A Multi-perspective Framework for Web API Search in Enterprise Mashup Design

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. Enterprise mashups are agile applications which combine enterprise resources with other external applications or web services, by selecting and aggregating Web APIs provided by third parties. In this paper, we provide a framework based on different Web API features to support Web API search and reuse in enterprise mashup design. The framework operates according to a novel perspective, focused on the experience of web designers, who used the Web APIs to develop enterprise mashups. This new perspective is used jointly with other Web API search techniques, relying on classification features, like categories and tags, and technical features, like the Web API protocols and data formats. This enables designers, who as humans learn by examples, to exploit the collective knowledge which is based on past experiences of other designers to find the right Web APIs for a target application. We also present a preliminary evaluation of the framework.

Keywords: Multi-perspective mashup model, Web API search and ranking, enterprise mashup, collective knowledge.

1 Introduction

The widespread adoption of mashup as a new development style for quick-to-build applications has enabled its introduction also within enterprises. Enterprise mashups are usually developed by selecting and aggregating domain-specific Web APIs, provided internally to the enterprise, and general-purpose Web APIs, provided by third parties. Web APIs are used to access underlying resources, functionalities or data, through web-based user interfaces [2]. Effective Web API search should be based not only on Web API features, such as categories, tags, protocols and data formats. Advanced methods and techniques should further enable web designers, both internal and external to the enterprise, to exploit experience of other designers in developing web applications, starting from the same repositories of Web APIs.

As a motivating example, consider Josephine, a designer who aims at building a new web application for her enterprise. The application should provide access to details about products advertised on the network through an enterprise e-commerce portal and information about customers who bought the products,

C. Salinesi, M.C. Norrie, and O. Pastor (Eds.): CAiSE 2013, LNCS 7908, pp. 353–368, 2013.
© Springer-Verlag Berlin Heidelberg 2013

such as shipping data, purchase history and contact information. Moreover, the web application could provide additional facilities to display on a map customers' addresses to better schedule shipping activity. The Josephine's problem is to decide if the functionalities of the new application, such as e-commerce facilities, must be implemented from scratch or if she may rely on available Web APIs. For instance, Josephine may use the well-known e-commerce facilities of the `Amazon` API, thus making the new application as more interoperable as possible by using a widely adopted component. Similarly, she may adopt `Salesforce.com`, for customer relationship management, or `Google Maps`. Josephine may be supported in her task by searching for Web APIs not only according to their categories, tags or more technical features, such as the protocols and data formats. Web APIs could be suggested to Josephine on the basis of past choices made by her colleagues or other web designers while developing similar applications or using similar Web APIs.

To support Web API search according to the vision highlighted above, we propose a framework based on a novel enterprise mashup model. The model reflects three different perspectives, namely *component* (Web APIs), *application* (enterprise mashups built with Web APIs) and *experience perspective* (web designers, who used Web APIs to develop enterprise mashups). We exploit the multi-perspective framework for Web API search and ranking, according to different search scenarios, namely the development of a new enterprise mashup and the enlargement of functionalities or the substitution of Web APIs in an existing mashup.

The paper is organized as follows. Section 2 describes the multi-perspective mashup model. In Section 3 we describe the search scenarios and the functional architecture of the proposed framework. The Web API search and ranking techniques are described in Section 4. In Section 5, we discuss about implementation issues and preliminary evaluation. Section 7 presents a comparison with the state of the art. Finally, Section 8 closes the paper.

2 Multi-perspective Mashup Model

The model we propose in this paper is designed according to three interconnected perspectives as shown in Figure 1 and detailed in the following. Each perspective is focused on specific elements, namely Web APIs, enterprise mashups and web designers, further described with proper features and relationships with elements of the other perspectives.

The Component Perspective. According to this perspective, Web APIs descriptive features are distinguished between classification features and technical features. These features have been chosen to be compliant with the descriptions of Web APIs indexed within the `ProgrammableWeb` repository, that is the most populated and better updated Web API public registry[1]. Classification features

[1] `http://www.programmableweb.com`.

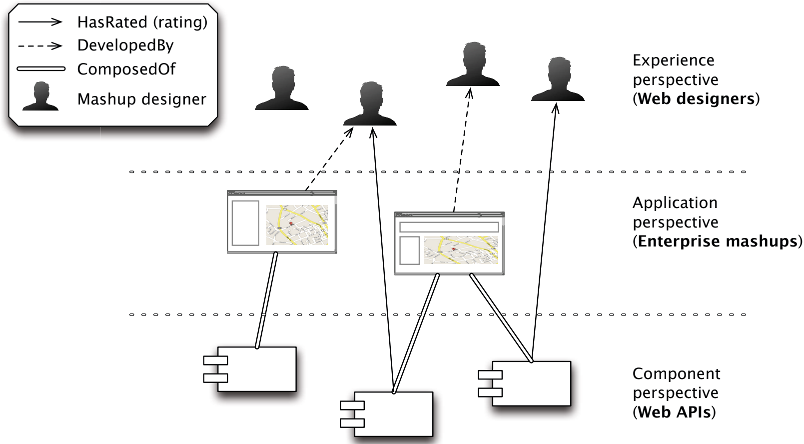

Fig. 1. The multi-perspective mashup model

are categories and semantic tags. In our model, a Web API $\mathcal{W}$ is assigned to a category $c_{\mathcal{W}}$ among the 67 `ProgrammableWeb` categories, such as `mapping`, `search` or `payment`. Semantic tags are adopted to face polisemy (that is, the same tag refers to different concepts) and homonymy problems (i.e., the same concept is pointed out using different tags), which traditional tagging may present. During the assignment of such tags, sense disambiguation techniques based on the WordNet 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. Starting from the tag specified by the designer, WordNet is queried, all the synsets that contain that term are retrieved, thus enabling the designer to select the intended meaning. In our model, each semantic tag $t_{\mathcal{W}}$, associated with the Web API $\mathcal{W}$, is a triplet, composed of: (i) the term itself extracted from WordNet; (ii) the set of all the terms in the same synset; (iii) the human readable definition associated with the synset. The semantic tagging procedure has been extensively described in [4]. The `ProgrammableWeb` repository also enables to retrieve additional Web API information, namely the technical features listed in Table 1. Sample descriptions of the `Amazon` e-commerce and `eBay` APIs using classification and technical features are the following:

