Journal of Web Engineering, Vol. 11, No. 3 (2012) 247–268
© Rinton Press

QUALITY-DRIVEN EXTRACTION, FUSION AND MATCHMAKING OF SEMANTIC WEB API DESCRIPTIONS

LUCA PANZIERA, MARCO COMERIO, MATTEO PALMONARI,
FLAVIO DE PAOLI and CARLO BATINI
DISCo, University of Milan-Bicocca, Viale Sarca, 336 - U14 building
Milano, 20126, Italy
{*panziera,comerio,palmonari,depaoli,batini*}*@disco.unimib.it*

Received February 1, 2012
Revised May 24, 2012

The composition of Web APIs provides a great opportunity to Web engineers that can reuse existing software components available on the Web. Finding the best API, fulfilling a set of user requirements, among the many described on the Web is a key step in order to develop an effective Web application; however, Web engineers have little support in solving this problem due to poor search mechanisms and to the heterogeneity of sources and descriptions. Semantic technologies and matching algorithms provide accurate methods to match user requirements against a set of descriptions. Nonetheless, semantic descriptions of APIs are not available in practice. In this paper, we propose a method to extract information on Web APIs published in several Web sources and create semantic descriptions that can be then fused to deliver comprehensive descriptions associated with APIs. During the extraction process, we take into account that collected information has different levels of accuracy, currency, and trustworthiness to state a confidence level of the results. The method is based on the evaluation of the quality of the involved sources, the extracted values, and the overall descriptions. The resulting semantic descriptions are then matched with expressive user requirements to address the API selection problem.

Keywords: service matchmaking, semantic matching, quality assessments, Web data extraction, Web data fusion

1 Introduction

Web APIs have experienced an exponential increase in popularity and usage in the past few years [1]. These days, they are an important tool for Web developers; however, the process of discovery of available services and selection of the most appropriate APIs from a set of similar ones is mostly performed manually by looking for information on large public repositories that collect API descriptions or on other websites. For example, popular public repositories of API descriptions like *ProgrammableWeb*[a] and *Webmashup*[b], which describe, on April 2012, respectively more than 5700 and 1700 APIs; other information about these APIs are available from the website of each provider, but also from thematic webpages dedicated to group of services such as the Wikipedia page listing and comparing APIs of online music databases.[c]

[a]http://www.programmableweb.com/
[b]http://www.webmashup.com
[c]http://en.wikipedia.org/wiki/List_of_online_music_databases

Descriptions in different sources are often incomplete or inaccurate and do not follow a standard schema: they are often classified by categories (e.g. *ProgrammableWeb* and *Webmashup* use respectively 62 and 65 categories, on April 2012), tags and a limited set of metadata that describe the type, the functionalities and few other qualities of the provisioning. The number of functional-equivalent APIs characterized by different properties such as service availability, rating, and usage licensing, is also rapidly increasing. For example, *ProgrammableWeb* offers about 180 geo-localization services that differ in usage limits (e.g., requests per day), data licensing (e.g., copyright on maps) or geographic coverage (e.g., only U.S.); descriptions of many of these services are also available from *Webmashup*, with different descriptions (e.g. different categories, different qualities).

In other words, Web developers who want to find the APIs that better match their preferences, have to deal with disperse and heterogeneous sources to collect enough information to make conscious choices, which yields to costly and inaccurate decisions. In this paper we present an approach to semantic matchmaking of given user requirements against API descriptions collected from heterogeneous sources available on the Web. A set of wrappers, one for each source, allows for the extraction, at run time, of fresh data about target APIs that are transformed into semantic descriptions; a single, more complete, description of each target API is then built by fusing the extracted descriptions adopting a quality-aware method. Once the set of semantic descriptions is available, we exploit an extension of the semantic matching techniques defined in [2] to provide a final rank of the APIs. The approach has been implemented into PoliMaR-DD, a software tool that ensures the scalability and significantly improves the performance of available semantic matchmaking systems by the adoption of a distributed architecture [2]. A prototype discovery service based on PoliMaR-DD is available on line.[d] As a result, the approach proposed in this paper has the advantage of reducing the human effort to create semantic service descriptions and to consider always up-to-date descriptions, yet supporting effective and scalable semantic matching methods.

The assessment of data and information quality plays a major role in the approach described in this paper. Data and information quality has deserved in the last years considerable attention in the literature (see [3] and [4] for a comprehensive discussion on data and information quality). In our approach, three of the most relevant Web data quality dimensions, namely accuracy, currency and trustworthiness [5, 4], are considered for the effective fusion of descriptions extracted from Web sources. Moreover, we consider the overall quality of the fused policies in the ranking method. The definition of techniques for assessing these three quality dimensions for sources of Web API descriptions is another novel contribution of the paper.

The proposed discovery service is novel with respect to discovery engines proposed so far for several reasons: compared to [6, 7, 8, 9], our matchmaking techniques use semantics for describing services and user preferences; compared to [10, 11, 12, 13, 14, 12, 15, 16, 17], we do not assume that semantic service descriptions are already available, instead we build them by extracting and fusing descriptions from the Web; compared to [18, 19, 20], where automatic or semi-automatic methods are used to extract static semantic descriptions, we dynamically fuse descriptions when a request is issued by the user; therefore, our descriptions are always as up-to-date as the source descriptions are; furthermore, even when static approaches provide

[d] `http://jeeg.siti.disco.unimib.it:8080/polimar/discovery.jsp`

search mechanisms over documents crawled from legacy sources [19, 20], we match more expressive requests against more expressive descriptions, by allowing for constraints on non functional properties.

The rest of the paper is organized as follows. Section 2 discusses the motivation of this work and provides the background on semantic property modeling and service selection. Section 3 describes the policy extraction and fusion processes. Section 4 provides details on the distributed approach to the API matchmaking process. Section 5 discusses the distributed architecture of PoliMaR-DD that implements the approach. Section 6 provides experimental results by evaluating the effectiveness of the approach and the performance of the prototype. Related work is discussed in Section 7. We conclude the paper and outline future works in Section 8.

2 Motivation and Background

Let's consider a scenario in a music domain to motivate and illustrate the solution discussed in the paper. Mary, a music fan, is looking for on-line music services to find and download albums. She prefers MP3 or WMA files to be able to play the downloaded music on her devices. Moreover, she wishes to use *PayPal* as payment method, pay less than 1 Euro for each song, and use high-rated services.

Today, several services for music download exist on the Web. As of October 2011, the *ProgrammableWeb* Web API repository offers more than 120 music services, and many of them offer the download functionality (e.g., *eMusic, Napster, 7digital, Last.fm*[e]). Moreover, descriptions of these services are available on *Webmashup* and other sources that publish music service reviews (e.g., *cnet* and *Digital Trends*[f]). Since the manual analysis of such a number of services across several sources is difficult and time consuming, Mary would like to use a discovery tool that matches her preferences and Web APIs descriptions to find the best service to use. Such a tool should be able to dynamically aggregate data from the available sources in order to retrieve information, compile current descriptions of services, and return a list of matching services.

Given the large number of functional-equivalent services, traditional Web service discovery methods and techniques, which consider almost exclusively the functional properties, need to be extended with the evaluation of a wider set of non functional properties, such as Quality of Service, Licensing and Terms of Use. In the literature, several approaches considering these properties have been proposed (e.g., [11, 7, 9, 14, 16]), anyway some aspects have been neglected. In particular when dealing with:

- *heterogeneous formats of descriptions.* Service descriptions are specified as textual documents like Web pages and PDF files, micro-formats like RSS feeds (e.g., *ProgrammableWeb*), and, more recently, linked data (e.g., *iServe* [21]);
- *dynamic information.* Values that change frequently over time (e.g., response time, availability and user rating) needs to be collected at run-time;
- *dispersed information sources.* Often, information about a single service is available

[e]`http://www.emusic.com`, `http://www.napster.com`, `http://www.7digital.com`, and `http://www.last.fm`
[f]`http://www.digitaltrends.com/how-to/music-services-compared/` and `http://reviews.cnet.com/`

from multiple sources over the Web, such as Web API repositories, blogs, wikis and provider sites.

