Model-Based Search and Ranking of Web APIs across Multiple Repositories

Devis Bianchini, Valeria De Antonellis, and Michele Melchiori

Dept. of Information Engineering University of Brescia
via Branze, 38, 25123 Brescia, Italy
{devis.bianchini,valeria.deantonellis,michele.melchiori}@unibs.it

Abstract. Web API search and reuse for agile Web application development may benefit from selection criteria that combine several perspectives: they can be performed based on features used to describe APIs, or according to the co-occurrence of Web APIs in the same applications, or they can be driven through ratings assigned by designers who used the Web APIs for their own mashups. Nevertheless, different Web API repositories usually focus on a subset of these perspectives, thus providing complementary Web API descriptions. In this paper, we propose a unified model for Web API characterization. The model enables a cross-repository search of Web APIs and mashups, based on different kinds of similarity between them, identified regardless the complementarity of their descriptions. This unified representation improves retrieval results if compared with a Web API search performed over multiple repositories considered separately.

1 Introduction

Web API selection and aggregation, performed for mashup and short-living application development, may benefit from the adoption of criteria that combine different perspectives [1]: a *component* perspective (based on features used to describe Web APIs); an *application* perspective (i.e., information about mashups composed of the Web APIs); an *experience perspective* (including ratings assigned by web designers, who used Web APIs to develop their own mashups). The advantages coming from a multi-perspective Web API search have been confirmed by several approaches, that combined categories, tags and technical features like the adopted protocols and data formats in Web API descriptions with the co-occurrence of APIs in the same applications [2], with a quality-based model for Web APIs [3], with the network traffic around APIs and mashups, as an indicator of their success, and ratings assigned by designers [4]. Existing approaches rely on a single Web API repository. The `ProgrammableWeb` repository[1] is the most common one for sharing Web APIs and mashups. It contains over 11,500 Web APIs, where about 1,200 of them have been registered in the last year. APIs have been used in more than 7,400 mashups, while over 2,800 mashup

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

owners are registered in the repository. Nevertheless, different repositories emphasize complementary aspects to be considered for Web API search. Although `ProgrammableWeb` constitutes a well-known meeting point for the community of mashup developers, it does not provide a comprehensive Web API model that includes all the perspectives: it is mainly focused on a feature-based description of Web APIs (through categories, tags and technical features) and on the list of mashups that have been developed using the Web APIs. Another repository, `Mashape`[2], a cloud API hub leveraging a twitter-like organization, associated each Web API with the list of developers who adopted or declared their interest for it, denoted as *consumers* and *followers*, respectively. Other public repositories, such as `apigee` or `Anypoint API Portal`[3], focus on different and only partially overlapping aspects as well. This scenario brings to situations where: (i) the same Web APIs or mashups are registered multiple times within different repositories; (ii) Web APIs (resp., mashups) are searched and ranked according to distinct criteria in separate repositories, to meet different Web API (resp., mashup) descriptions (for instance, in `ProgrammableWeb` Web APIs are ranked with respect to the number of mashups they have been used in, Web API ranking performed on `Mashape` repository depends on the number of API followers). As proved in [1], performing Web API search and ranking on a comprehensive API descriptor, that includes different and complementary descriptive aspects, would improve retrieval results. This implies that it is not enough to search for APIs within distinct repositories considered separately and simply merge search results, but a real unified view over the repositories before starting the search is required. In this sense, similarity between Web APIs and mashups across different repositories should be exploited to enrich search results. Current Web API search scenarios lack of a model that provides a unified representation of Web APIs and mashups, to ease the identification of similar resources regardless the complementarity of their descriptions across different repositories [5]. Behind the advantage of avoiding multiple copies of the same API among the search results, although described with different properties depending on the repository from where API has been extracted, such a unified view would improve the retrieval outcomes as expected.

In this paper, we discuss about the definition of this model such that: (i) its unified representation covers the three perspectives mentioned above in Web API description, namely component, application and experience perspectives; (ii) it is part of a framework that enables the identification of different kinds of similarity between Web APIs and mashups, to provide a cross-repository search of these resources; (iii) it is integrated with a Web API and mashup search engine, that exploits similarity measures to properly access complementary information across repositories. A preliminary experimental evaluation confirms the improved search results, obtained by applying our approach, compared with Web API search performed on multiple repositories considered separately.

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

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

The paper is organized as follows: Section 2 presents a motivating example for introducing the unified model, that is discussed in Section 3; similarity criteria are presented in Section 4; in Section 5 we describe the Web API and mashup search based on the model; results of the preliminary evaluation are discussed in Section 6; a comparison with related work is provided in Section 7; Section 8 closes the paper.

2 Motivating Example

Let's consider a web designer who aims at including a face recognition functionality to access the private area of his/her own web site. Since developing this kind of applications from scratch would require very specific competencies and could be costly and time-consuming, the designer prefers to look for existing available Web APIs, that implement the desired functionalities, and examples of their use in mashups shared by other designers. Now let's consider the situation depicted in Figure 1. The figure reports some mashups and Web APIs that are relevant for the designer's purpose, obtained from `ProgrammableWeb` and `Mashape` repositories, by issuing a query with "face recognition" keywords. For example, the `Recognizer` mashup, where the `LambdaLabs Face` and `SkyBiometry` APIs are used together to provide multiple recognition services based on biometrics features, might fit the designer's goal. The `Recognizer` mashup can be used by the designer to infer how `LambdaLabs Face` and `SkyBiometry` APIs can be fruifully used together. Nevertheless, while the `SkyBiometry` API has been used in 49 mashups (including `Recognizer`, `Art4Europe`, `SaveUp` applications) and has been positively rated by other designers, the `LambdaLabs Face` API did not reached the same popularity. However, the latter API is similar to the `ReKognition` API in a different repository. This API is rated better than the `LambdaLabs Face` one and has 237 consumers and 372 followers on `Mashape`. Therefore, a satisfactory search should return `ReKognition` API ranked better than `LambdaLabs Face` API to be used, for instance, together with `SkyBiometry` API for developing a face recognition application.