`Amazon` = [c_{Amazon}: `Shopping`;
t^1_{Amazon}: ⟨`e-commerce`, {}, "commerce conducted electronically (as on the internet)"⟩;
$\mathcal{P}_{Amazon}$: {`REST`, `SOAP`}; $\mathcal{F}_{Amazon}$: {`XML`}; SSL$_{Amazon}$: `no`; $\mathcal{A}_{Amazon}$: {`APIKey`};
τ_{Amazon}: 2006-04-04; $\mathcal{H}_{Amazon}$: 17]

`eBay` = [c_{eBay}: `Search`;
t^1_{eBay}: ⟨`e-commerce`, {}, "commerce conducted electronically (as on the internet)"⟩;
$\mathcal{P}_{eBay}$: {`REST`, `SOAP`, `HTTP`}; $\mathcal{F}_{eBay}$: {`JSON`, `CVS`, `RDF`, `RSS`}; SSL$_{eBay}$: `yes`; $\mathcal{A}_{eBay}$: {`APIKey`};
τ_{eBay}: 2005-12-05; $\mathcal{H}_{eBay}$: 11]

Table 1. Technical features of a Web API $\mathcal{W}$

Feature	Description
Protocols $\mathcal{P}_{\mathcal{W}}$	A set of protocols adopted by the Web API (e.g., REST, SOAP, XML-RPC)
Data formats $\mathcal{F}_{\mathcal{W}}$	A set of formats adopted by the Web API for data exchange (e.g., XML, JSON)
$SSL_{\mathcal{W}}$	A flag that is set to `true` if the Web API provides SSL support for security purposes
$\mathcal{A}_{\mathcal{W}}$	The authentication mechanism implemented within the Web API, to be chosen among *no authentication*, *API key*, *developer key* and *user account* [6]
API update time $\tau_{\mathcal{W}}$	The date of the last update of the Web API
$\mathcal{H}_{\mathcal{W}}$	The number of "howTo" documents about the Web API

The Application Perspective. According to this perspective, an enterprise mashup M_k is composed of a set $\{\mathcal{W}_M\}$ of Web APIs and is described through a set $\{t_M\}$ of semantic tags, that are defined in the same way as Web API semantic tags. No technical features are provided for mashups, since these features are directly related to the component Web APIs. For instance, the following two mashups have been implemented using the `Salesforce.com` API together with other APIs, such as `Amazon` e-commerce and `eBay` APIs:

$M_{Channel}$ = [$t^1_{Channel}$: ⟨`marketing`, {`selling`,`merchandising`}, "the exchange of goods for an agreed sum of money"⟩;
{$\mathcal{W}_{Channel}$} = {`ChannelAdvisor`, `eBay`, `Salesforce.com`}]

$M_{eCommazon}$ = [$t^1_{eCommazon}$: ⟨`retail`, {}, "the selling of goods to consumers; usually in small quantities and not for resale"⟩;
$t^2_{eCommazon}$: ⟨`e-commerce`, {}, "commerce conducted electronically (as on the internet)"⟩;
{$\mathcal{W}_{eCommazon}$} = {`Amazon`, `Salesforce.com`, `UPS`}]

The Experience Perspective. This perspective models the set $\mathcal{D}_{\mathcal{W}}$ of designers, both internal and external to the enterprise, who used the Web API $\mathcal{W}$ to develop their own mashups. Each designer $d_i \in \mathcal{D}_{\mathcal{W}}$ is modeled through: (a) the skill σ_i for developing web applications; (b) a set of triplets $\langle \mathcal{W}_j, M_k, \mu_{jk} \rangle$ to denote that the designer assigned a quantitative rating μ_{jk} to the Web API $\mathcal{W}_j$ when used within the mashup M_k. The skill is asked to the web designer during the registration to the system, according to a discrete scale: 1.0 for `expert`, 0.8 for `high confidence`, 0.5 for `medium confidence`, 0.3 for `low confidence` and 0.0 for `unexperienced`.

The value of the quantitative rating μ_{jk} is selected by the designer according to the NHLBI 9-point Scoring System, whose adoption for Web API rating has been described in [4]. 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, the designer is provided with the set of options that are mapped into

the $[0, 1]$ range. In the following example, the external designer d_1, who is a medium-skilled enterprise mashup developer, used `eBay` API in the $M_{Channel}$ mashup and rated the Web API as excellent, while the internal designer d_2, who is an expert developer, used the `Amazon` API in the $M_{eCommazon}$ mashup and rated the API as very good.

$d_1 = \langle \texttt{external}, 0.5\ (\texttt{medium}), \{\langle \texttt{eBay}, M_{Channel}, 0.8\ (\texttt{excellent})\rangle\}\rangle$
$d_2 = \langle \texttt{internal}, 1\ (\texttt{expert}), \{\langle \texttt{Amazon}, M_{eCommazon}, 0.7\ (\texttt{very good})\rangle\}\rangle$

3 The Proposed Architecture

The metrics based on the proposed model have been implemented in the system we called *APITagger*. The functional architecture of APITagger is shown in Figure 2.

