A Flexible and Semantic-aware Publication Infrastructure for Web Services

Luciano Baresi, Matteo Miraz, and Pierluigi Plebani

Dipartimento di Elettronica e Informazione – Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano (Italy)
{baresi, miraz, plebani}@elet.polimi.it

Abstract. This paper presents an innovative approach for the publication and discovery of Web services. The proposal is based on two previous works: DIRE (DIstributed REgistry), for the user-centered distributed replication of service-related information, and URBE (UDDI Registry By Example), for the semantic-aware match making between requests and available services. The integrated view also exploits USQL (Unified Service Query Language) to provide users with a higher level and homogeneous means to interact with the different registries. The proposal improves background technology in different ways: we integrate USQL as high-level language to state service requests, widen user notifications based on URBE semantic matching, and apply URBE match making to all the facets with which services can be described in DIRE. All these new concepts are demonstrated on a simple scenario.

1 Introduction

The publication and discovery of Web services [1] have been tackled in several different ways so far. While the community agrees on WSDL and BPEL, as description and composition languages, respectively, the efficient and effective *exposition* and *retrieval* of Web services are still open problems. For example, UDDI [2] and ebXML [3] are probably the two most "famous" registry solutions, proposals based description logics and ontologies [4–6] try to improve service discovery, while METEOR-S [7] and Pyramid-S [8] integrate registries and ontologies to offer semantically-enriched service publication and discovery. The lack of a winning solution pushed us to further analyze the problem and concentrate on the distributed publication of services as a way to improve both exposition and retrieval.

Even if all the main registry standards have moved towards distributed approaches, we think that this distribution cannot be defined a priori. We think that the information about available services must be moved closer to their possible users, and this must be done in a user-centric way. Our proposal lets users fully control their registries (i) by defining what they want to share with the others and (ii) by specifying the services potentially available on external registries they are interested in. Therefore, the paper concentrates on the distributed *user-centered* propagation of service information and on the discovery features that such a distribution enables. The discovery, in particular, is enriched with the adoption of semantic-aware analysis to improve the responsiveness of the system and help users with solutions (services) that are close enough to what they would have liked to get (even if they do not fully match their expectations).

The proposed interaction among registries exploits a *publish and subscribe* [9] (P/S, hereafter) communication infrastructure to allow for flexible and dynamic interactions. This means that each registry can decide the services it wants to publish, that is, the services it wants to share with the others. Similarly, it can declare its interests by means of special-purpose subscriptions. The infrastructure ensures that as soon as a registry publishes the information about one of its services, this same information is propagated to (and replicated on) all the registries that had declared their interest. Subscriptions (and unsubscriptions) can be issued dynamically and thus each registry can accommodate and tailor its interests (i.e., those of its users) while in operation.

The second key message of the paper is that oftentimes users are not only interested in services that fully and exactly match their requests, but they would like to know if there are "similar" solutions, that is, services that suitably adapted can be used instead of the ones part of the original request. This requirement is tacked in the paper in two different and orthogonal ways. User requests are formulated in a technology neutral and high-level query language, called USQL (Unified Service Query Language, [10]), and are then automatically translated into subscriptions suitably distributed through the communication infrastructure. On the other hand, the dispatching is powered with matchmaking capabilities to provide the different registries with semantically-enriched notifications, that is, information about services whose match with the original request (subscription) is within a given threshold.

The work presented in this paper builds on top of two existing proposals: DIRE (DIstributed REgistry, [11]), as for the communication framework among registries and the *facet*-based [12] description of services, and URBE (UDDI Registry By Example, [13]), for the matchmaking and semantic awareness. The integration of the two proposals allows us to consider a semantically-enabled replication infrastructure that supports different registry technologies (UDDI, ebXML, and the SeCSE registry[1]) by means of JAXR (Java API for XML Registries, [14]).

Besides the obvious integration of the two proposals, the novel contributions of this paper lie in: (i) the use of USQL as high-level language to state service requests, along with its automatic translation in terms of subscriptions for the communication infrastructure, (ii) the widening of notifications based on URBE semantic matching, and (iii) the extension of the URBE matching to all the facets with which services can be described.