If we would consider the two repositories separately, the `ReKognition` API taken from `Mashape` can not be associated with any mashup that is relevant for the designer's search, since in this repository mashups information are not shared. Similarly, the `SkyBiometry` API can not be suggested to be used together with the successful `ReKognition` API on `Mashape`. Public Web API repositories (e.g., `ProgrammableWeb`, `Mashape`, `apigee`, `Anypoint API Portal`) provide facilities that enable to search for both Web APIs and mashups (if available) by specifying one or more keywords, that are matched against textual descriptions of APIs and mashups, but they do not enable any advanced search and ranking strategy relying on the component, application and experience perspectives highlighted in the introduction. The only way a designer may combine different viewpoints for Web API ranking is to manually analyze basic sorting facilities provided by existing repositories (such as, the popularity on `ProgrammableWeb`, meant as the number of mashups where a Web API has been included, and the

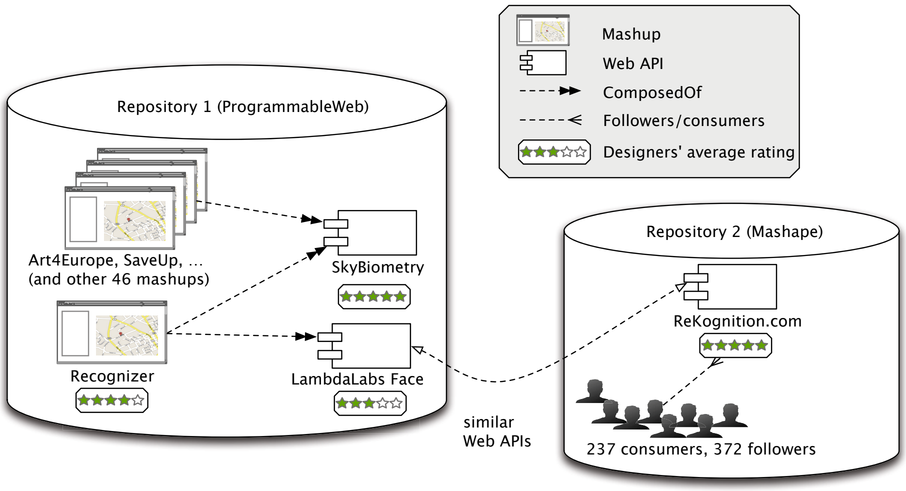

Fig. 1. The scenario considered in the motivating example, where a subset of face recognition APIs and mashups taken from the `ProgrammableWeb` and `Mashape` repositories is shown

number of followers of an API on `Mashape`). Finally, a seamless selection of Web APIs and mashups is necessary to be exploited as suggestions for a designer who aims at developing a new application from scratch (he/she may start from a single API, e.g., `SkyBiometry`) or at completing an existing one (he/she may learn from available mashups, viewed as sets of already aggregated APIs, e.g., `Recognizer`).

To overcome these limitations, in the following, we provide the designer with a unified model that enables a cross-repository search of Web APIs and mashups.

3 Web API and Mashup Unified Model

We introduce a unified *Web Mashup resource Descriptor* (hereafter, WMD), that embraces: (i) Web APIs as extracted from repositories; (ii) Web mashups, composed of one or more APIs. WMDs collect together these two kinds of resources by abstracting the set of their common features, as emerged through the analysis of the most popular public repositories and of state of the art approaches on Web API selection. Web APIs and mashups are basic elements of the *component* and *application perspectives* proposed in [1]. On top of this representation, ratings assigned by designers to Web mashup resources are considered to estimate their popularity (*experience perspective*). The WMD representation, derived from the proper combination of the three perspectives above, enables advanced search and ranking capabilities as described in the next sections. WMDs are collected and stored within a relational database, as shown in Figure 2.

Modeling Web Mashup Resources. A WMD is denoted by a unique identifier, corresponding to the URL of the Web API or the mashup, a human-readable

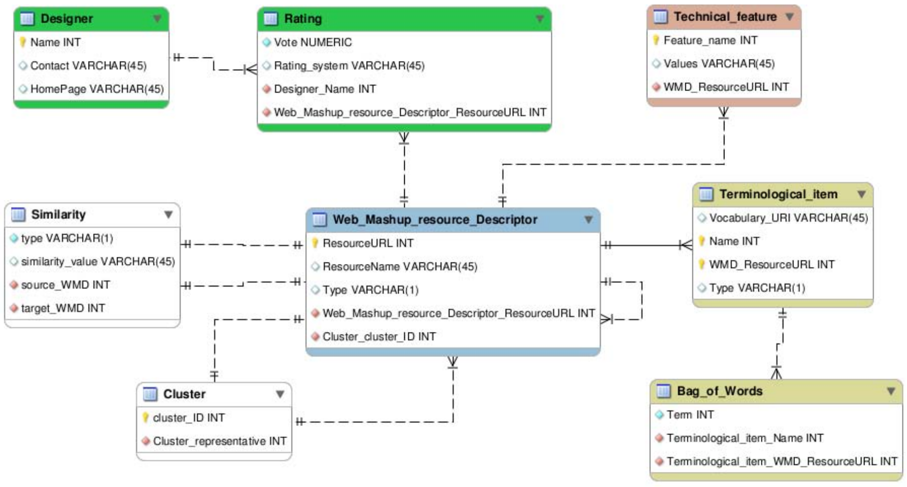

Fig. 2. The relational schema of the database containing the Web Mashup resource Descriptors, represented according to the unified model