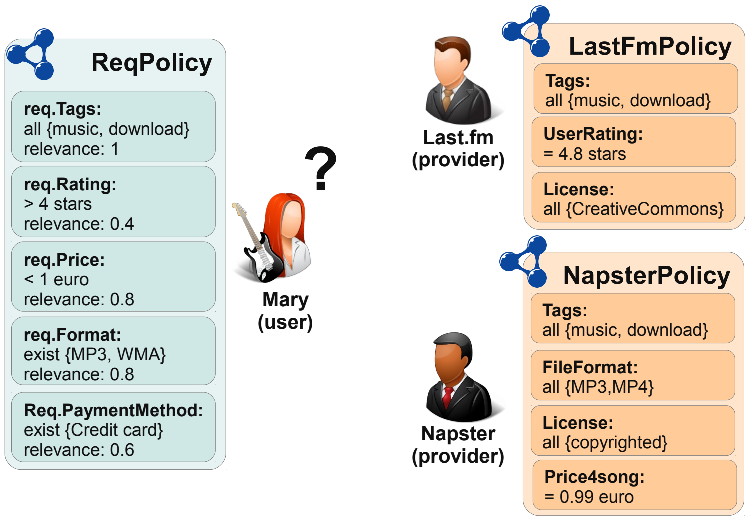

Fig. 1. Matching of a service request against two service offers based on PCM

2.1 Policy-based Semantic Modeling of Service Descriptions and Matchmaking

A key aspect when dealing with descriptions available from Web sources is that these descriptions are often collections of labels, which means that both properties and values are strings of characters without explicit semantics or reference to an ontology. In the past years, several ontologies (e.g. OWL-S [22] and WSMO [23]) and microformats (e.g. MicroWSMO [24]) for the semantic annotation of services have been defined. However, none of them have been widely adopted to become a de-facto standard, and they provide very limited support for modeling non functional properties. Thus, we have developed the *Policy Centered Meta-model* (PCM) that supports the definition of properties collected in *policies* (like the ones in Figure 1) associated with APIs [25]. In PCM, both qualitative and quantitative properties can be expressed, which means that property values can be either symbolic or numeric, and are defined by expressions. Expressions can include *constraint operators* (e.g., $<$, $>$, $=$, *exist*, *all*) and *units of measure* (e.g., *euro*, *metric*). PCM supports the definition of API descriptions, as well as *requested policies* that represent user preferences on API properties. Each preference is associated with a *relevance* value that specifies the importance that the requester gives to that preference.

Policies, properties, and expressions are defined by ontology elements in order to exploit high-level modeling and effective semantic matching techniques. PCM has been formalized both in OWL and WSML-Flight, two of the most popular semantic Web languages used in the context of Web service modeling[g]

PCM was exploited as intermediate format to develop a hybrid matchmaking process [26], where hybrid indicates the capability of addressing symbolic and numeric values, and

[g]PCM formalizations are available at: `http://www.siti.disco.unimib.it/research/ontologies/`

expressions. The process is composed of four phases: a *property matching* phase, which identifies couples of comparable requested and offered properties; a *local property evaluation*, that computes a *local matching score* (LMS) of match between each couple; a *global policy evaluation*, which makes use of LMSs to compute a *global matching score* (GMS) of match for each offered *policy*; and a *policy ranking* that performs a sorting based on the GMSs.

A clear limitation of the described process is the assumption that a set of semantic descriptions of APIs already exists and is one of the inputs of the discovery process (being a set of domain ontologies and the requested policy the other two inputs). In this paper, we make a significant improvement by addressing the automatic creation of API descriptions (i.e., policies) starting from properties and values extracted from heterogeneous sources available on the Web. Moreover, we adapt the matchmaking process to the new scenario, where the quality of the extracted data has to be considered when the matching scores are computed.

2.2 Overview of the Approach

In order to associate properties and values, available as strings of characters in Web sources, with explicit semantics, a wrapper for each source has been defined at design time. Wrappers, first, extract properties and values from documents in various formats retrieved from a given set of sources, and, second, give an interpretation to such properties and values by computing mappings to ontology concepts. While the extraction of current values is performed at runtime, the definition of mappings occurs at design time through the definition of *source-to-policy templates.*

Assuming that a single API is described in more sources, more than one *extracted policies* are created for each API. These policies need to be fused to provide a single, comprehensive description associated with that API. The fusion process selects the properties and the respective values from the extracted policies based on the assessment of the quality of the extracted values. An example of the extraction and fusion process is sketched in Figure 2: three PCM policies describing LastFm API are extracted from three sources (namely, *WebMashup*, *ProgrammableWeb* and *Wikipedia*) and fused to deliver a single description. Each fused policy is also associated with an overall quality score, which will be considered in the matchmaking process.

3 Quality-driven Policy Extraction and Fusion

Our approach extracts semantic policies from a variety of semi-structured Web sources, and represents them according to the PCM model in the OWL language. We can process every source that publishes its data through semi-structured documents, like FEED RSS, XML, JSON and XHTML documents, thus encompassing actual sources that provide or describe APIs. The policies extracted from each source are fused together to deliver one policy per API. In this section, we first describe three main data structures and knowledge elements used to support the policy extraction and fusion processes; afterwards, we define the algorithms that describe those processes.

3.1 Source-to-Policy Templates and Data Structures

Our approach uses three main data structures and knowledge elements along the policy extraction, fusion and matchmaking processes: *source-to-policy templates* to support the extraction

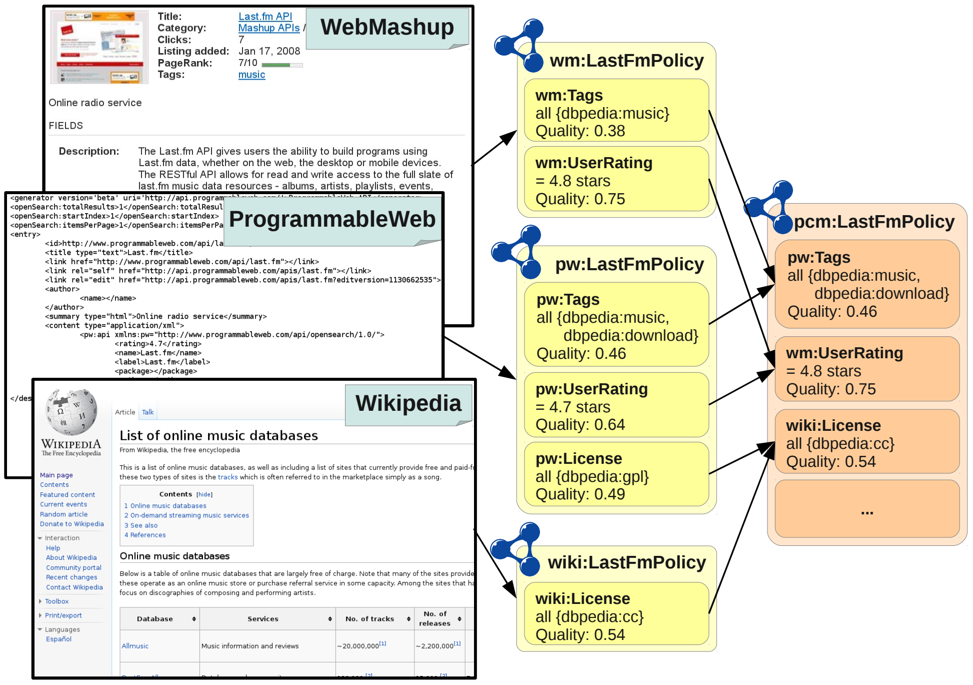

Fig. 2. Example of policy extraction and fusion

of data from Web sources, *value profiles* to provide a quality-based description of values of properties, and *semantic mappings* to state semantic equivalence between properties.