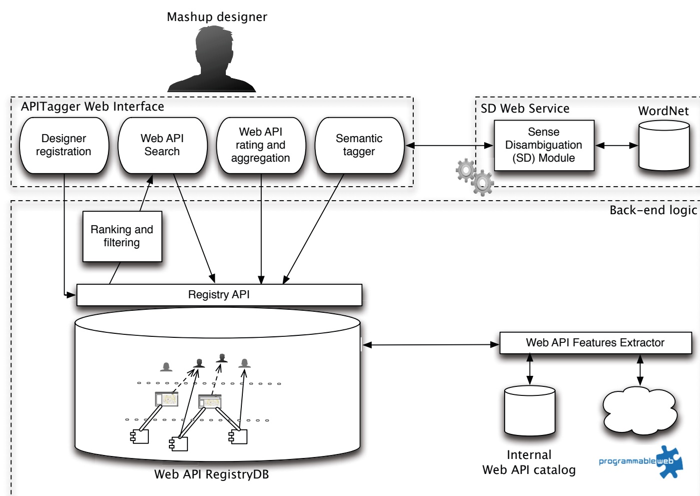

Fig. 2. The functional architecture of the APITagger system, which implements the multi-perspective framework

The *APITagger Web Interface* guides the web designer through the registration process, the Web API search and rating, the Web API and enterprise mashup semantic tagging. Designers' registration is required to setup the development skill and other features according to the experience perspective. Semantic tagging is supported by a *Sense Disambiguation (SD) module* that implements the sense disambiguation facilities introduced in the previous section. The module is implemented as a Web service and is invoked by other modules of the architecture.

Web API categories and technical features are obtained from a catalog internal to the enterprise or from `ProgrammableWeb` public registry through proper

methods (`api.programmable.com`), invoked within the *Web API Features Extractor*. *Ranking and filtering* module implements the metrics described in this paper (see Section 4). The designer is supported throughout the formulation of the request (*Web API search*), according to different *search scenarios*, which are defined in the next section. The request is issued as a query on the *Web API RegistryDB* and query results are ranked, filtered and presented to the designer, who may rate them and/or specify their joined use in her own enterprise mashups (*Web API rating and aggregation*).

3.1 Search Scenarios

We classify different search scenarios which the web designer may be acting in according to two dimensions: the *search target* and the *search typology*.

The **search target** identifies the goal of the Web API search, namely the selection of a single Web API, that we denote with *single selection target* (for instance, when the designer starts the design of a new mashup), the completion of an existing mashup by adding new Web APIs (*completion target*) or the substitution of a Web API in an existing mashup (*substitution target*).

The **search typology** identifies the way Web API search is performed. We distinguish between *simple search*, *advanced search* and *proactive search*. In the simple search, the web designer is looking for a Web API by specifying a category and a set of semantic tags. A variant of this search, denoted with *advanced search*, is the one where the designer has also in mind the mashup where the Web API to search for should be used. The mashup is specified through a set of semantic tags and, optionally, a set of Web APIs already included in the mashup where the new Web API will be inserted. In the proactive search, the designer does not specify the Web API to search for; she starts from a mashup, specified through a set of semantic tags and a set of Web APIs already included in the mashup, and she relies on the system that proactively suggests which Web APIs could be added given similar mashups developed in the past. For instance, in the running example, the system could proactively suggest to Josephine to include a chat API since this kind of functionality is often adopted within mashups which contain both `Salesforce.com` and `Amazon` e-commerce APIs.

A generic Web API request is defined as a 4-tuple:

$$\mathcal{W}^r = \langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{t^r_M\}, \{\mathcal{W}^r_M\}\rangle \tag{1}$$

where $c^r_{\mathcal{W}}$ is the requested Web API category, $\{t^r_{\mathcal{W}}\}$ is a set of semantic tags specified for the Web API to search for, $\{t^r_M\}$ is a set of semantic tags featuring the mashup M where the Web API to search for should be used, $\{\mathcal{W}^r_M\}$ is the set of Web APIs already included in the mashup M.

The definition of the request $\mathcal{W}^r$ is specialized depending on the search scenario as shown in Table 2. In the simple search, the requested Web API category $c^r_{\mathcal{W}}$ and semantic tags $\{t^r_{\mathcal{W}}\}$ are specified, while in the advanced search also $\{t^r_M\}$ are used to describe the mashup that is being developed and where the Web API to search for should be included. When simple or advanced search are used for

Table 2. The specification of the request $\mathcal{W}^r$ depending on the search scenario

	search typology		
search target	**simple search**	**advanced search**	**proactive search**
single selection	$\langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}\rangle$	$\langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{t^r_M\}\rangle$	*n.a.*
completion	$\langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{\mathcal{W}^r_M\}\rangle$	$\langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{t^r_M\}, \{\mathcal{W}^r_M\}\rangle$	$\langle \{t^r_M\}, \{\mathcal{W}^r_M\}\rangle$
substitution	$\langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{t^r_M\}, \{\mathcal{W}^r_M\}\rangle$	$\langle c^r_{\mathcal{W}}, \{t^r_{\mathcal{W}}\}, \{t^r_M\}, \{\mathcal{W}^r_M\}\rangle$	*n.a.*

completion or substitution purposes, also the set $\{\mathcal{W}^r_M\}$ of Web APIs already included in the mashup to be completed or modified is specified in the request. Note that in the substitution target, $c^r_{\mathcal{W}}$ and $\{t^r_{\mathcal{W}}\}$ are automatically extracted from the category and semantic tags of the Web API selected by the designer for substitution. Proactive search is applicable only for completion purposes. In fact, in the substitution target the Web API to substitute is known (through its $c^r_{\mathcal{W}}$ and $\{t^r_{\mathcal{W}}\}$), that is, no proactivity is involved. In the single selection target, there is no mashup under development yet and the system has no information which proactive suggestion could be based on.