name, a resource type, whose values, either M or W, denote the fact that WMD represents a Web API or a mashup. Each WMD is associated with a set of *terminological items*, that correspond to: (a) categories extracted from top-down classifications imposed within a given repository, where the resource is registered; (b) a term with explicit semantics, either a term extracted from WordNet or a concept extracted from an ontology in the Semantic Web context [6]; (c) a simple keyword or tag without an explicit representation of semantics. We distinguish keywords and tags as follows: tags are designer-assigned, bottom-up terms aimed at classifying WMDs in a folksonomy-like style, keywords are recurrent terms extracted from WMD textual descriptions using common IR techniques. A terminological item is in turn described by a representative name, an optional property that denotes the vocabulary, ontology, taxonomy or WordNet sense where the item is defined (denoted with `Vocabulary_URI` in Figure 2) and a set of other terms (denoted as *bag of words*) used to further characterize the item (optional). In particular, given an item t_i, if t_i is a category, the item name corresponds to the category name, its bag of words is empty and t_i is described by the taxonomy or the classification the category belongs to. If t_i is extracted from WordNet, its bag of words coincides with the list of synonyms of the term, and it is described by a reference to the WordNet sense the term belongs to. If t_i is a concept extracted from an ontology, its bag of words is composed of the names of other concepts related to t_i by semantic relationships in the ontology (in the current version of our approach, we consider OWL/RDF equivalence and direct subsumption relationships); moreover, t_i is described by the URI of the ontology where it is defined. Finally, if t_i is a keyword or a tag without explicit semantics, its bag of words is empty and the item does not refer to any vocabulary or taxonomy where its meaning is properly defined. A WMD is further characterized through the set of technical features (e.g., protocols, data formats, security mechanisms) that have been adopted for the Web API (if the WMD represents a single component) or for the APIs that compose the mashup (if the WMD represents a whole web application). A self-relationship is defined on WMDs, to denote Web API composition into mashups. The abstraction of Web API and mashup descriptions through a single data structure, namely the `Web_Mashup_resource_Descriptor` table, is meant to support the unified search of Web APIs and mashups as shown in the motivating example (Section 2). The search might start from a Web API (e.g., `SkyBiometry` in Figure 1), pass through mashups that contain the API (e.g., `Recognizer`) and find other Web APIs that have been used in the same mashups. Or it might start from a mashup and retrieve all the Web APIs that have been used together in the mashup. The final goal is to retrieve APIs or mashups that are relevant for the keywords, tags or categories specified in the request. In this sense, collecting both Web API and mashup representations as records in a single table ensures the maximum search flexibility. Conceptually, it is equivalent to a pair of entities, representing Web APIs and mashups, and a parent entity that collects common features. Moreover, a unique table, if properly indexed, allows for good performance while inspecting the database during search.

Modeling the Experience Perspective. Each WMD is also associated with *ratings* assigned to it by designers, who may be either Web API providers, Web mashup owners, or they may be WMD consumers, who rate resources according to their personal opinion. In [1] the value of quantitative ratings is selected by the designers according to the NHLBI 9-point Scoring System. In this paper, we assume the same system as well. Since we refer to quantitative ratings uniformly distributed over a continuous range, the mapping from a different scoring system to this one is possible. In our previous work, we discussed how designers can be further characterized by their development skill, that is self-declared, as shown in [1]. Designers' skill can be exploited to properly weight their ratings, considering as more trustworthy the opinions of more expert designers. In [1] designers' skills have been used by the system for Web API search, but have not been published to preserve the anonymity of designers' reputation stored in the database. A designer is allowed to know his/her own skill only. More sophisticated anonymization techniques can be investigated, if necessary, as well as methods to automatically estimate designers' skill based on their experience in mashup development. In our unified model, we did not considered designers' skills yet, since they can not be directly extracted from public repositories we considered. Future efforts will be devoted to the integration of this aspect as well.

Extracting and Organizing Descriptors. WMDs are acquired by means of wrappers, designed to invoke specific methods made available by public repositories to query their contents[4]. Moreover, wrappers may interact with proper modules implemented on the Java platform to extract specific kinds of

[4] See, for instance, `http://api.programmableweb.com` for the `ProgrammableWeb` repository or `http://www.mashape.com/mashaper/mashape#!documentation` for the `Mashape` repository.

terminological items: a module based on `jWordNet` library, used to extract synonyms and senses; a module implemented on `Jena`, to parse concept definitions from OWL/RDF ontologies; an IR-based module, to extract keywords from textual descriptions of resources (when available). Other wrappers can be added to the system according to the modularized architecture, that requires only to create wrappers and connect them to the specific modules listed above. Within the database, *similarity* values between WMDs are stored as well, as detailed in the next section.

4 Cross-Repository Similarity Metrics

Similarity metrics have been defined to compare Web mashup resources, represented according to the unified model. We will distinguish among the following kinds of similarity, that will be detailed in the following:

– *terminological similarity*, based on terminological items;
– *technical similarity*, computed as the number of common values of technical features among the compared resources;
– *compositional similarity*, aimed at measuring the degree of overlapping between two mashups or compositions of Web APIs, evaluated as the number of common or similar Web APIs in the two compositions.