3.1.1 Source-to-Policy Templates

A *Source-to-Policy Template* (S2P template for short) associated with a source declaratively specifies the properties that will be included in the extracted policies and, for each property, the process of extraction of property values. In particular, an XPath query expression is associated with each property. The wrapper uses this expression to formulate queries over a semi-structured document to extract the desired data. Currently, S2P templates are manually written by experts, anyway, due to the limited number of sources to be considered (e.g., API repositories and blogs) this should not be regarded as a critical limit of the approach. Once a significant set of templates will be available, it is reasonable to foresee a reuse activity to address (semi)automatic definition of templates for new sources.

S2P templates are XML documents including the following main elements (details have been omitted for seek of clarity; datatypes or allowed values for each element are specified within round brackets):

- **apiName**(XPath): represents the path (defined by an XPath expression) to extract the name of the API;
- **apiProperty***[propertyID(Uri), propertyValue(XPath), propertyType(String), propertyDescription(String)]: represents a property that describes a characteristic of an API. The URI of *propertyID* unequely identifies the property in the extracted policies by

means of properties already defined in a Web ontology or defining a new one; the XPath expression of *propertyValue* extracts the value of that property; the *propertyType* states weather the property is qualitative or quantitative; finally, the *propertyDescription* is a free-text label describing the property.

- **socialProfileID**(Uri): represents the URI that identifies the profile of the source on a social network (if available).

- **lastUpdateDate**(XPath): defines the XPath expression that supports the extraction of the date of the last update of the document from which the API description is extracted.

An example of S2P template for *ProgrammableWeb* is represented in the following listing. The elements tagged by *apiName* and *apiProperty* are the ones used to build PCM policies. Additional data such as *socialProfileID* and *lastUpdateDate* provide information that will be used to assess the quality of the extracted policies.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:template
  xmlns:tns="http://pcm.itis.disco.unimib.it/s2ptemplate"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <tns:apiName>
    /feed/entry/content/api/name
  </tns:apiName>
  <tns:apiProperty>
    <tns:propertyID>
      http://www.programmmableweb.com/dataFormats
    </tns:propertyID>
    <tns:propertyValue>
      /feed/entry/content/api/dataFormats
    </tns:propertyValue>
    <tns:propertyType>qualitative</tns:propertyType>
    <tns:propertyDescpription>
      Data formats
    </tns:propertyDescription>
  </tns:apiProperty>
    ...
  <tns:socialProfileID>
    http://twitter.com/programmableweb
  </tns:socialProfileID>
  <tns:lastUpdateDate>
    /feed/entry/content/api/dateModified
  </tns:dateTag>
</tns:template>
```

3.1.2 Value Profiles

The *value profile* describes the current value of a property associated with an API and a set of meta-data about the quality of the extracted value; these meta-data will be used in the policy fusion process. Value profiles provide a compact representation of the information represented in PCM policies by means of RDF statements.

Formally, a *value profile* is defined as a quintuple $vp = < p, v, a, c, t >$, where p represents the property URI, v represents the extracted value of the property, and a, c and t represent respectively the *accuracy*, the *currency* and the *trustworthiness* of the value. Given a value profile $vp =< p, v, a, c, t >$ we call p and v respectively the property and the value of the value profile vp.

Since PCM policy supports multi-value qualitative properties, the actual value v of a property is represented, in general, by a set. *Accuracy*, *currency* and *trustworthiness* provide information about the quality of the data extracted from a source, denoting respectively a measure of how close are the data extracted to the true value, how up-to-date the values are, and how trustable is the source from which the value has been extracted; a formal definition of the quality dimensions and of the metrics for their evaluation is given in Section 3.3. An example of a *value profile* for the property *tags* of *Last.fm* is the following:

`< pw:tags, {dbpedia:music, dbpedia:social}, 0.33, 0.64, 0.5 >`

3.1.3 Semantic Mappings

Semantic mappings state equivalence relations between couples of properties $< p, q >$ belonging to two or more S2P templates. The mapping relation has the transitive closure property.

A mapping can be written in a semantic rule language (e.g., SWRL[h] and Jena Rules[i]) or expressed by semantic relations in RDF/OWL (e.g., *is-a* or *sameAs*). Mappings are defined, at design time, manually, by domain experts, or automatically, by ontology matching tools, such as AgreementMaker[j].

A single mapping involves properties of the same kind (e.g., qualitative/qualitative) that address the same characteristic of a service (as stated by domain experts); but, since values can be expressed by different units of measure, functions to convert the values expressed in a unit of measure into values expressed in the other, and vice versa, are associated with the mapping.

A set of *semantic mappings* is defined between the property URIs referred in two or more S2P templates at design time. The mappings are defined by equivalence relations between couples of properties $< p, q >$ and the transitive closure of the mappings is computed. We assume that the mappings are established between properties of the same kind (e.g., qualitative properties are mapped only to qualitative properties). An example of mapping of the *license* property between *ProgrammableWeb* and *Wikipedia* can be the following:

`< www.programmableweb.com/license, www.wikipedia.org/license >`

3.2 The Extraction and Fusion Processes

3.2.1 Policy Extraction

The policy extraction process returns a set of PCM policies by taking a data source s, a set of target APIs, and a S2P template for s. The set of APIs can be already known or be the result of a preliminary search phase, which identifies a set of similar APIs that have to be matched against a set of user requirements (see Section 4 for details). The API set contains the API names, each one associated with the URIs of Web documents that describe the API.

The algorithm defining the policy extraction process is described in Algorithm 1. In line 1, an empty set of policies related to the source is created. The algorithm identifies the set of documents to be considered, which are the ones belonging to the source and describing the

[h] `http://www.w3.org/Submission/SWRL/`
[i] `http://jena.sourceforge.net/inference/`
[j] `http://agreementmaker.org/`

Algorithm 1 Policy Extraction (*source*, *apiSet*, *s2pTemplate*)

```
1: $extractedPolicies(source) \leftarrow \emptyset$;
2: for all $d \in getSourceDocuments(source, apiSet)$ do
3:    for all $apiID \in getApiID(d)$ do
4:       for all $p \in getProperties(s2pTemplate)$ do
5:          $valueProfile(apiID, p) \leftarrow$ the value profiles extracted for the property
6:          $pcmPolicy(apiID) \leftarrow sem(valueProfile(apiID, p))$;
7:       end for
8:       $extractedPolicies(source) \leftarrow pcmPolicy(apiID)$;
9:    end for
10: end for
11: return $extractedPolicies(source)$.
```

APIs in the set (*apiSet*), and scans each document (line 2). The set of APIs described in the document (e.g. a page in Wikipedia describes more services) are identified by following the XPath expression specified in the *apiName* element of the S2P template (line 3). For each *apiProperty* specified in the S2P template (line 4), a *value profile* for that property is extracted (line 5) and then transformed into semantic statements that are added to the PCM policy associated with the API by the *sem* transformation function (line 6). The PCM policy is then added to the extracted policy set (line 8). Finally, all the extracted policies related to the considered source is returned (line 11).

The current value in a *value profile* is extracted using the XPath expression specified in the S2P template. Moreover, the value of a qualitative property is transformed into semantic values by extracting a set of ontology instances from the free-text description available in the original sources. Semantic values can be extracted by applying popular techniques such as *Named Entity Recognition* [27]. In this paper, we refer to DBpedia Spotlight [28], which performs Named Entity Recognition to extract entities from the DBpedia ontology; in this way we can take advantage of the large DBpedia dataset as a background ontology. To give an example, the result of DBpedia Spotlight is `http://dbpedia.org/resource/Creative_Commons_licenses` for the following input text: "Under Creative Commons license".

The quality scores included in the value profile are computed by using a set of metrics that will be described in details in section 3.3.

3.2.2 Policy Fusion

The policy fusion process, described in Algorithm 2, takes a set of target APIs (*apiSet*), a set of extracted policies associated with those services (*extractedPolicies*) and the set of semantic mappings (*mappings*) as inputs, and returns a set of *fusedPolicies* as output. For every API a (line 1), an empty cluster set of value profiles associated with a is created (line 2). Given an API a, all the value profiles that occur in the extracted policies that describe a are grouped into clusters (lines 3-11). Every value profile *vp* of every extracted policy describing a is considered: if the property of *vp* is equivalent, according to *mappings*, to the property of another value profile of an existing cluster (line 4), then the value profile is added to the cluster (line 5); else, the value profile is added to a new cluster that is added to the clusters (lines 7-8). Afterwards, the value profiles belonging to each cluster are fused into a unique value profile to deliver the fused policy for the referred API (lines 11-14). Finally, the *fusedPolicies* describing the APIs are returned.

Algorithm 2 Policy Fusion (*apiSet*, *extractedPolicies*, *mappings*)