For example, a request $\mathcal{W}^r$, formulated according to the *advanced search* and *completion target*, to find an e-commerce Web API in the category `Shopping`, to be used in a mashup specified through the tag `selling` and which already includes `Salesforce.com` and `UPS` APIs, can be represented as follows:

$\mathcal{W}^r = \langle c^r_{\mathcal{W}}$: `Shopping`;
$\{t^r_{\mathcal{W}}\} = \{\langle$`e-commerce`, $\{\}$, "commerce conducted electronically (as on the internet)"$\rangle\}$;
$\{t^r_M\} = \{\langle$`selling`, $\{$`marketing`, `merchandising`$\}$, "the exchange of goods for an agreed sum of money"$\rangle$,
$\langle$`retail`, $\{\}$, "the selling of goods to consumers; usually in small quantities and not for resale"$\rangle\}$;
$\{\mathcal{W}^r_M\} = \{$`Salesforce.com`, `UPS`$\}$

4 Web API Search and Ranking

4.1 Web API Search

Web API search and ranking have been implemented through a set of metrics, which compare the elements of the request $\mathcal{W}^r$ with the features of each API $\mathcal{W}$ indexed in the Web API RegistryDB, according to the three perspectives described in Section 2. Search and ranking metrics can be properly set up depending on the search scenarios. For Web API search purposes, the most a Web API $\mathcal{W}$ fits the request $\mathcal{W}^r$, the most their categories, their semantic tags and the mashups which contain them are similar. The building blocks are the *category similarity*, the *semantic tag similarity* and the *mashup composition similarity* metrics.

The **category similarity** between the category $c^r_{\mathcal{W}}$ of $\mathcal{W}^r$ and the category $c_{\mathcal{W}}$ of $\mathcal{W}$ can not be inferred using advanced semantic-driven techniques (such as category subsumption checking), since no hierarchies are defined among the available categories in the `ProgrammableWeb` categorization we chose for our model. 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:

$$Sim_{cat}(c^r_{\mathcal{W}}, c_{\mathcal{W}}) = \frac{2 \cdot |c^r_{\mathcal{W}} \cap c_{\mathcal{W}}|}{|c^r_{\mathcal{W}}| + |c_{\mathcal{W}}|} \in [0, 1] \quad (2)$$

The **semantic tag similarity** between two sets of semantic tags, denoted with $Sim_{tag}() \in [0, 1]$, is computed by evaluating the term affinity between pairs of tags, one from the first set and one from the second set, and by combining them through the Dice formula. The term affinity between two tags t_1 and t_2 belongs to the range $[0, 1]$ and it is computed as extensively described in [4], based on WordNet. In WordNet, synsets are related by *hyponymy/hypernymy relations*, used to represent the specialization/generalization relationship between two terms. Term affinity is equal to 1.0 if the two tags belong to the same synset or coincide; it decreases as long as the path of hyponymy/hypernymy relations between the two synsets of the tags increases. In particular, term affinity is equal to 0.8^L, where there is a path of L hyponymy/hypernymy relations between the two terms. The value 0.8 has been proven to be optimal in our experiments on WordNet term affinity. Pairs of tags to be considered in the Sim_{tag} computation are selected according to a maximization function that relies on the assignment in bipartite graphs. This function ensures that each tag from the first set participates in at most one pair with one of the tags from the second set and viceversa and the pairs are selected in order to maximize the overall Sim_{tag}. Figure 3 shows a portion of WordNet vocabulary and a sample computation of Sim_{tag}.

The **mashup composition similarity** between a mashup composed of a set $\{\mathcal{W}^r_M\}$ of Web APIs and another mashup composed of a set $\{\mathcal{W}_k\}$ of Web APIs, denoted with $Sim_{comp}(\{\mathcal{W}^r_M\}, \{\mathcal{W}_k\})$, evaluates the number of common Web APIs in the two mashups. The corresponding formula is built by following the same rationale of $Sim_{cat}()$, that is:

$$Sim_{comp}(\{\mathcal{W}^r_M\}, \{\mathcal{W}_k\}) = \frac{2 \cdot |\{\mathcal{W}^r_M\} \cap \{\mathcal{W}_k\}|}{|\{\mathcal{W}^r_M\}| + |\{\mathcal{W}_k\}|} \in [0, 1] \quad (3)$$

where $|\cdot|$ denotes the number of Web APIs in the set and $|\{\mathcal{W}^r_M\} \cap \{\mathcal{W}_k\}|$ denotes the number of common Web APIs in the two sets. For instance, if $\{\mathcal{W}^r_M\} = \{$`Salesforce.com`, `UPS`$\}$ and $\{\mathcal{W}_k\} = \{$`Amazon`, `Salesforce.com`, `UPS`$\}$, therefore

$$Sim_{comp}(\{\mathcal{W}^r_M\}, \{\mathcal{W}_k\}) = \frac{2 \cdot 2}{3 + 2} = 0.8 \in [0, 1]$$

$Sim_{comp}(\{$`eBay`, `Salesforce.com`$\}, \{$`ChannelAdvisor`, `Salesforce.com`, `UPS`$\})$ $= 0.4$ are computed in the same way.