The rest of the paper is organized as follows. Section 2 introduces an example scenario to motivate the proposal presented in the paper, while Section 3 summarizes background technologies. Section 4 describes the proposed infras-

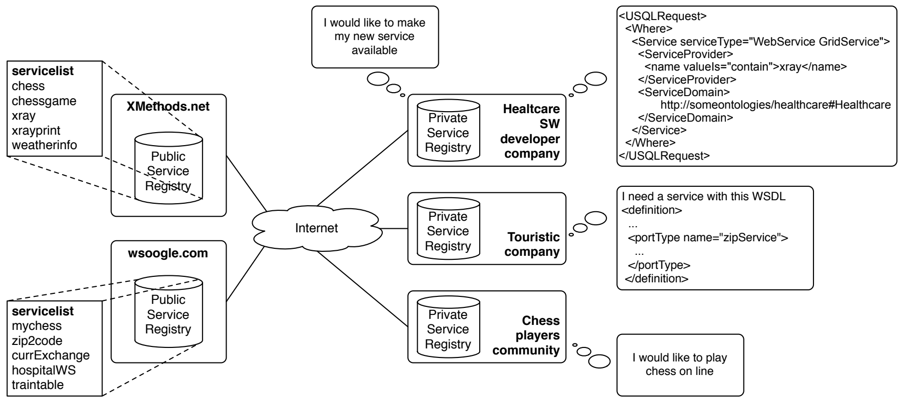
[1] `http://secse.eng.it`

tructure, along with the new features. Section 5 surveys some related proposals and Section 6 concludes the paper.

2 Example scenario

Even if the UDDI Business Registries by IBM, Microsoft, and SAP are not operated anymore, alternative “global” Web service registries are still available. Among the others, XMethods[2] and Wsoogle[3] are currently used worldwide, host Web services of any kind, and provide facilities to ease their discovery. Since the number of available services is always increasing, this section introduces the approach presented in this paper as a means to better exploit these “global” registries and increase the effectiveness of service discovery.

The example scenario[4] of Fig. 1 assumes the presence of three different (classes of) users interested in the Web services advertised by these global registries. The first is a company specialized in software development for healthcare solutions, which is interested in Web services able to support as many activities as possible in this application domain. The second is a tour operator willing to improve its Web site with mash-up services, while the third is a community of chess players who want to be aware of new opportunities (Web services) to play chess over the Internet.



```
<USQLRequest>
 <Where>
  <Service serviceType="WebService GridService">
   <ServiceProvider>
    <name valueIs="contain">xray</name>
   </ServiceProvider>
   <ServiceDomain>
       http://someontologies/healthcare#Healthcare
   </ServiceDomain>
  </Service>
 </Where>
</USQLRequest>
```

```
I need a service with this WSDL
<definition>
  ...
  <portType name="zipService">
   ...
  </portType>
 </definition>
```

Fig. 1. Example scenario.

All these three groups of users decide to run their own local registries and periodically browse XMethods and Wsoogle to find the services of interest and

[2] `http://www.xmethods.net`

[3] `http://www.wsoogle.com`

[4] This example only aims at exemplifying how our approach works; further technical considerations behind it are outside the scope of the paper.

update their local copies. Each requester is interested in services of different categories, but also their requirements are stated in different ways. For instance, on the average, chess players do not know WSDL, and can only express their requirements using chess- and QoS-related keywords (e.g., *chess* or *free chess server*). In contrast, the software company wants Web services with particular WSDL interfaces and is also interested in becoming a *quality* service provider for its clients.

All these activities are time consuming and the actual results heavily depend on the ability of who works on service discovery. Automatic ways to feed the local registries with no need for period updates would definitively ease their management, and would also help obtain better results (in terms of discovered services).

Our solution works in this direction. Each registry, be it global or local, must be connected to the communication infrastructure described in Section 4, and only has to declare its interests. The infrastructure grabs relevant services as soon as they become available directly from where they are published (mainly the two big repositories, in our example). Similarly, when one of the users (e.g., the software company) also holds the role of service provider, the infrastructure automatically publishes the new services onto the infrastructure and they (immediately) become available for the other interested registries.

Since Web services can be published, updated, and unpublished, the infrastructure is also in charge of updating the proprietary replicas as soon as new information (services) becomes available. In this scenario, all the services published in the general purpose registries are public by definition, and these registries are interested in all the public services in the local registries.

3 Background

This section briefly recalls DIRE, URBE, and USQL to provide the reader with a self-contained paper, and also highlight those elements that will be used in the next sections.

3.1 DIRE

DIRE[5] (DIstributed REgistry, [11]) provides a common service model for heterogenous registries and makes them communicate through a P/S middleware.

