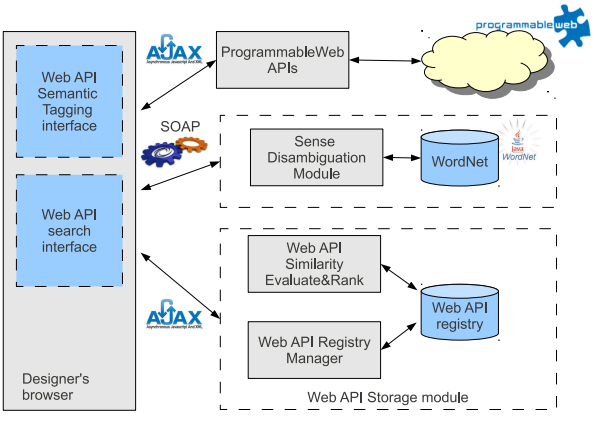

Fig. 2. Architecture of the Web API semantic collaborative tagging system

4.1 The Web API Semantic Collaborative Tagging

The *Web API Semantic Tagging interface* is implemented to manage the interaction of the designer during the semantic and social characterization of Web APIs after registration and authentication. Designer's registration is required to setup the development skill, authentication enables the system to associate the characterization of the Web API with the designer's profile. The designer is guided in a set of steps in the semantic and social characterization of the Web API (see Figure 3). On the top, the details of the Web API extracted from ProgrammableWeb are shown, together with the category which the Web API belongs to. On the bottom, four tabs which correspond to the four steps of the Web API semantic and social characterization are shown: the specification of semantic tags on the Web API; the optional specification of semantic tags on the Web mashup which the Web API has been used in; the optional specification of other Web APIs, among the ones registered in ProgrammableWeb, which have been used together with the current one in the Web mashup; the rating of the Web API, mapped into the numeric scores as shown in Table 1. In particular, the figure shows the first step. A text field is provided to enter the tag. As the designer inputs the characters of the term he/she wants to specify for tagging, the system provides an automatic completion mechanism based on the set of terms contained in WordNet. Starting from the tag specified by the designer, the Sense Disambiguation Module queries WordNet and retrieves all the synsets that contain that term and shows the semantic tags list.

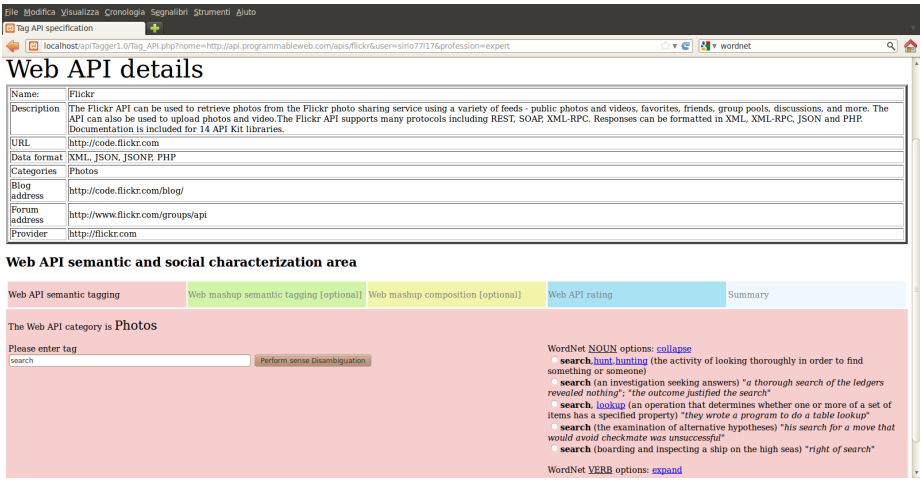

Fig. 3. The Web API Semantic Tagging interface

4.2 The Web API Search Interface

The *Web API Search interface* presents the same steps described for the Web API Semantic Tagging interface (except for the rating step) to specify the request $\mathcal{W}^r$. The designer can be either authenticated in the system or not, to execute a semantic tagging-based search. The designer's authentication could be explored to provide targeted search results, adapted to the skill and past Web mashups developed by the designer. This functionality has not been developed yet, but we plan to introduce it into future versions of the system. Different search modalities (*basic*, *advanced*, *completion*, *proactive completion*) can be chosen by the designer to enable/disable the tabs for specifying the $\mathcal{W}^r$ elements.