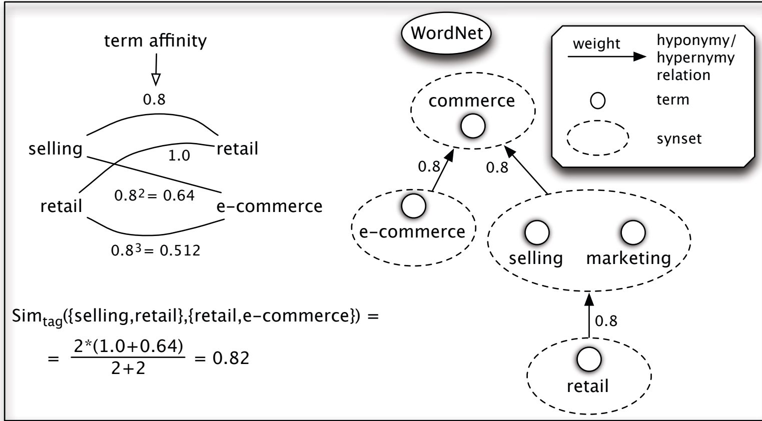

Fig. 3. An example of Sim_{tag} computation

Category, semantic tag and mashup composition similarity metrics are used to compute: (a) the similarity between the request $\mathcal{W}^r$ and an API $\mathcal{W}$ according to the features of the component perspective, denoted with $APISim()$; (b) the similarity between $\mathcal{W}^r$ and $\mathcal{W}$ in the context of mashups where $\mathcal{W}$ has been used (application perspective), denoted with $MashupSim()$. The similarity $APISim(\mathcal{W}^r, \mathcal{W}) \in [0, 1]$ is defined as:

$$APISim(\mathcal{W}^r, \mathcal{W}) = \omega_1 \cdot Sim_{cat}(c^r_{\mathcal{W}}, c_{\mathcal{W}}) + \omega_2 \cdot Sim_{tag}(\{t^r_{\mathcal{W}}\}, \{t_{\mathcal{W}}\}) \tag{4}$$

where $c_{\mathcal{W}}$ is the category of $\mathcal{W}$, $\{t_{\mathcal{W}}\}$ is the set of semantic tags associated with $\mathcal{W}$, $0 \leq \omega_1, \omega_2 \leq 1$ and $\omega_1 + \omega_2 = 1$. Inspection of the adopted `ProgrammableWeb` categorization showed that the category is only a coarse-grained entry point to look for Web APIs. According to this, the weights have been configured in our experiments as $\omega_1 = 0.2$ and $\omega_2 = 0.8$. An example of $APISim()$ computation is the following:

$$APISim(\mathcal{W}^r, \texttt{Amazon}) = 0.2 \cdot 1.0 + 0.8 \cdot \frac{2 \cdot 1.0}{2} = 1.0$$

Similarly, $APISim(\mathcal{W}^r, \texttt{eBay}) = 0.8$. Note that, in the last case, the $APISim()$ is high although $Sim_{cat}() = 0$, due to the proper setup of ω_1 and ω_2 weights.

The similarity between the request $\mathcal{W}^r$ and an API $\mathcal{W}$ in the context of an enterprise mashup M, denoted with $MashupSim(\mathcal{W}^r, \mathcal{W}, M) \in [0, 1]$, is defined as the following linear combination:

$$\begin{aligned} MashupSim(\mathcal{W}^r, \mathcal{W}, M) = \omega_3 \cdot Sim_{tag}(\{t^r_M\}, \{t_M\}) + \\ + \omega_4 \cdot Sim_{comp}(\{\mathcal{W}^r_M\}, \{\mathcal{W}_k\}) \end{aligned} \tag{5}$$

where $0 \leq \omega_3, \omega_4 \leq 1$, $\omega_3 + \omega_4 = 1.0$, M is composed of Web APIs in $\{\mathcal{W}_k\}$ and $\mathcal{W}$ belongs to M. To setup the weights in this case we have to take into account the search target. In fact, in the single selection search target, $\{\mathcal{W}^r_M\}$ is not specified

(see Table 2). Therefore, $\omega_3 = 1$ and $\omega_4 = 0$. In the other cases, ω_3 and ω_4 weights are both set to 0.5 to equally consider the two kinds of similarity. For instance:

$$MashupSim(\mathcal{W}^r, \texttt{Amazon}, M_{\texttt{eCommazon}}) = 0.5 \cdot \frac{2 \cdot 0.82}{3} + 0.5 \cdot 0.8 = 0.67$$

Similarly, $MashupSim(\mathcal{W}^r, \texttt{eBay}, M_{\texttt{Channel}}) = 0.53$. The overall matching measure between the request $\mathcal{W}^r$ and each API $\mathcal{W}$, denoted with $Sim(\mathcal{W}^r, \mathcal{W}) \in [0, 1]$, is computed as:

$$\begin{aligned} Sim(\mathcal{W}^r, \mathcal{W}) &= \omega_5 \cdot APISim(\mathcal{W}^r, \mathcal{W}) + \\ &+ (1 - \omega_5) \cdot (1 - \tfrac{\sum_i (1 - \sigma_i \cdot MashupSim(\mathcal{W}^r, \mathcal{W}, M_i))}{|\mathcal{D}_{\mathcal{W}}|}) \end{aligned} \quad (6)$$