```
1:  for all $a$ in $apiSet$ do
2:      $clusters \leftarrow \emptyset$;
3:      for all $vp \in getValueProfiles(a, extractedPolicies)$ do
4:          if $\exists c \mid c \in clusters \wedge < property(vp), property(c) > \in mappings$ then
5:              $c \leftarrow vp$;
6:          else
7:              $clusters \leftarrow newCluster$;
8:              $newCluster \leftarrow vp$;
9:          end if
10:     end for
11:     for all $c \in clusters$ do
12:         $pcmPolicy(a) \leftarrow fusion(c)$;
13:         $fusedPolicies \leftarrow pcmPolicy(a)$;
14:     end for
15: end for
16: return $fusedPolicies$.
```

3.3 Quality Dimensions and Fusion Principles

The fusion function (line 12 in Algorithm 2) implements a decision process which takes into account some quality dimensions associated with the extracted data. In particular we consider accuracy, currency and trustworthiness, which are three of the the most significant dimensions among the several dimensions investigated in the literature on information quality for information retrieved from the Web [4]. For each dimension we provide a short description and the chosen metric to assign a numeric value. Moreover, we show how the dimensions are considered in the decision process.

3.3.1 Accuracy

Accuracy of a value v is defined as the closeness of the value v to a reference value $v*$, considered as the correct representation of the phenomenon that v aims to represent [3]. In our approach the accuracy is associated with value profiles. Since values of quantitative properties are extracted from the original sources without additional processing, value profiles are assumed to be accurate for these properties. Instead, values of qualitative properties undergo a process that transforms data represented in free text into ontology instances exploiting DBpedia Spotlight. The metric to compute accuracy is therefore based on the similarity score that DBpedia Spotlight associates with every extracted entity. Let $sm(w)$ be the similarity score associated with an entity extracted from the word w by DBpedia Spotlight; $sm(w)$ is computed making use of the frequency of w in the text (f_w) and the Inverse Candidate Frequency (ICF) of w according to the following function (details on the similarity measure can be found in [28]):

$$sm(w) = f_w \cdot ICF(w),$$

Based on this similarity score, the *accuracy* a_{vp} of a value profile vp is computed by the following formula:

$$a_{vp} = \begin{cases} 1 & \text{if the property of } vp \text{ is quantitative} \\ \frac{1}{|EW|} \sum_{w \in EW} sm(w) & \text{if the property of } vp \text{ is qualitative} \end{cases}$$

where EW is the set of words from which entities have been extracted.

3.3.2 Currency

Currency refers to the promptness of an API description published in the Web source to reflect a change of the values of a property occurred in the real world. Since changes are typically performed at description level, the currency is assessed at that level of granularity, and afterwards inherited by the value profiles extracted from the descriptions. Let d be a source document and vp a value profile extracted from d; the currency for vp is defined by the following formula

$$c_{vp} = c_d = 1 - e^{-t^2},$$

where t is a time difference computed by the difference of days between the current date and the publication date in the source document d. Despite the proposed formula assigns a low *currency* value to old documents that could contain correct information, the proposed metrics is suitable in this domain because APIs change frequently and the documents describing them should be up to date. Often, say, one-year-old descriptions are assumed to be out of date. The XPath specified in the *lastUpdateDate* field of S2P templates supports the retrieval of the publication date related to a document, and therefore to the computed PCM policy of an API.

3.3.3 Trustworthiness

Trustworthiness of information in the Web is hardly evaluated on atomic data, since their origin and provenance is frequently unknown; as a consequence, trustworthiness of data is indirectly evaluated from trustworthiness of sources. Since most of the sources have public profiles (*source profile* in the following) on social networks (e.g. ProgrammableWeb has a public page on Facebook and a Twitter profile) can be *followed by* several users; we propose to evaluate the trustworthiness of a source s by analyzing the activity of social-network users with public pages associated with s. Examples of activities that can be considered are placing a "like" to a post of a source profile on Facebook, and "retwitting" a twit of a source profile on Twitter. Specific *social wrappers* are required to actually extract information on activities related to specific sources from the social networks specified by the *socialProfileIDs* defined in the S2P templates.

The activity rate ar of a source s on a social network sn is defined by the number of the followers of sn that have been active in a recent time window t, divided by the total number of followers of s in sn. Therefore, the activity rate $ar(s, sn)$ is computed as follows:

$$ar(s, sn) = \frac{|activeFollowers(s, sn, t)|}{|followers(s, sn)|}$$

The trustworthiness of a source can be defined by aggregating the active rates on several social networks. Given a set SN of social networks, the trustworthiness t_s of a source s is defined as follows:

$$t_s = 1 - e^{-\left[\sum_{sn \in SN} ar(s,sn)\right]^2}$$

Observe that we sum the active rates, since the trustworthiness of a source increases as the source is actively followed on more social networks; anyway, we use a gaussian function to normalize the trustworthiness in the range [0,1] to be used in the value profile. Therefore, in a value profile vp extracted from a source s, the trustworthiness score t_{vp} is assigned the trustworthiness t_s:

$$t_{vp} = t_s$$

3.3.4 The aggregated quality measure

The aggregated quality measure q_{vp} associated with a value profile vp is defined as the following weighted sum:

$$q_{vp} = w_a a_{vp} + w_c c_{vp} + w_t t_{vp}, \text{ where } w_a + w_c + w_t = 1 \text{ and } w_a > w_c > w_t \ .$$

The inequality constraints on the weights reflect the different granularity levels (value, document and source) at which the quality dimensions are assessed. The idea is that the more fine grained the method to measure a quality is, the higher the contribution of the quality to the aggregated measure should be; in other words, the aggregated quality of a value profile vp depends more strongly on its accuracy than on its currency and trustworthiness, which are qualities originally associated with the document and the source respectively from which the value profile has been extracted. Experimental results will confirm that this hypothesis improves the effectiveness of the fusion method. The weights w_a, w_c and w_t are established by experts of Web API domains (e.g., music service or social network API).

The aggregated quality measure provides a criterion to select the more reliable value profile among the ones that belong to a same cluster. Given a cluster c of value profiles, the $fusion(c)$ function of Algorithm 2 selects the value profile with higher aggregated quality.

4 Semantic Policy Matchmaking

The proposed matchmaking process has the task of producing a list of ranked APIs according to their matching score with a given user request (namely, *requested policy*). For seek of clarity, we describe the matchmaking process as an algorithm that takes as inputs: a *requested policy*, a set of *fusedPolicies*, and a set of *mappings*.

Properties can be either functional or non-functional. The former are typically represented by vectors of category names and tag labels, the latter consist of qualitative and quantitative expressions; functional properties and non functional properties defined by qualitative expressions are represented by qualitative properties in PCM policies; non functional properties defined by quantitative expressions are represented by quantitative properties in PCM policies.

The matchmaking process is defined by Algorithm 3 and consists of four main phases: *property matching* (lines 1-3), *local property evaluation* (lines 4-8), *global policy evaluation* (lines 9-13), and *ranking* (line 14). The algorithm extends the one described in [26] by considering also the assessment of the quality of the extracted data when comparing the policies.

Algorithm 3 Matchmaking (*requestedPolicy*, *fusedPolicies*, *mappings*)