As for the abstraction of Web API and mashup features within the same table, also these different kinds of similarity are abstracted using a single table in Figure 2, that is, `Similarity` table, where the `type` attribute denotes the kind of similarity (among terminological, technical and compositional), similarity value is always normalized in the $[0, 1]$ range and is computed between two WMDs, namely the `source_WMD` and the `target_WMD`. All metrics are symmetric. The aim is at exploiting similarity values to cluster descriptors in order to support their search (see Section 5). Starting from the hypothesis that clustered resources tend to be relevant for the same request [7], similarity-based clustering is exploited to identify a unique resource, that represents a bundle of similar ones (i.e., the *cluster representative*). The request is then compared against the representative resource instead of against each clustered ones, in order to filter out not relevant results, thus improving the resource retrieval effectiveness (see Section 6 on experiments).

Terminological Similarity between Web Mashup Resources. The terminological similarity between two Web mashup resources res_1 and res_2, denoted with $TermSim(res_1, res_2) \in [0, 1]$, is based on the comparison of their terminological items, that is:

$$TermSim(res_1, res_2) = \frac{2 \cdot \sum_{t_1 \in \mathcal{T}_1, t_2 \in \mathcal{T}_2} itemSim(t_1, t_2)}{|\mathcal{T}_1| + |\mathcal{T}_2|} \in [0, 1] \tag{1}$$

where we denote with $\mathcal{T}_i$ the set of terminological items used to characterize res_i, t_1 and t_2 are terminological items, $|\mathcal{T}_i|$ denotes the number of items in $\mathcal{T}_i$ set and $itemSim(\cdot)$ values are aggregated through the Dice formula. Pairs to be considered for the *TermSim* computation are selected according to a maximization function that relies on the assignment in bipartite graphs. The point here is how to compute $itemSim(t_1, t_2) \in [0, 1]$ given the different types of involved terminological items. The algorithm for the $itemSim(\cdot)$ calculus is shown in Algorithm 1.

Algorithm 1. The $itemSim(\cdot)$ calculus algorithm

```
Input  : Two terminological items $t_1$ and $t_2$.
Output: The calculated $itemSim(t_1, t_2)$ value.
1  if ($t_1$.type == C) and ($t_2$.type == C) (categories) then
2      $itemSim(t_1, t_2) = Sim_{cat}(t_1, t_2)$ (using $Sim_{cat} \in [0, 1]$ defined in [1]);
3  else if ($t_1$.type == WD) and ($t_2$.type == WD) (WordNet terms) then
4      $itemSim(t_1, t_2) = Sim_{tag}(t_1, t_2)$ (using $Sim_{tag} \in [0, 1]$ defined in [1]);
5  else if ($t_1$.type == O) and ($t_2$.type == O) (ontological concepts) then
6      $itemSim(t_1, t_2) = \text{H-MATCH}(t_1, t_2)$ (using the H-MATCH$\in [0, 1]$
       function, given in [8]);
7  else if ($t_1$.type == K) and ($t_2$.type == K) (keywords) then
8      $itemSim(t_1, t_2) = StringSim(t_1, t_2) \in [0, 1]$ (using the Levenshtein
       measure);
9  else
10     $\Upsilon_1 = \{t_1.\texttt{name}\} \cup t_1.\texttt{bagOfWords}$;
11     $\Upsilon_2 = \{t_2.\texttt{name}\} \cup t_2.\texttt{bagOfWords}$;
12     $itemSim(t_1, t_2) = max_{i,j}\{StringSim(t_1^i, t_2^j)\}$, where $t_1^i \in \Upsilon_1$ and $t_2^j \in \Upsilon_2$;
13 return $itemSim(t_1, t_2)$;
```

When the types of t_1 and t_2 coincide, proper metrics from the literature are used for the comparison. In all the other cases, a comparison between the names of terminological items using the Levenshtein string similarity measure ($StringSim(\cdot)$) is performed, except for the case of WordNet terms and ontological concepts, that are expanded with the bag of words assigned to each item in order to look for a better matching term in the set (in fact, for these kinds of items only, `bagOfWords` is not empty).

Technical Similarity between Web Mashup Resources. The technical similarity between two Web mashup resources res_1 and res_2, denoted with $TechSim() \in [0, 1]$, evaluates how many common feature values the two resources share. This metric is used to quantify the degree of compatibility between the resources in terms of protocols, data formats and other technical features. Feature values are compared only within the context of the same feature. Let $\mathcal{F}_X^{res_1}$ (resp., $\mathcal{F}_X^{res_2}$) the set of values admitted for the technical feature X associated with the Web mashup resources res_1 and res_2, respectively. The technical similarity between the Web mashup resources res_1 and res_2 is computed as follows:

$$TechSim(res_1, res_2) = \frac{1}{N}\Big[\sum_j \frac{2 \cdot |\mathcal{F}_j^{res_1} \cap \mathcal{F}_j^{res_2}|}{|\mathcal{F}_j^{res_1}| + |\mathcal{F}_j^{res_2}|}\Big] \in [0, 1] \quad (2)$$

where j iterates over the kinds of technical features, $|\mathcal{F}_j^{res_1} \cap \mathcal{F}_j^{res_2}|$ denotes the set of common values for the technical feature j on res_1 and res_2, $|\mathcal{F}_j^{res_k}|$ denotes the number of values admitted for technical feature j on resource res_k, N is the number of kinds of technical features on which the comparison is based. For example, if res_1 presents {`XML, JSON, JSONP`} as data formats and {`REST`} as protocol, while res_2 presents {`XML, JSON`} as data formats and {`REST, Javascript, XML`} as protocols, the $TechSim()$ value is computed as:

$$\frac{1}{2}\left[\frac{2 \cdot |\{\texttt{XML, JSON, JSONP}\} \cap \{\texttt{XML, JSON}\}|}{|\{\texttt{XML, JSON, JSONP}\}| + |\{\texttt{XML, JSON}\}|} + \frac{2 \cdot |\{\texttt{REST}\} \cap \{\texttt{REST, Javascript, XML}\}|}{|\{\texttt{REST}\}| + |\{\texttt{REST, Javascript, XML}\}|}\right] \quad (3)$$