where $\mathcal{W} \in M_i$. In particular, the second term in Equation (6) takes into account that the Web API $\mathcal{W}$ has been used in different mashups, by designers with different skills σ_i, where σ_i is the declared skill of designer $d_i \in \mathcal{D}_{\mathcal{W}}$. The second term in Equation (6) ensures that the past experiences of more expert designers have a higher impact on the $Sim()$ computation. Intuitively, the closest the σ_i and $MashupSim()$ values to 1 (maximum value) for all the designers d_i, the closest the second term in Equation (6) to 1.0. The weight ω_5 is set according to the search typology. If a simple search is being performed, the second term in Equation (6) must be ignored, that is, $\omega_5 = 1.0$; similarly, if we are performing a proactive search, $\omega_5 = 0$ to ignore the first term; otherwise, $\omega_5 = 0.5$ to equally weight the two terms in the $Sim()$ evaluation. For instance:

$$Sim(\mathcal{W}^r, \texttt{Amazon}) = 0.5 \cdot 1.0 + 0.5 \cdot (1 - \frac{(1 - 1.0 \cdot 0.67)}{1}) = 0.84$$

Similarly, $Sim(\mathcal{W}^r, \texttt{eBay}) = 0.53$. 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 $\gamma \in [0, 1]$ set by the web designer.

4.2 Web API Ranking

The Web APIs $\{\mathcal{W}'\}$, included among the search results, are ranked taking into account both the technical features in the component perspective and ranking of web designers who used the Web APIs to develop new mashups. In particular, the **ranking** function $\rho : \{\mathcal{W}'\} \mapsto [0, 1]$ is defined as a linear combination as follows:

$$\rho(\mathcal{W}') = \alpha \cdot \rho_1(\mathcal{W}') + \sum_{j=1}^{6} \gamma_j \cdot \widehat{\rho}_j(\mathcal{W}') \in [0, 1] \quad (7)$$

where $0 \leq \alpha, \gamma_j \leq 1$, $j = 1, ..., 6$, and $\alpha + \sum_{j=1}^{6} \gamma_j = 1$ are weights that are equally weighted because we consider all the terms in Equation (7) as equally relevant. Future work will be devoted to the setup of preferences among $\rho_1()$ and $\widehat{\rho}_j()$ functions.

The computation of $\rho_1(\mathcal{W}')$ follows the same rationale of the second term in Equation (6), that is, considers as more important ratings assigned by more expert designers:

$$\rho_1(\mathcal{W}') = 1 - \frac{\sum_i (1 - \frac{\sum_k \sigma_i \cdot \mu_{jk}}{|M_k|})}{|\mathcal{D}_{\mathcal{W}'}|} \in [0, 1] \quad (8)$$

where Equation (8) must consider the ratings μ_{jk} given by all the designers $d_i \in \mathcal{D}'_{\mathcal{W}}$ who used the Web API $\mathcal{W}'$ in mashups $\{M_k\}$. If the search has a completion or a substitution target, then rating must be further weighted by the mashup similarity, that is:

$$\rho_1^{ext}(\mathcal{W}') = 1 - \frac{\sum_i (1 - \frac{\sum_k \sigma_i \cdot \mu_{jk} \cdot MashupSim(\mathcal{W}^r, \mathcal{W}', M_k)}{|M_k|})}{|\mathcal{D}_{\mathcal{W}'}|} \in [0,1] \quad (9)$$

The computation of $\widehat{\rho}_j(\mathcal{W}')$ is based on the technical features in the component perspective (see Table 1). As for possible metrics to be adopted within $\widehat{\rho}_j(\mathcal{W}')$, a good survey can be found in [6]. For instance, $\widehat{\rho}_1(\mathcal{W}')$, that is associated with the set of protocols $\mathcal{P}_{\mathcal{W}'}$, can be based on the popularity of protocols $\mathcal{P}_{\mathcal{W}'}$ (that is, the number of APIs which use them); $\widehat{\rho}_5(\mathcal{W}')$, that is associated with the API update time $\tau_{\mathcal{W}'}$, is based on the delta of $\tau_{\mathcal{W}'}$ with respect to the current date. Nevertheless, among technical features, we distinguish those that may behave differently for Web APIs $\mathcal{W}'$ that must be placed within existing mashups, for completion or substitution purposes. Specifically, we consider the protocols in $\mathcal{P}_{\mathcal{W}'}$ and the data formats in $\mathcal{F}_{\mathcal{W}'}$. In these specific cases, the best solution is that all the Web APIs within the mashup share the same protocol and data format. Therefore, let be $\overline{\mathcal{P}}_{\mathcal{W}}$ and $\overline{\mathcal{F}}_{\mathcal{W}}$ the union set of protocols and data formats, respectively, used by Web APIs within an existing mashup, and $\mathcal{W}'$ an API to be added to such mashup for completion and substitution purposes. We propose a variant of $\widehat{\rho}_j(\mathcal{W}')$ functions, where $j = 1, 2$, denoted with $\overline{\rho_j}(\mathcal{W}')$, where $\overline{\rho_1}(\mathcal{W}') = 1.0$ if $\mathcal{P}_{\mathcal{W}'} \subseteq \overline{\mathcal{P}}_{\mathcal{W}}$, 0.0 otherwise ($\overline{\rho_2}(\mathcal{W}')$ is defined in the same way for $\overline{\mathcal{F}}_{\mathcal{W}}$). For instance, `eBay` API provides more data formats and protocols with respect to `Amazon`, but if we are going to search for a Web API to be included in a mashup where the XML data format is required, then `Amazon` should be the preferred choice. This example shows how the *search target* is used to select the ranking functions. Specifically, if the target is the *completion* of an existing mashup or the *substitution* of a Web API in an existing mashup, then the variants $\overline{\rho_1}(\mathcal{W}')$ and $\overline{\rho_2}(\mathcal{W}')$ are adopted in Equation (7). A further filtering could be performed if the Web API $\mathcal{W}'$ has been used by a designer, external or internal to the enterprise. For instance, a web designer may prefer to consider only those search results which have been used by other designers within the enterprise (e.g., to consider Web APIs which expose functionalities for the enterprise core business).