```
1: for all $p \in requestedPolicy$ do
2:     $matchingProperties \leftarrow propertyMatching(fusedPolicies, mappings)$;
3: end for
4: for all $< reqp, offp > \in matchingProperties$ do
5:     $LMS_{reqp,offp} \leftarrow localPropertyEvaluation(reqp, offp)$;
6:     $Q_{reqp,offp} \leftarrow quality(offp)$;
7:     $propertyProfile_{offp} \leftarrow < LMS_{reqp,offp}, Q_{reqp,offp} >$;
8: end for
9: for all $fp \in fusedPolicies$ do
10:    $propertyProfiles(r) \leftarrow$ all the $propertyProfile_{offp}$ such that $offp \in r$;
11:    $GMS_{fp} \leftarrow globalPolicyEvaluation(propertyProfiles(r))$;
12:    $globalScores \leftarrow < fp, GS_{fp} >$;
13: end for
14: $rankedApiList \leftarrow policyRanking(globalScores)$;
15: return $rankedApiList$.
```

The *property matching* activity identifies the properties ($offp$) in the offered *policies* that match with the ones ($reqp$) in the *requested Policy* by using the available mappings. The result is a set of couples $< reqp, offp >$ that are evaluated to compute, for each couple, a *Local Matching Score* (LMS). The LMS measures, by a value in range [0, 1], how the offered property satisfies the requested one. Mathematical methods are used to match quantitative properties, and logical and set-based methods are used to match qualitative properties; we refer to [26] for the details about these methods.

In order to consider that policies can be automatically extracted and fused from different Web sources, the proposed algorithm takes into account the effectiveness of the extraction and fusion processes (see Section 3). We therefore associate every LMS with a quality value Q defined by the overall quality associated with each property in the fused policy; we call *property profiles* every such couple that have the form $< LMS, q_{vp} >$.

Then, a *Global Matching Score* (GMS) for each API is computed by exploiting the property profiles. The GMS is calculated using the following formula:

$$GMS_P = \frac{1}{|PP|} \sum_{p \in PP} w_{LMS} \cdot r_p \cdot LMS_p + w_q \cdot q_p$$

where $w_{LMS} + w_q = 1$, PP is the property profile set associated with the policy P and r_p is the *relevance* of the property profile p for the user. The weights w_{LMS} and w_q are established by domain experts. Finally, GMSs are used to delivery a ranked list of offered Web APIs ($rankedApiList$).

5 The Architecture of the Distributed Matchmaking Service

The solution discussed in the previous sections has been implemented by a service-oriented framework called Distributed and Dynamic Policy Matchmaking and Ranking (PoliMaR-DD), which is composed of three kinds of main components, as illustrated in Figure 3: a service matching orchestrator (SMO), service matching endpoints (SMEs), and wrappers, besides data sources available on the Web and a ranking component that provides the final list of Web APIs according to their scores.

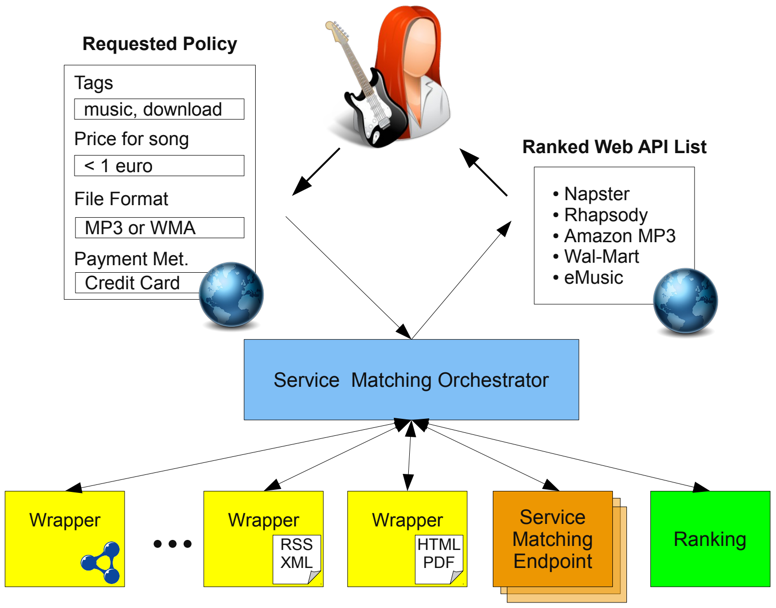

Fig. 3. The distributed matchmaking service architecture

SMOs are the access points through which users can issue property requests (*requested policy* in PCM terminology), and get the ranked list of selected services. The task of SMO components is to orchestrate the matchmaking process. SMOs rely on wrappers to extract the policies associated with candidate services by applying algorithm 1. Wrappers are software components specialized for the type of data retrieved from the considered sources, for example, semantic data (e.g., Linked Data from iServe), structured data (e.g., TopTenReviews Web APIs) and textual documents (e.g., Napster terms of use).

Once a set of policies referring to an API has been extracted from the sources, SMOs rely on SMEs to perform fusion and matchmaking, according to the algorithms 2 and 3. The fused policies allow the evaluation of the Local Matching Score *LMS* of single properties and the Global Matching Score *GMS* of policies with respect to the given user request. After computing the global scores, the ranking component is invoked to define the list to be returned to the user.

The service-oriented architecture in Figure 3 has been conceived to address two main issues: to cope with the distributed and unstructured architecture of the Web, which provides for a huge amount of heterogeneous dispersed data; and to make semantic techniques effective by increasing performance with parallelism. The wrappers are the answer to the first issue: by means of mappings, URIs and run-time extraction, the wrappers can provide updated profiles of available APIs. Moreover, the PoliMaR-DD components exchange data in RDF format to represent the output data as lightweight ontologies according to the Linked Data principles (that are compliant to REST interaction protocol) which facilitate the retrieval of additional information related to the selected Web APIs.

The second issue is addressed by SMEs, which can be instantiated in several processes to support parallel execution of fusion and matchmaking. Since such tasks may require the use of reasoners to deal with qualitative values, the possibility of concurrent execution dramatically improves the performance. Several tests conducted in the past have demonstrated the high execution-time cost of semantic matchmaking when a number of descriptions are involved [26]. With the proposed architecture, the load can be balanced among a dynamic set of SMEs, each of which has to evaluate smaller sets of descriptions.

6 Experimental results

In this section, the evaluation tests on the PoliMaR-DD prototype aim to show: (i) the effectiveness of policy fusion approach making use of real data extracted from the Web and (ii) a performance evaluation of the prototype that implements the distributed architecture proposed in Section 5.

For these evaluation tests, we selected four properties that are addressed by at least two sources: *tags*, that support the identification of the Web API functionality, *interaction protocol* (e.g., GET, POST), *data formats* (e.g, XML, JSON) and *licensing* (e.g., Creative Commons).

Table 1. Web API properties described in the considered sources

Web source	Tags	Int. protocol	Data formats	Licensing
ProgrammableWeb	$\surd$	$\surd$	$\surd$	$\surd$
Webmashups	$\surd$	$\surd$	$\surd$	-
Wikipedia	-	-	-	$\surd$

Moreover, we selected a data set composed of descriptions of 100 Web APIs published in at least two sources. The APIs come from five differents domains: music services (e.g., *Last.fm* and *Napster*), mapping services (e.g., *Google Maps* and *Bing Maps*), Internet services (e.g., *Amazon EC2*), social network APIs (e.g., *Twitter*, *Facebook* and *LinkedIn*) and enterprise APIs (e.g., CRM, ERP and HR applications).

In this case study, we used different parameters to measure the components of the *quality* of the extracted information. *Accuracy* depends on the technique adopted for the value identification. For this case study, we exploit DBpedia Spotlight to perform the extraction from textual descriptions and its similarity measure to compute the *accuracy* as introduced in Section 4. Instead, the publication date necessary to compute the *Currency* is extracted from specific fields offered by the source documents. Finally, *Trustworthiness* is measured by exploiting the followers activities of the three Web sources on *Twitter* and *Facebook*. All sources have a *Twitter* profile, but only *ProgrammableWeb* and *Wikipedia* have a *Facebook* public page. The activity rate is measured in the following way: on *Twitter*, by counting the number of followers that retweet to a source public profile; instead, on *Facebook*, by counting the number of users that like the public page and like posts of the source. The counting exploits the *Twitter*[k]and *Facebook*[l]Web APIs to collect data of the last three days of activity.