In this example, `XML` is used both as data format and as `XML-RPC` protocol and it is considered separately in the two cases. The terminological and the technical similarity measures are equally weighted to compute the overall Web resource similarity, computed as follows:

$$\begin{aligned} &WebResourceSim(res_1, res_2) = 0.5 \cdot TermSim(res_1, res_2) + \\ &+ 0.5 \cdot TechSim(res_1, res_2) \in [0, 1] \end{aligned} \quad (4)$$

By contruction, if $res_1 = res_2$, then $WebResourceSim(res_1, res_2) = 1.0$.

Compositional Similarity between Web Mashup Resources. The *compositional similarity* between two Web mashup resources res_1 and res_2, that represent two Web mashups, denoted as $MashupCompSim(\cdot) \in [0, 1]$, measures the degree of overlapping between two mashups as the number of common or similar APIs between them, that is

$$MashupCompSim(res_1, res_2) = \frac{2 \cdot \sum_{i,j} WebResourceSim(res_1^i, res_2^j)}{|res_1| + |res_2|} \quad (5)$$

where res_1^i and res_2^j are two Web APIs, used in res_1 and res_2 mashups, respectively, $|res_1|$ (resp., $|res_2|$) denotes the number of Web APIs in res_1 (resp., res_2). $WebResourceSim(\cdot)$ values are aggregated through the Dice formula. Pairs to be considered for the $MashupCompSim$ computation are selected according to a maximization function that relies on the assignment in bipartite graphs.

5 Web Mashup Resource Search

The similarity metrics introduced in the previous section have been exploited for Web mashup resource search, that relies on the unified representation of resources through the model presented in Section 3. The basic idea of our search approach is to avoid a pairwise comparison of the Web API request $\mathcal{R}$ against each WMD extracted from the multiple repositories. Instead, we provide a WMD clustering based on terminological similarity. The request is compared against a representative WMD for each cluster, in order to identify the most relevant cluster(s) of WMDs. After the identification and the selection of such clusters, we perform a more in depth comparison between $\mathcal{R}$ and each relevant WMD according to all the types of similarities described in the previous section, distinguishing between WMDs that represent Web APIs and WMDs that represent mashups, and we provide the designer with a ranked list of relevant resources (either APIs or mashups) to be selected for his/her purposes. Finally, a further modification of the ranking is based on ratings (if available) assigned by other designers to Web mashup resources. In the following, we present the main phases of the search procedure.

Request Formulation. The request for a resource is formulated by the designer as follows: $\mathcal{R} = \langle \mathcal{K}_{\mathcal{R}}, \mathcal{F}_{\mathcal{R}}, \mathcal{M}_{\mathcal{R}} \rangle$, where $\mathcal{K}_{\mathcal{R}}$ is a set of keywords, $\mathcal{F}_{\mathcal{R}}$ is a set of pairs $\langle$`tech_feature=value`$\rangle$ and $\mathcal{M}_{\mathcal{R}}$ is a mashup (that is, a set of Web APIs) where the Web API to search for will be aggregated. The elements $\mathcal{F}_{\mathcal{R}}$ and $\mathcal{M}_{\mathcal{R}}$ in the request are optional. In particular, the latter is used to differentiate the kind of search that is being performed: (i) if $\mathcal{M}_{\mathcal{R}} = \emptyset$, then the designer is looking for a single Web API, for instance to start a new mashup application from scratch; (ii) otherwise, if $\mathcal{M}_{\mathcal{R}} \neq \emptyset$, the designer's purpose is to find a Web API to be included in an existing mashup, to complete it or to substitute a Web API within the mashup.

Clustering. The clustering procedure is performed off-line, thus not affecting the performance of the approach. We employ a hierarchical bottom-up clustering algorithm [9]. The term "hierarchical" means that this technique classifies WMDs into clusters at different levels of similarity. Pairwise comparisons between WMDs is performed according to the terminological similarity. In this way, we give more importance first to the terminological items, that are usually adopted to give a functional characterization of Web APIs and mashups in current repositories. Roughly speaking, we agree on the fact that categories, (semantic) tags, ontological concepts and keywords are adopted to categorize or classify the repository contents. The technique operates in a bottom-up way since it places a WMD into its own cluster and then proceeds through a progressive merging of clusters until all WMDs are clustered. Two clusters are merged first if they contain two WMDs, one from each cluster, with the maximum terminological similarity. The result of clustering is a similarity tree, where single WMDs are the leaves and intermediate nodes have an associated value representing the $TermSim()$ value at which a pair of clusters is merged. Only those nodes whose associated $TermSim()$ value is equal or greater than a threshold $\delta \in [0, 1]$ are considered as candidate clusters. Higher values of δ determine higher similarity between cluster members, but also an higher number of clusters with few members. This will impact on performance, as discussed in the experimental results. For each cluster C_k, the centroid is selected as C_k representative, denoted with $\widehat{C_k}$, that is the descriptor closest to all the other descriptors in C_k, considering the terminological similarity.