5 Related Work and Evaluation Issues

Some authors proposed the use of models to define Web APIs and the way they are composed in a Web mashup to empower their reuse and sharing. In [7] an abstract component model and a composition model are proposed, expressed by means of an XML-based language, for the construction of dashboards. In particular, components abstract the descriptions of enterprise internal services, public APIs and Web resources from technical details. Other efforts based their recommendations upon models. In [8] the formal model based on Datalog rules defined in [1] is proposed to capture all the aspects of a mashup component (called *mashlet*). In this model authors also consider the mashups that have been implemented using the Web API which is being modeled, but do not rely on other social aspects such as ratings and designers' expertise: when the designer

selects a mashlet, the system suggests other mashlets to be connected on the basis of recurrent patterns of components in the existing mashups. In [3] semantic annotations have been proposed to enrich Web API modeling in presence of high heterogeneity and proper metrics based on such annotations have been defined to improve recommendations on Web API retrieval and aggregation. This model has been extended in [9] with traditional API tagging to exploit *collective knowledge* for Web API recommendation, but also in this case ratings and designers' expertise are not taken into account. Although such models enable more precise metrics for Web API retrieval and the (semi)automatic generation of the glue code for deploying the final mashup, their use is not always feasible in a social context, where Web designers' expertise is mainly focused on Web programming technologies, and the ever growing addition of new Web APIs which present an high heterogeneity hampers the definition of proper wrappers to load Web API descriptions inside the model itself. In such a context, the adoption of Datalog rules to describe Web APIs and mashups [1] or of XML-based, abstract models for Web API description and composition [3,7] should be further investigated. In [10], a faceted classification of unstructured Web APIs and a ranking algorithm to improve their retrieval are proposed. The classification and searching solutions are based on IR techniques. The proposal provides a coarse-grained discovery mechanism and adopts components descriptions as published on ProgrammableWeb, without any further semantic characterization of components. In this paper, we also rely on the information stored on ProgrammableWeb, but we extend Web API descriptions with additional facets based on semantic tagging and on the past experiences of designers who adopted the Web APIs in their own mashups. With respect to approaches on semantic tagging of Web pages [11] and social search engines, such as Yahoo My Web 2.0, the semantic and social characterization of Web APIs on top of ProgrammableWeb must take into account not only each single resource (i.e., Web API) to be recommended, but also the way they are wired together within Web applications or mashups, thus raising additional aspects also related to the Web 3.0 context.

Preliminary evaluation. We performed an initial evaluation on the precision of the semantic tagging system in retrieving relevant Web APIs and on the ranking procedure. We focused on the application domain of the running example: we considered a subset of 395 Web APIs grouped in the `Entertainment`, `File Sharing`, `Mapping` and `Photos` categories of ProgrammableWeb repository; we collected a subset of mashups from the same repository, among the ones built with the selected Web APIs, and the corresponding developers (for example, the `Flickr` Web API has been used in about 602 mashups owned by 302 developers, while `23hq.com` has been used by 4 developers in 8 mashups); we performed semantic tagging starting from the keywords extracted from the Web APIs and Web mashups descriptions; finally, we classified developers' skills on the basis of the number of mashups and APIs they own. After semantic and social characterization, we performed four different kinds of search, corresponding to the four search scenarios. We manually built twelve requests $\mathcal{W}^r$ like the sample one

Table 3. Search results collected during the preliminary evaluation of the system (PW = ProgrammableWeb)

Search tags	# of retrieved APIs	Precision	Recall	Relevant APIs in the first 20 results
`photo`, `sharing`	83 APIs (233 on PW)	85% (33% on PW)	79% (40% on PW)	15 (9 on PW)
`picture`, `sharing`	83 APIs (30 on PW)	85% (22.5% on PW)	79% (19% on PW)	15 (6 on PW)

shown in Section 3.3 and we manually classified relevant Web APIs by carefully analyzing Web API descriptions in the ProgrammableWeb repository and the mashups where Web APIs have been used. We performed search experiments on our systems and directly on ProgrammableWeb, using the first elements in $\{t^r_{\mathcal{W}}\}$ and $\{m^r_{\mathcal{W}}\}$ as normal keywords without considering synonyms and human-readable descriptions. The obtained results are like the ones presented in Table 3, where we showed the output for the request $\mathcal{W}^r$ used in Section 3.3.