6.1 Effectiveness of the Policy Fusion

In order to measure the effectiveness of the policy fusion, we have measured the hit rate h based on the number of correct fusions. The evaluation of the fusion is based on the property

[k]Available at: `https://dev.twitter.com/docs/api`
[l] Available at: `http://developers.facebook.com/docs/`

characteristics. For the property *tags*, we considered a fusion correct when the selected value is the most appropriate to understand the API functionality. Instead, for the other three properties, the fusion is successful if the selected value is the most similar to the information published on the Web by the API provider after manual analysis. We run three tests with different configurations of the weights used to compute the aggregate quality measure. The experiments have the goal to analyze the correlation between the weights associated with the individual qualities and the effectiveness of the fusion process (measured by the hit rate).

Table 2. Effectiveness of the fusion process measured by hit rate

	w_a	w_c	w_t	Tags	Int. Prot.	Data For.	Licensing	Global
# fusions	-	-	-	92	86	74	6	258
Test A	0.33	0.33	0.33	84.8%	96.5%	95.9%	33.3%	90.7%
Test B	0.5	0.25	0.25	92.4%	97.6%	97.3%	66.6%	95%
Test C	0.6	0.3	0.1	94.6%	100%	98.6%	83.3%	97.3%

The overall results of the experiments are shown in Table 2, where w_a, w_c and w_t represent respectively the accuracy, currency and trustworthiness weight. These results confirm the hypothesis that the fusion is more effective if the constraint $w_a > w_c > w_t$ is enforced. The *licensing* property is the one that is most affected by a violation of the constraint because the fusion is performed between *Wikipedia*, which presents high trustworthiness (0.85) and low accuracy, and *ProgrammableWeb*, which presents medium trustworthiness (0.5) and good accuracy. Therefore, the high trustworthiness of *Wikipedia* has negative impact on the fusion accuracy.

Table 3 presents an example of the impact of different weight configurations (Tests A, B and C defined in Table 2)) on the fusion of two *licensing* values extracted from *ProgrammableWeb* and *Wikipedia*. The fusion process, which selects the value with highest aggregate quality (highlighted in boldface in the table), selects the wrong value under the configurations A and B (GPL as liquefied petroleum gas is not a license term); when accuracy is highly weighted, the correct value is selected. The accuracy of the value extracted from *ProgrammableWeb* is higher than the accuracy of the value extracted from *Wikipedia* because the source value is a longer text, which makes *DBpedia Spotlight* perform better.

Table 3. Quality assessment of the *license* values extracted for *FreeDB* Web API

Source	*ProgrammableWeb*	*Wikipedia*
Source value	"Under GNU general public license"	"GPL"
Extracted semantic value	GPL (license)	GPL (liquefied petroleum gas)
Accuracy	0.25	0.1
Currency	0.52	0.57
Thrustworthiness	0.5	0.85
Aggregate (Test A)	0.423	**0.506**
Aggregate (Test B)	0.38	**0.405**
Aggregate (Test C)	**0.356**	0.316

6.2 Performance Evaluation

In this subsection, we show that the transmission overheads introduced by the distributed protocols and the dynamic compilation of service descriptions are largely compensated by faster semantic matchmaking.

The PoliMaR-DD prototype used for the tests was implemented in Java SE 6 and exploiting Jena[m](version 2.6.4) for the management of the semantic data. The software was deployed on a Intel Core2 T5500 CPU @ 1.66GHz with 2 GB of RAM, that hosted a SMO, and eight Intel Xeon X5550 quad-core CPUs @ 2.67GHz with 2 GB of RAM, that hosted twenty SMEs. Each node is equipped with 64-bit Linux operating system (kernel version 2.6.24) with 100 Mbps network connection.

We performed two different tests. *Test A* to measure the execution time to perform the selection of 500 *offered policies* fused from *ProgrammableWeb*, *Webmashup* and *Wikipedia* repositories. The *requested policy* used for this test defines constraints on the four properties of the case study. We chose to extract properties with symbolic values to stress the evaluation of performance with intensive reasoning activities for both *matchmaking* and *local property evaluation*, and therefore to highlight potential performance improvements introduced by the distributed architecture.

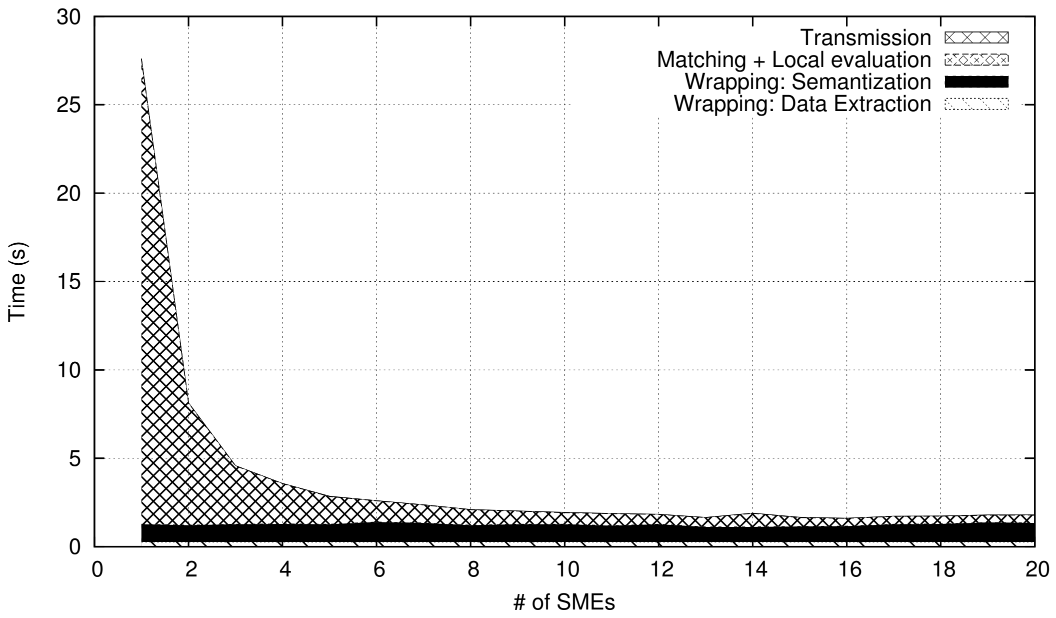

Fig. 4. Test A: Execution time for 500 *policies*

The results computed for ten measurements using the same *requested policy* are shown in Figure 4. The time measurements have been obtained by varying the number of involved SMEs. *Test A* highlights that the execution time behaves like a hyperbola arm. The results show that there is a relevant performance improvement already with few SMEs, resulting an

[m]http://jena.sourceforge.net/

abatement approximately of a 8.6 factor with 4 SMEs, and no relevant improvement with more than 10 SMEs.

Making use of the threshold identified by *Test A*, we performed *Test B* with the aim of analyzing the execution time of the most relevant process phases with 10 SMEs, with different number of *offered policies*.

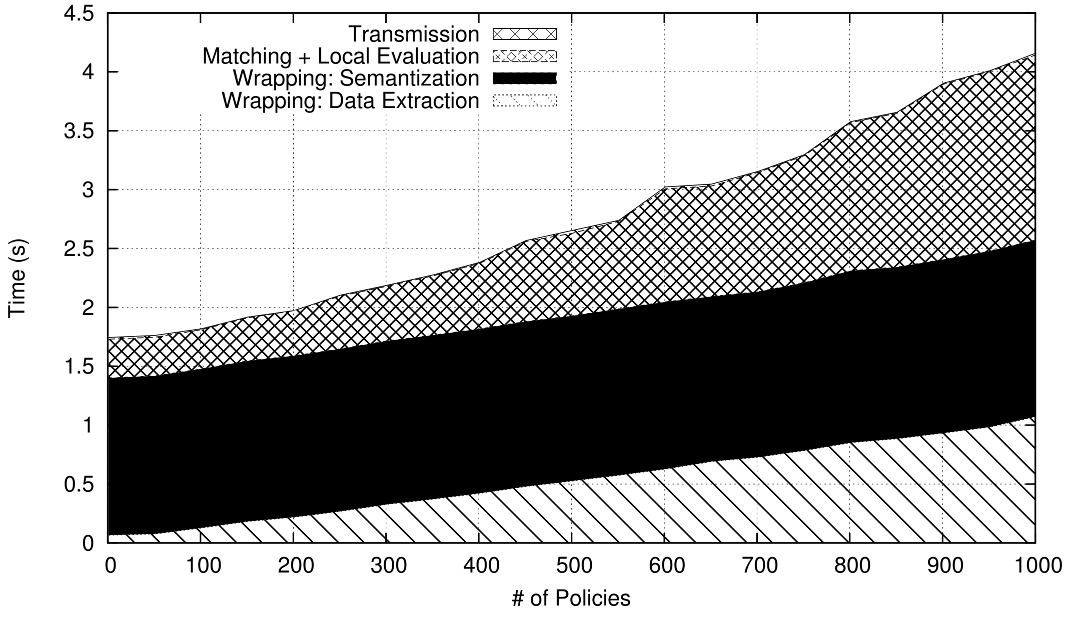

Fig. 5. Test B: Execution time for 10 SMEs

The results of *Test B*, reported in Figure 5, show that the most time consuming phase is the semantization. This measure depends on the response time of *DBpedia Spotlight*. In this experiment, we used the name identity recognition tool through its public Web APIs. However, it is reasonable to assume that the time consumed by the semantization phase could be reduced by installing the tool on a local machine. The second and the third more consuming phases are the *property matching* and the *local property evaluation*, because of intensive reasoning. The exponential increment of time consumption for reasoning intensive activities is a well-known problem reported in the literature [29]. In this experiment, the transmission time between components is negligible even if it can vary because of the dependency on several network parameters (e.g., the response time of the Web data source and the network traffic). The measured transmission time between the orchestrator and the 10 matching endpoints is very low (approximately 10 milliseconds) with respect to the other measures, also because all nodes are in the same local network. Finally, notice that the *policy fusion*, *global policy evaluation* and *policy ranking* activities do not appear in the charts since their execution time is negligible.

7 Related Work