Search. The search procedure, starting from the set of clusters and the request $\mathcal{R}$ formulated by the designer, is described in Algorithm 2. In the algorithm, the set $\mathcal{W}$ of relevant Web APIs, the set $\mathcal{M}$ of relevant mashups and a buffer set Ω are initialized as empty sets to be further populated (row 1). The request $\mathcal{R}$ is compared against the centroids of clusters according to the $TermSim()$ similarity (rows 2-4). Clustering enables to apply terminological comparison only

Algorithm 2. Web mashup resource search algorithm

```
Input : The set $\{C_k\}$ of clusters; the request $\mathcal{R} = \langle \mathcal{K}_\mathcal{R}, \mathcal{F}_\mathcal{R}, \mathcal{M}_\mathcal{R} \rangle$
        formulated by the designer.
Output: The set $\mathcal{W}$ of ranked relevant Web APIs; the set $\mathcal{M}$ of ranked
        relevant mashups.
1  $\mathcal{W} = \emptyset$; $\mathcal{M} = \emptyset$; $\Omega = \emptyset$;
2  foreach Centroid $\widehat{C_k}$ do
3      if TermSim($\mathcal{R}$,$\widehat{C_k}$) $\geq\gamma_1$ then
4          $\Omega = C_k \cup \Omega$;
5  foreach $WMD_i \in \Omega$ do
6      Compute the WebResourceSim($\mathcal{R}$, WMD$_i$);
7      if WebResourceSim($\mathcal{R}$, $WMD_i$) $\geq\gamma_2$ then
8          if $WMD_i$.type == $W$ then
9              Add WMD$_i$ to $\mathcal{W}$;
10         else if $WMD_i$.type == $M$ then
11             Add WMD$_i$ to $\mathcal{M}$;
12 $\mathcal{M}$ = Rank($\mathcal{M}$, $\rho_1$); $\mathcal{W}$ = Rank($\mathcal{W}$, $\rho_2$);
13 $\mathcal{W}$ = ApplyRatings($\mathcal{W}$); $\mathcal{M}$ = ApplyRatings($\mathcal{M}$);
14 return $\mathcal{W}$ and $\mathcal{M}$;
```

to cluster centroids, thus avoiding overloading due to the pairwise comparison between the request $\mathcal{R}$ and each WMD extracted from the repositories. Relevant Web mashup resource descriptors are temporarily stored within the Ω buffer set (row 4). At this point, a more in depth comparison between the request and each relevant descriptor is performed according to the $WebResourceSim()$ metric, that takes into account both the terminological and the technical similarity. We note that two thresholds, namely γ_1 and γ_2, are used in rows 3 and 7 to filter out not relevant results. These thresholds are set within the $[0, 1]$ range and must be chosen according to the following considerations: (i) the higher the thresholds, the faster the search, since less resources are marked as relevant, but the search recall is obviously decreased; (ii) according to this viewpoint, the value of γ_1 dominates the one of γ_2, since resources are filtered out according to γ_1 first. We performed preliminary experiments on a training set of resources and we fixed $\gamma_1 \backsimeq 0.7$ to increase filtering and ensuring best precision; the recall reduction is balanced by the clustering procedure performed off-line, which collect together very close resources. On the other hand, we kept γ_2 low (i.e., $\gamma_2 \in [0.3, 0.5]$) in order to accept as much search results as possible.

Ranking. The ranking procedure applied in the last part of the algorithm ensures that the most relevant results are proposed to the designer first, moving the less relevant ones at the end of the results list. It is worth noting that, if the descriptor extracted from one of the repositories is incomplete (e.g., the technical features are not specified), the overall $WebResourceSim()$ value is lower. This meets our aim of proposing first Web resource descriptors that present a more complete specification, as extracted from available repositories.

Ranking is performed by invoking the *Rank*() function (row 12), that is differentiated with respect to the type of Web mashup resources. In case of mashups, a ranking function $\rho_1 : \mathcal{M} \mapsto [0,1]$ is used, that is computed as follows:

$$\begin{aligned}&\rho_1(WMD) = WebResourceSim(\mathcal{R}, WMD)\cdot\\&\cdot MashupCompSim(\mathcal{M}_{\mathcal{R}}, WMD) \in [0,1]\end{aligned} \tag{6}$$

According to this equation, the closer the WMD, that in this case represents a mashup, to $\mathcal{M}_{\mathcal{R}}$ in the request, according to the compositional similarity between mashups, the better the ranking of WMD in the results list. This means that those mashups, that are more similar to the mashup where the designer will insert the API he/she is looking for, will be suggested to the designer first. In case of APIs, a ranking function $\rho_2 : \mathcal{W} \mapsto [0,1]$ is computed as a variant of ρ_1, that is:

$$\begin{aligned}&\rho_2(WMD) = WebResourceSim(\mathcal{R}, WMD)\cdot\\&\cdot \frac{1}{|\mathcal{M}_{WMD}|} \sum_{k=1}^{|\mathcal{M}_{WMD}|} MashupCompSim(\mathcal{M}_{\mathcal{R}}, M_k) \in [0,1]\end{aligned} \tag{7}$$

where $\mathcal{M}_{WMD}$ is the set of mashups that contain the resource WMD, that in this case represents a Web API, $M_k \in \mathcal{M}_{WMD}$ is one of these mashups and $|\mathcal{M}_{WMD}|$ denotes the number of mashups. According to this equation, the closer the mashups to MR where WMD is used, according to the compositional similarity, the better the ranking of WMD in the results list.