We evaluated the precision (number of relevant results among the retrieved ones) and the recall (number of relevant results that are retrieved). Experiments showed as our system presents better search results and, in particular, presents relevant results in the first positions of the search outcome. Increased precision is due to the adoption of sense disambiguation techniques, that enable to discard not relevant Web APIs, while increased recall is obtained thanks to the inclusion among search results of those Web APIs that have been annotated with tags that are synonyms of the ones specified during search. For example, it is interesting to underline that, if we change the search tag `photo` with `picture`, the performances of our system do not change due to the exploitation of synsets, while ProgrammableWeb returns worse search results (e.g., it does not return `Flickr` Web API). More in-depth tests with designers will be performed in future work, after a long-term interactions of designers with the system.

6 Conclusions

In this paper we described the functional architecture of a Web API semantic tagging system to be deployed on top of the public ProgrammableWeb API repository. The system is designed to be used in a social context: the designers can take actively part into the semantic tagging of Web APIs, thus sharing their experience in developing their own Web applications. Moreover, they can exploit the search facilities of the system to find out relevant Web APIs according to different search scenarios and reuse them for fast deployment of new applications. To this aim, they rely in an hybrid fashion on the semantic tags and on the collective knowledge derived from past designers' experiences. The proposed system does not require the application of complex models for Web API description, while ensuring extensibility, for example by adding further features beyond the semantic ones (such as the one defined in [10]) or additional sources for sense disambiguation (such as DBPedia), thus improving the precision and recall of

the searching procedure. The final product will be a Web API and Web mashup sharing and reuse system, built on top of common social applications, in Web 2.0 and Web 3.0 contexts.

References

1. Abiteboul, S., Greenshpan, O., Milo, T.: Modeling the Mashup Space. In: Proc. of the Workshop on Web Information and Data Management, pp. 87–94 (2008)
2. Bislimovska, B., Bozzon, A., Brambilla, M., Fraternali, P.: Graph-Based Search over Web Application Model Repositories. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 90–104. Springer, Heidelberg (2011)
3. Bianchini, D., De Antonellis, V., Melchiori, M.: Semantics-Enabled Web API Organization and Recommendation. In: De Troyer, O., Bauzer Medeiros, C., Billen, R., Hallot, P., Simitsis, A., Van Mingroot, H. (eds.) ER Workshops 2011. LNCS, vol. 6999, pp. 34–43. Springer, Heidelberg (2011)
4. Fellbaum, C.: Wordnet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
5. Bianchini, D., De Antonellis, V., Melchiori, M.: Flexible Semantic-based Service Matchmaking and Discovery. World Wide Web Journal 11(2), 227–251 (2008)
6. van Rijsbergen, C.J.: Information Retrieval. Butterworth (1979)
7. Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D., Francalanci, C.: DashMash: A Mashup Environment for End User Development. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 152–166. Springer, Heidelberg (2011)
8. Greenshpan, O., Milo, T., Polyzotis, N.: Autocompletion for Mashups. In: Proc. of the 35th Int. Conference on Very Large DataBases (VLDB 2009), Lyon, France, pp. 538–549 (2009)
9. Melchiori, M.: Hybrid techniques for Web APIs recommendation. In: Proceedings of the 1st International Workshop on Linked Web Data Management, pp. 17–23 (2011)
10. Gomadam, K., Ranabahu, A., Nagarajan, M., Sheth, A., Verma, K.: A Faceted Classification Based Approach to Search and Rank Web APIs. In: Proc. of International Conference on Web Services (ICWS 2008), Beijing, China, pp. 177–184 (2008)
11. Marchetti, A., Tesconi, M., Ronzano, F., Rosella, M., Minutoli, S.: SemKey: A Semantic Collaborative Tagging System. In: Proc. of WWW 2007 Workshop on Tagging and Metadata for Social Information Organization, Banff, Canada (2007)