Several approaches to service matchmaking have been proposed. To the best of our knowledge this is the first approach to matchmaking of service descriptions that holds all the following features: (i) the descriptions are semantic and semantic matching techniques are used; (ii) the descriptions (and the user requests) include non functional properties; (iii) the descriptions are dynamically extracted from available, not semantic, Web sources.

Seekda! is a discovery engine that performs semantic matching of Web API descriptions [18]. It uses a keyword-based semantic search method, and allows for filtering over few non functional properties (e.g. service availability, rating). This is the approach that is more similar to the one proposed in this paper. However, the descriptions are crawled from the Web and not dynamically extracted and fused. Moreover, semantics is used only for annotating functional aspects of the services. Therefore Seekda! does not consider the problem of extracting semantic values (i.e. instances from an ontology) from descriptions of non functional properties, and provides less expressive matching functionalities.

Two other approaches that provide discovery functionalities on descriptions available from Web sources have also been proposed [19, 20]. However, these approaches do not aim to extract semantic descriptions and support only keyword-based search. Moreover, the descriptions are crawled and not dynamically extracted and fused from multiple sources.

Most of Web APIs are RESTful services. Two RESTful semantic service matchmakers based on hREST/MicroWSMO model are proposed in [12] and [17]. The first work exploits similarity techniques only on service functionalities, the latter one performs the matchmaking through Linked Open Data covering functional characteristics, while search over non-functional characteristics is poorly supported. A matchmaking approach for RESTful and SOAP-based services is proposed in [13], but only functional properties are addressed. Finally iSEM [10] is an effective and efficient discovery engine for SA-WSDL and OWL-S service descriptions; however only functional aspects of services are considered in descriptions represented with this language. Moreover, these approaches assume the availability of semantic descriptions and do not extract them from available Web sources.

Service matchmaking approaches for SOAP-based services that cover also non-functional properties are presented in [11, 7, 9, 14, 16]. A hybrid architecture for service selection based on description logic (DL) reasoners and constraint programming techniques is proposed in [11]. An approach for the service selection based on the normalization of QoS values is described in [7]. A NFP-based service selection approach that modifies the Logic Scoring Preference (LSP) method with Ordered Weighted Averaging (OWA) operators is proposed in [9]. A framework for service selection that combines declarative logic-based matching rules with optimization methods is described in [14]. Finally, a QoS-based discovery tool is proposed in [16]. Even though the tool exploits WSMO, it performs only syntactical matchmaking based on an algebraic discovery model, without reasoning activities. All the above mentioned approaches that cover non functional properties assume the availability of rich semantic descriptions and do not extract them from available Web sources; the manual creation of such rich descriptions is costly and not scalable at Web scale.

The evaluation of quality dimensions to support integration of data from different sources is mainly addressed in the database research field. Several proposals exist in the area of virtual data integration architectures for quality-driven query processing, which return an answer to a global query, by explicitly taking into account the quality of data provided by local sources. All approaches consider several quality metadata associated with sources and the user query, that support the query processing activity. A comparative description and analysis of proposals appears in [3].

In the area of Web services, the evaluation of the quality of service descriptions is only marginally covered by few approaches in the literature [15, 30]. An approach to achieve the completeness of semantic Web service descriptions by generating OWL-S files, that provide a more complex representation of profile, grounding and partial process model, is proposed in [15]. A language and tool for Web service composition that considers their availability and quality of service description is described in [30].

Coming to specific quality dimensions evaluation, the expressiveness of the QoS description model is mentioned in several papers, such as [6], as a relevant aspect to be considered along Web service discovery.

The usage of quality dimensions to guide the data source selection process is addressed in [31]. The proposed approach is based on data source reputation that measures the trustworthiness and importance of a data source. Reputation is considered as a multi-dimensional quality attribute defined extending fundamental data quality dimensions (i.e. accuracy, completeness, and time) defined in [4] with additional dimensions (i.e. interpretability, authority, and dependability) that should be considered when assessing reputation, especially for semi-structured and non structured sources of information.

Several papers deal with the trustworthiness of Web data and Web sources. A data provenance model which estimates the level of trustworthiness of both data and data providers by assigning trust scores to them is defined in [32]. Based on the trust scores, users can make more informed decisions on whether to use the data or not.

The issues of trustworthiness and accuracy are investigated in [33] within the more general concept of credibility, in the domain of social networks managing volunteered geographic information. Credibility is seen as composed of two primary dimensions: trustworthiness and expertise. Trustworthiness is a receiver judgment based primarily on subjective factors, while expertise can also be subjectively perceived but includes relatively objective characteristics of the source, that correspond to source credentials and information accuracy.

Finally, the skills that Internet users need to assess the credibility of online information are analyzed in [5]. Credibility is investigated in its relationship with (i) accuracy, namely the degree to which a Web site is free from errors, (ii) objectivity, that involves identifying the purpose of the site and whether the information provided is fact or opinion, (iii) currency, whether the information is up to date, and (iv) coverage, that refers to the comprehensiveness or depth of the information provided on the site.

8 Conclusions

In this paper we have proposed an approach for a quality-driven and distributed selection of heterogeneous Web API descriptions. This selection is enabled by: (i) the extraction and fusion of property values from heterogeneous Web data sources; (ii) the mapping of the extracted property values in PCM-compliant descriptions and (iii) a distributed architecture implemented through the PoliMaR-DD prototype, in order to increase performances of the advanced semantic techniques adopted for the matchmaking of Web API descriptions.