5 Implementation Issues

Figure 4 shows the APITagger search page for a logged web designer. Designer's registration is required to setup the development skill and to list designer's private information, including the list of her mashups in the `My MashUp` menu. The selected mashup is described in the middle of the search page, together with the details about mashup APIs. Here the designer may tag both the mashups and the Web APIs through the `tag` button.

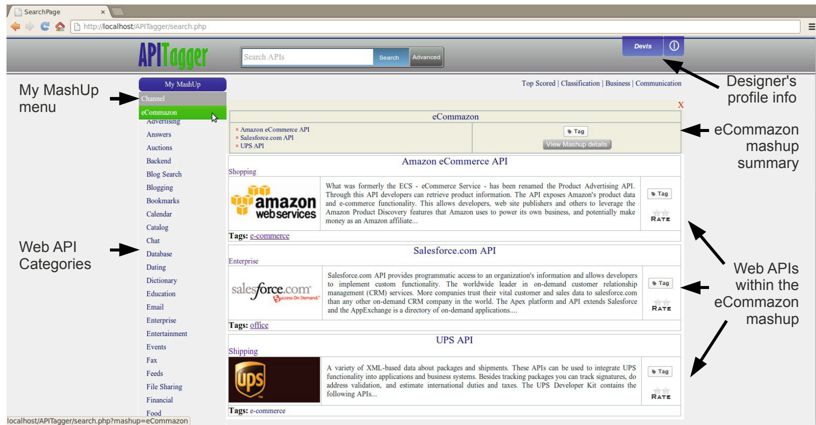

Fig. 4. The framework web interface

By pushing the button, a popup window is displayed, where a text field is provided to enter the tag. As the designer inputs the characters of the term 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 Web service queries WordNet and retrieves all the synsets that contain that term and shows the semantic tags list.

In the search page, the designer may also rate the Web APIs for their use in a given mashup, according to the NHLBI 9-point Scoring System: for example, in Figure 4 the designer may rate the `Amazon`, the `Salesforce.com` or the `UPS` API as used within the `eCommazon` mashup. The other elements of the search page, such as the list of categories on the left or the search field on the top, enable the designer to perform traditional Web API search, namely category-based or keyword-based. Finally, by pushing the `Advanced` button, the system guides the designer through the formulation of request $\mathcal{W}^r$ by means of a sequence of popup windows.

6 Preliminary Evaluation

Since there are no benchmarks to compare our implementation with similar efforts, we built our own dataset to perform a preliminary laboratory experiment on the framework. The experiment focuses on the application domain of the running example: we considered a subset of 922 Web APIs grouped in the `Advertising`, `Enterprise`, `Office`, `Shopping` and `Shipping` categories together with their technical features extracted from the `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 `Amazon` API has been used in about 416 mashups owned by 216 developers).

We performed semantic tagging starting from the keywords extracted from the Web API and mashup descriptions; finally, we classified developers' skills on the basis of the number of mashups and APIs they own. We merged this dataset with the one built in the same way in [4], obtaining a total of 1317 Web APIs and related mashups.

We ran two kinds of experiments on an Intel laptop, with 2.53 GHz Core 2 CPU, 2GB RAM and Linux OS. Experiments have been performed ten times using different requests. In the first experiment, we performed a single Web API search by specifying a category and a tag and issuing the request to four different systems: (a) the `ProgrammableWeb` search facilities; (b) an implementation of the system that relies on the classification and technical features in the component perspective only, inspired by ApiHut [10]; (c) the APITagger system implementation. Then we randomly selected 20 Web APIs, both included and not considered among the first 10 search results and we asked five expert users to classify the Web APIs as relevant and not relevant for the request issued. Finally, we compared the search results against the classification made by users, considering as relevant for the systems those Web APIs that have been listed among the first 10 search results. The comparison is based on the well known Cohen's kappa statistical measure. The results are shown in the first column in Table 3: as expected, all systems perform better than `ProgrammableWeb`, but APITagger outperforms all the systems, due to the extensive use of other designers' experiences both for searching and ranking.

In the second experiment, we randomly chose a mashup M and we extracted from the mashup a Web API $\mathcal{W}$. We then issued a request using the features of $\mathcal{W}$ given a mashup $M' = M/\{\mathcal{W}\}$ and we calculated the average position of $\mathcal{W}$ among search results given by our system. The results are shown in the second column of Table 3.