Finally, a further promotion/penalty mechanism is implemented to take into account the ratings assigned by designers to Web mashup resources (row 13). The mechanism starts from the scoring system we adopted in our approach, that has been widely described in [1]. Here, we further extended this rating system adding ranking promotions/penalties as reported in Table 1. Depending on the rating in which the average score of a Web mashup resource falls, the position of the resource in the results list is increased or decreased as shown in the third column of the table.

Table 1. The 9-point Scoring System for the assignment of ranking promotions and penalties to the Web mashup resources

Rating (additional guidance on strengths/weaknesses)	**Score**	**Ranking promotion or penalty**
POOR (*completely useless and wrong*)	0.2	-4
MARGINAL (*several problems during execution*)	0.3	-3
FAIR (*slow and cumbersome*)	0.4	-2
SATISFACTORY (*small performance penalty*)	0.5	-1
GOOD (*minimum application requirements are satisfied*)	0.6	0
VERY GOOD (*good performance and minimum application requirements are satisfied*)	0.7	+1
EXCELLENT (*discreet performance and satisfying functionalities*)	0.8	+2
OUTSTANDING (*very good performance and functionalities*)	0.9	+3
EXCEPTIONAL (*very good performance and functionalities and easy to use*)	1.0	+4

6 Experimental Evaluation

The aim of the preliminary experiments described in this section has been to check the capability of our approach to provide improved search results compared with the separated use of multiple repositories. For the experiments, we considered: (i) the `ProgrammableWeb` repository, focused on mashups (built with Web APIs), Web API technical features, tags and categories; (ii) the `Mashape` repository, where APIs are classified through categories and associated with the number of designers interested in the Web APIs; (iii) the `Anypoint API Portal` repository, where interested designers, categories and technical features are considered for Web API characterization. We considered the application scenario presented in the motivating example. Moreover, we extracted about 1,400 Web APIs and related mashups (if available) from the three repositories, also considering other orthogonal application domains, related to different categories. Experiments have been run 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 each 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 $M' = M/\{\mathcal{W}\}$. The same request has been issued multiple times using different synonyms. For each request, we manually tagged as relevant the Web API $\mathcal{W}$ itself and all those Web APIs close to $\mathcal{W}$ in terms of protocols, data formats, similarity of mashups where they have been included, functionalities provided by the Web APIs (according to the documentation for the Web APIs provided in the considered repositories). We also asked five expert users to validate our choices. We selected as expert users a set of designers who developed at least ten mashups in the domain of interest, using different kinds of APIs, different data formats and protocols. The idea was to evaluate the search results using the classical IR measures of precision and recall and the average position of $\mathcal{W}$ among the first 10 search results. Precision refers to the number of relevant Web APIs within the set of search results, that is:

$$precision = \frac{|\{relevant\ Web\ APIs\} \cap \{retrieved\ Web\ APIs\}|}{|\{retrieved\ Web\ APIs\}|} \in [0,1] \tag{8}$$

Recall refers to the percentage of relevant Web APIs that have been effectively retrieved, that is:

$$recall = \frac{|\{relevant\ Web\ APIs\} \cap \{retrieved\ Web\ APIs\}|}{|\{relevant\ Web\ APIs\}|} \in [0,1] \tag{9}$$

Precision and recall measure the effectiveness of the retrieval process and should be maximized. The average position of $\mathcal{W}$ among the first 10 search results is a measure to evaluate Web API ranking. If, among search results, we find mashups, as allowed by our model, we considered as positive those mashups that contain at least a relevant API for the request. The results are shown in Table 2.

We note that, even if we merge the results from the considered repositories, queried separately, our approach presents better precision, recall and overall ranking. Better precision and recall are due to the particular $itemSim(\cdot)$ similarity we considered in our approach, that enables to overcome discrepancies due

Table 2. Preliminary evaluation results

	Precision	**recall**	**average $\mathcal{W}$ position**
`ProgrammableWeb`	0.62	0.59	7.9
`Mashape`	0.58	0.4	-
`Anypoint API Portal`	0.60	0.51	8.1
Union of results (considering repositories separately)	0.59	0.5	9.3
Our approach	0.91	0.79	2.1

to the adoption of synonyms instead of using the same term. Moreover, different repositories use different categories to classify the same APIs. This limitation cannot be solved simply merging search results coming from distinct repositories, while our approach is able to mitigate it by combining different kinds of similarity measures. Better ranking results are ensured since our approach enables to consider partially overlapping aspects coming from different repositories in a joint way. Given its relative complexity compared with simple keyword-based search and basic ranking facilities provided by available repositories, our approach pays in terms of response times, as shown in the second column of Table 3. However, by applying the clustering procedure, times significantly decrease, as evident in the third column of Table 3, using $\gamma_1 = 0.7$ and $\gamma_2 = 0.5$ (see Section 5).

Table 3. Response times (with and without clustering) in the experimental evaluation

	times without clustering (sec.)	**times with clustering (sec.)**
`ProgrammableWeb`	4.464	-
`Mashape`	3.234	-
`Anypoint API Portal`	2.360 -	
Union of results (considering repositories separately)	~10.058	-
Our approach	19.894	7.091

Of course, the cut-off imposed by thresholds γ_1 and γ_2 has an impact on the precision, recall and ranking of search results. But if we consider these values with and without clustering (see Table 4), they still outperform the values of the same measures if the search is performed on `ProgrammableWeb`, `Mashape` or `Anypoint API Portal` and by simply merging the sets of search results coming from these repositories. The considered public repositories do not present any difference with and without clustering, since a clustering mechanism is not provided on them. By varying γ_1 and γ_2 thresholds, precision, recall, ranking and response times change. For instance, if we decrease γ_1 to 0.5, the recall increases by 3%, but response times increase by 26%. Therefore, the increment of recall values does not justify decreased performance. Better response times are also ensured through the proper setup of δ threshold during cluster identification. If δ increases, more clusters are obtained, thus requiring an higher number of comparisons between the request and cluster centroids. However, also in this case, precision, recall and ranking slightly vary due to the more in-depth comparison between the request and each WMD within a candidate cluster as shown in rows 5-11 of Algorithm 2. Experimental results shown in Tables 2-4 have been obtained with $\delta = 0.6$.