Experimental results have shown the effectiveness of our approach using different Web data sources and a relevant performance improvement by distributing the selection process. Therefore, the overhead introduced by the property extraction and fusion from Web data sources and the wrapping in PCM-compliant descriptions is negligible.

The next steps of the research will focus on more in-depth experimentation of Web data extraction focusing on the management of the potential conflicts between data provided by different sources. Finally, we will focus on improving techniques for the automatic building of *source-to-policy templates* by exploiting fusion of schemas adopted by heterogeneous Web sources.

References

1. M. Maleshkova, C. Pedrinaci, and J. Domingue. Investigating Web APIs on the World Wide Web. In *proc. of IEEE European Conference on Web Services (ICWS 2010)*, pages 107–114, 2010.
2. Luca Panziera, Marco Comerio, Matteo Palmonari, and Flavio De Paoli. Distributed Matchmaking and Ranking of Web APIs Exploiting Descriptions from Web Sources. In *proc. of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2011)*, pages 1–8, 2011.
3. Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques.* Springer, 2006.
4. Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41:16:1–16:52, July 2009.
5. Miriam J. Metzger. Making sense of credibility on the Web: Models for evaluating online information and recommendations for future research. *J. Am. Soc. Inf. Sci. Technol.*, 58:2078–2091, November 2007.
6. Le-Hung Vu, Fabio Porto, Karl Aberer, and Manfred Hauswirth. An extensible and personalized approach to qos-enabled service discovery. In *proc. of the International Database Engineering and Applications Symposium (IDEAS 2007)*, pages 37–45, 2007.
7. X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A QoS-aware Selection Model for Semantic Web Services. In *proc. of the International Conference on Service Oriented Computing (ICSOC 2006)*, 2006.
8. Jingqi Wei, Dancheng Li, Jun Na, Jing Bi, Zhiliang Zhu, and Ying Chen. Web service publication and discovery architecture based on jxta. In *proc. of the International Conference on Service Sciences (ICSS 2010)*, pages 383–387, 2010.
9. H.Q. Yu and S. Reiff-Marganiec. A method for automated web service selection. In *proc. of the Congress on Services (SERVICES 2008)*, pages 513–520, 2008.
10. M. Klusch and P. Kapahnke. iSeM: Approximated Reasoning for Adaptive Hybrid Selection of Semantic Services. *proc. of the international conference on the Semantic Web (ESWC 2010)*, pages 30–44, 2010.
11. J. M. Garcia, D. Ruiz, A. Ruiz-Cortes, O. Martin-Diaz, and M. Resinas. An Hybrid, QoS-Aware Discovery of Semantic Web Services Using Constraint Programming. In *proc. of the Int. Conf. on Service-Oriented Computing (ICSOC 2007)*, pages 69–80, 2007.
12. Ulrich Lampe, Stefan Schulte, Melanie Siebenhaar, Dieter Schuller, and Ralf Steinmetz. Adaptive matchmaking for RESTful services based on hRESTS and MicroWSMO. In *proc. of the 5th Workshop on Emerging Web Services Technology (WEWST 2010)*, pages 10–17, 2010.
13. Jiaxuan Ji, Fenglin Bu, Hongming Cai, and Junye Wang. Ontology Model for Semantic Web Service Matching. In *proc. of International Conference on Information Computing and Applications (ICICA 2010)*, pages 181–188, 2010.
14. S. Lamparter, A. Ankolekar, R. Studer, and S. Grimm. Preference-based Selection of Highly Configurable Web Services. In *proc. of the International Conference on World Wide Web (WWW 2007)*, pages 1013–1022, 2007.

15. Preeda Rajasekaran, John Miller, Kunal Verma, and Amit Sheth. Enhancing Web Services Description and Discovery to Facilitate Composition. In *proc. of the 1st International Workshop on Semantic Web services and Web Process Composition (SWSWPC 2004)*, pages 34–47, 2004.
16. L.H. Vu, M. Hauswirth, F. Porto, and K. Aberer. A search engine for QoS-enabled discovery of semantic web services. *International Journal of Business Process Integration and Management*, 1(4):244–255, 2006.
17. H.Q. Yu, S. Dietze, and N. Benn. Autonomous matchmaking web services. In *In Proc. of International Conference on Computer Information Systems and Industrial Management Applications (CISIM), 2010*, pages 420–425. IEEE, 2010.
18. Nathalie Steinmetz, Holger Lausen, and Manuel Brunner. Web service search on large scale. In *proc. of International Conference on Service Oriented Computing (ICSOC 2009)*, pages 437–444, 2009.
19. E. Al-Masri and Q.H. Mahmoud. A framework for efficient discovery of Web services across heterogeneous registries. In *proc. of the IEEE International Conference on Consumer Communications and Networking (CCNC 2007)*, pages 415–419, 2007.
20. K. Gomadam, A. Ranabahu, M. Nagarajan, A.P. Sheth, and K. Verma. A faceted classification based approach to search and rank Web APIs. In *proc of the IEEE International Conference on Web Services (ICWS 2008)*, pages 177–184, 2008.
21. C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert, J. Kopecky, and J. Domingue. iServe: a linked services publishing platform. In *proc. of the workshop on Ontology Repositories and Editors for the Semantic Web (ORES 2010)*, 2010.
22. OWL-S Coalition. OWL-S. Semantic markup for Web services. Technical report, 2004.
23. D. Fensel, H. Lausen, A. Polleres, J. De Bruijn, M. Stollberg, D. Roman, and J. Domingue. *Enabling Semantic Web Services – The Web Service Modeling Ontology*. Springer, 2006.
24. J. Kopeckỳ, T. Vitvar, and D. Fensel. MicroWSMO: Semantic Description of RESTful Services. *Available at: http://wsmo.org/TR/d38/v0.1/20080219/d38v0120080219.pdf*, 1, 2008.
25. F. De Paoli, M. Palmonari, M. Comerio, and A. Maurino. A Meta-Model for Non-Functional Property Descriptions of Web Services. In *proc. of the IEEE International Conference on Web Services (ICWS 2008)*, pages 393–400, 2008.
26. M. Comerio, F. De Paoli, and M. Palmonari. Effective and Flexible NFP-based Ranking of Web Services. In *proc. of the International Conference on Service Oriented Computing (ICSOC 2009)*, pages 546–560, 2009.
27. Gerhard Weikum and Martin Theobald. From information to knowledge: harvesting entities and relationships from web sources. In Jan Paredaens and Dirk Van Gucht, editors, *PODS*, pages 65–76. ACM, 2010.
28. Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. DBpedia spotlight: shedding light on the Web of documents. In *proc. of the International Conference on Semantic Systems (I-SEMANTICS 2011)*, pages 1–8, 2011.
29. S.B. Mokhtar, A. Kaul, N. Georgantas, and V. Issarny. Towards efficient matching of semantic Web service capabilities. In *proc. of the international workshop on Web Services Modeling and Testing (WS-MATE'06)*, 2006.
30. Dmytro Rud, Andreas Schmietendorf, and Reiner Dumke. Performance modeling of ws-bpel-based web service compositions. In *proc. of the IEEE Services Computing Workshops (SCW 2006)*, pages 140–147, 2006.
31. D. Barbagallo, C. Cappiello, C. Francalanci, and M. Matera. Enhancing the selection of web sources: a reputation-based approach. *Journal of Software Technology*, 14(3), August 2011.
32. Chenyun Dai, Dan Lin, Elisa Bertino, and Murat Kantarcioglu. An approach to evaluate data trustworthiness based on data provenance. In *proc. of the 5th VLDB workshop on Secure Data Management (SDM 2008)*, pages 82–98. Springer-Verlag, 2008.
33. Andrew J Flanagin and Miriam J Metzger. The credibility of volunteered geographic information. *GeoJournal*, 72(3-4):137–148, 2008.