Table 3. Results collected during the preliminary evaluation of the system

Compared systems	**Cohen's kappa measure**	**Average ranking of $\mathcal{W}$ API**
`ProgrammableWeb`	0.43	6.9
Partial implementation based on the component perspective only	0.63	4.8
APITagger	0.78	2.1

7 Related Work

The approach described in [15] is the one closest to our, since authors consider also past experiences in building mashups to search for relevant Web APIs and for ranking the search results. As underlined in [15], approaches which rely only on a single perspective may suffer from the *cold start problem*, that is, a component that has been used in a significative number of cases gets more and more used despite the inner quality of the Web API, or may pass unnoticed because of its poor quality descriptions (as often happens when the designers

themselves are in charge of assigning categories or tags). These limitations apply to works which rely only on the popularity of Web APIs according to their use in existing mashups [8,11,16], or within approaches which only consider technical features, categories and tags for the classification and ranking of Web APIs, such as the one described in [10]. Nevertheless, despite its similarity with our multi-perspective framework, the approach described in [15] focuses on what we denoted as the component and application perspectives, without taking into account other aspects such as ratings and designers' expertise and without tuning search and ranking mechanisms according to different search scenarios. Other works [3,5,13] focus only on one of the perspectives we considered in this paper.

For what concerns the definition of models for Web API search and ranking, the ones which have been proposed in several papers focus on a subset of the features considered in our model. Technical features have been discussed in [6], where the authors includes them in a Web API model aimed at classifying the quality of resources accessed through the Web APIs (in terms of accuracy, completeness, timeliness and availability) and the quality of the Web API interface in terms of usability and accessibility. Component and application perspectives have been modeled in [1] and, in an enterprise context, in [12], without considering the experience perspective. In [4] a framework based on a Web API lightweight model is proposed. This model is compliant with the information extracted from the `ProgrammableWeb` repository and enriches such information by means of a collaborative semantic tagging system, where web designers can take actively part in the semantic tagging of Web APIs. In this paper, we perform several steps forward: i) we proposed a revision of the model by introducing three perspectives, focused on Web APIs, enterprise mashups built with Web APIs and web designers, who used Web APIs to develop enterprise mashups; ii) we extended the set of features to be considered for Web API search and ranking; iii) we tuned the multi-perspective framework according to different search scenarios, namely the development of a new enterprise mashup and the enlargement of functionalities or the substitution of Web APIs in an existing mashup. Specifically, the aim in this paper has been to rely on a lightweight API model that is compliant with existing Web API repositories as well. The need of adopting a lightweight model also affected the adoption of simplified matching techniques used for discovery purposes: the adoption of ontology-based techniques such as the ones described in [7,9] is unfeasible in this context, since they would require semantic annotation of Web API descriptions that is error-prone and time consuming. The evolution of our approach towards these kinds of metrics will be investigated as future work, after an integration of the APITagger framework with advanced techniques for extracting additional information from Web API documentation [14].

8 Conclusions

A framework that merges different Web API features, ranging from descriptive to social-based ones, is crucial for Web API sharing to enable the development

of quick-to-build applications starting from ready-to-use components. In this paper we proposed a multi-perspective framework, where a perspective focused on the experience of web designers is used jointly with other Web API search techniques, relying on classification features, like categories and tags, and technical features, like the Web API protocols and data formats. Future work will be devoted to enrich the model, both adding further social features (e.g., modeling the social network of web designers) and including features extracted from other Web API repositories (e.g., `mashape`). Future work will also concern the refinement of framework metrics, such as the implementation of different strategies to setup thresholds and weights (for instance, depending on the search scenario) or extending the sense disambiguation module with additional knowledge bases and lexical systems.

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. Beemer, B., Gregg, D.: Mashups: A Literature Review and Classification Framework. Future Internet 1, 59–87 (2009)
3. Bianchini, D., De Antonellis, V., Melchiori, M.: A lightweight model for publishing and sharing Linked Web APIs. In: Proceedings of the 20th Italian Symposium on Advanced Database Systems (SEBD 2012), pp. 75–82 (2012)
4. Bianchini, D., De Antonellis, V., Melchiori, M.: Semantic Collaborative Tagging for Web APIs Sharing and Reuse. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 76–90. Springer, Heidelberg (2012)
5. Bianchini, D., De Antonellis, V., Melchiori, M.: Towards semantic-assisted web mashup generation. In: Proceedings of International Workshop on Database and Expert Systems Applications, DEXA, pp. 279–283 (2012)
6. Cappiello, C., Daniel, F., Matera, M.: A Quality Model for Mashup Components. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 236–250. Springer, Heidelberg (2009)
7. Castano, S., Ferrara, A., Lorusso, D., Montanelli, S.: On the ontology instance matching problem. In: Proceedings - International Workshop on Database and Expert Systems Applications, DEXA, pp. 180–184 (2008)
8. Elmeleegy, H., Ivan, A., Akkiraju, R., Goodwin, R.: MashupAdvisor: A Recommendation Tool for Mashup Development. In: Proc. of 6th Int. Conference on Web Services (ICWS 2008), Beijin, China, pp. 337–344 (2008)
9. Ferrara, A., Lorusso, D., Montanelli, S., Varese, G.: Towards a benchmark for instance matching. In: CEUR Workshop Proceedings, pp. 37–48 (2008)
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), pp. 177–184 (2008)
11. 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)
12. Hoyer, V., Stanoevska-Slabeva, K.: Towards A Reference Model for Grassroots Enterprise Mashup Environments. In: 17th European Conference on Information Systems (2009)

13. Melchiori, M.: Hybrid techniques for Web APIs recommendation. In: Proceedings of the 1st International Workshop on Linked Web Data Management, pp. 17–23 (2011)
14. Rodríguez, R., Espinosa, R., Bianchini, D., Garrigós, I., Mazón, J.-N., Zubcoff, J.J.: Extracting Models from Web API Documentation. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 134–145. Springer, Heidelberg (2012)
15. Tapia, B., Torres, R., Astudillo, H.: 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, pp. 1–8 (2011)
16. Weiss, M.: Modeling the mashup ecosystem: structure and growth. R&D Management 1, 40–49 (2010)