Table 4. Preliminary evaluation results: impact of clustering on precision, recall and ranking of search results ($\gamma_1 = 0.7$, $\gamma_2 = 0.5$)

	Without clustering				**With clustering**			
	Precision	**recall**	**average $\mathcal{W}$ ranking**	**times (sec.)**	**Precision**	**Recall**	**average $\mathcal{W}$ ranking**	**times (sec.)**
Our approach	0.91	0.79	2.1	19.894	0.85	0.71	2.1	7.091

7 Related Work

Model-driven Web API selection and mashup development have been addressed by several approaches in the last years. Advanced solutions for mashup development [10] provide technologies, models and CASE tools to ease the designers in aggregating the component Web APIs, setting the interactions between them and generating the glue code required to deploy the mashup application. These models are not targeted at Web API or mashup search and ranking over public or private repositories. In [11] the formal model based on Datalog rules defined in [12] is proposed to search for mashup components (called *mashlets*): 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 [13] 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 recommendation of Web APIs. These approaches rely on a complex Web API model and a complex request formulation, that are unfeasible for Web designers' expertise, that is mainly focused on Web programming technologies. Different approaches followed, where simpler models have been discussed, not based on formal specifications or semantic annotations, and further extended with other aspects, such as the *collective knowledge* on Web API use, coming from experiences of other designers, and ratings assigned to APIs and mashups [1–4] (see [1] for a detailed survey).

With respect to these most recent approaches for Web API selection, we aim at providing a unified representation of Web APIs and mashups over multiple repositories, aggregating complementary descriptions of resources to search for, and we proposed a more flexible search, that enables seamless selection of Web APIs and mashups. To the best of our knowledge, this is the first attempt to provide a cross-repository search of Web APIs and mashups.

8 Concluding Remarks

In this paper, we discussed a unified model for Web API and mashup characterization and search across multiple repositories, based on selection criteria, that combine several complementary perspectives. As proved in [1], performing Web API search and ranking on a comprehensive API descriptor, that includes different and complementary descriptive aspects, would improve retrieval results. Preliminary experiments have been run to test effectiveness of Web API search in terms of precision and recall. Experiments demonstrated an improved search results, obtained by applying our approach, compared with Web API search performed on multiple repositories considered separately. Evolutions of this approach will be investigated to check how the productivity of Web designers is increased through the use of multiple repositories for Web API selection, where different repositories focus on complementary Web mashup resource descriptions. The possibility of further enriching the model through semantic aspects (e.g., semantic annotation of the unified view over the resources) will be investigated as well.

References

1. Bianchini, D., De Antonellis, V., Melchiori, M.: A Multi-perspective Framework for Web API Search in Enterprise Mashup Design. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 353–368. Springer, Heidelberg (2013)
2. Torres, R., Tapia, B., Astudillo, H.: Improving Web API Discovery by leveraging social information. In: Proceedings of the IEEE International Conference on Web Services, pp. 744–745 (2011)
3. Cappiello, C., Matera, M., Picozzi, M., Daniel, F., Fernandez, A.: Quality-Aware Mashup Composition: Issues, Techniques and Tools. In: Proc. of 8th Int. Conference on Quality of Information and Communications Technologies (QUATIC 2012), pp. 10–19 (2012)
4. 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), pp. 177–184 (2008)
5. Upadhyaya, B., Xiao, H., Zou, Y., Ng, J., Lau, A.: A Framework for Composing Personalized Web Resources. In: Chignell, M., Cordy, J.R., Kealey, R., Ng, J., Yesha, Y. (eds.) The Personal Web. LNCS, vol. 7855, pp. 65–86. Springer, Heidelberg (2013)
6. Bianchini, D., De Antonellis, V., Melchiori, M., Salvi, D.: Semantic-enriched service discovery. In: Proc. of the 22nd International Conference on Data Engineering (ICDE), pp. 38–47 (2006)
7. Trombos, A., Villa, R., van Rijsbergen, C.: The effeeffective of query-specific hierarchic clustering in information retrieval. Information Processing & Management (38), 559–582 (2002)
8. Castano, S., Ferrara, A., Montanelli, S.: Matching Ontologies in Open Networked Systems: Techniques and Applications. Journal on Data Semantics 2, 25–63 (2006)
9. Castano, S., De Antonellis, V., De Capitani di Vimercati, S.: Global Viewing of Heterogeneous Data Sources. IEEE TKDE 13(2), 277–297 (2001)
10. Matera, M., Picozzi, M., Pini, M., Tonazzo, M.: PEUDOM: A mashup platform for the end user development of common information spaces. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 494–497. Springer, Heidelberg (2013)
11. Greenshpan, O., Milo, T., Polyzotis, N.: Autocompletion for Mashups. In: Proc. of the 35th Int. Conference on Very Large DataBases (VLDB), Lyon, France, pp. 538–549 (2009)
12. 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)
13. 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 2011 Workshops. LNCS, vol. 6999, pp. 34–43. Springer, Heidelberg (2011)