DIRE is in line with those approaches that tend to unify the service model (e.g., JAXR and USQL). Business data are rendered by pre-defined elements called Organizations, Services, and ServiceBindings, with the meaning that these elements usually assume in existing registries. Technical data are described by typed Facets, where each facet addresses a particular feature of the service by using an XML language. StandardFacets characterize recurring features (for example the compliance with an abstract interface), and we assume that they

[5] `http://code.google.com/p/delivery-manager/`

are shared among services. SpecificFacets describe the peculiarities of the different services (for example, particular SLAs or additional technical information). Users can attach new facets to services, even if they are not their provider, to customize the way services are perceived by the different registries (users), and to let them share this information with the other components attached to the communication bus.

The communication bus, which is based on a distributed P/S middleware called *ReDS* [15], decouples the interactions among components by means of a *dispatcher*. Each component can *publish* its messages on the dispatcher, and decide the messages it wants to listen to (*subscribe/unsubscribe*). The dispatcher forwards (*notifies*) received messages to all registered components. ReDS filters, which can both refer to shared standard facets and embed XPath expressions on the content of specific facets, let the different registries declare their interests for particular services. The goal is to disseminate the information about services based on interests and requests, instead of according to predefined rules.

A *delivery manager* is attached to each registry and acts as *facade*, that is, it is the intermediary between the registry and the bus and manages the information flow in the two directions. The adopted service model is generic enough to let different vendors create adapters for their registries. The adoption of the delivery manager does not require modifications to the publication and discovery processes used by the different users. They keep interacting with the (local) registry they were used to, but published services are distributed through the P/S infrastructure, which in turn provides information about the services published by the others (if they are of interest). In the end, each single registry is able to notify its users about the new services published in the other registries.

Notice that a registry can connect to the bus and declare its interests at any time. The infrastructure guarantees that a registry can always retrieve the information it is interested in by means of *lease* contracts. The lease period, which is configurable at run-time, guarantees that the information about services is re-transmitted periodically. This is also the maximum delay with which a registry is notified about a service. Moreover, if the description of a service changes, the lease guarantees that the new data are distributed to all subscribed registries within the period.

3.2 URBE

URBE[6] (UDDI Registry By Example) is an extension of typical UDDI Registries to support content-based queries, that is, the retrieval of services whose operations have a given input or output. Users submit the WSDL description of the requested Web service, and the system returns a ranked list of services whose signature is similar to the submitted one.

URBE supports service substitutability at both design- and run-time, and also the top-down design of BPEL processes. Traditional design approaches push the designer to identify the potential partner services and then design the BPEL

[6] `http://black.elet.polimi.it/urbe`

process by exploiting the previously selected WSDL interfaces (bottom-up approach). URBE allows the designer to start focusing on the definition of the process before selecting the Web services that fit it.

URBE's *similarity engine* compares WSDL descriptions of Web services. Assuming that users express their queries using WSDL, this component compares the submitted WSDL with the WSDL of all the Web services in the registry. Each comparison relies on function $WSDLSim : (wsdl_q, wsdl_p) \rightarrow [0..1]$, where the higher the result is, the higher the similarity between the two Web services is [16]. This value is obtained recursively by analyzing the overall signature, the operations, and their parameters. For each operation in $wsdl_q$, the similarity engine finds the operation in $wsdl_p$ with maximum similarity. This similarity depends on the similarity between the operations' names (calculated by $opSim$) and the similarity of their input and output parameters (calculated by $parSim$). Finally, the similarity of parameters depends on the similarity of the parameters' names and their data types. Figure 2 shows a high level overview of the similarity evaluation process.

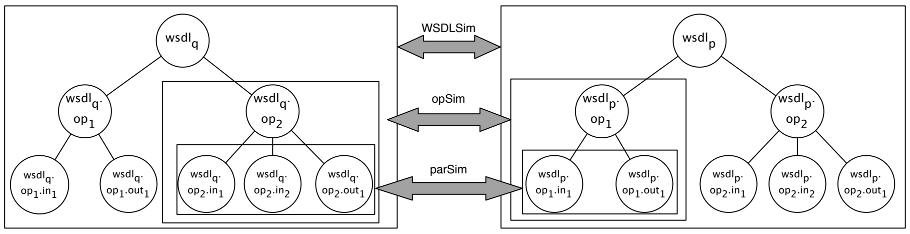


Fig. 2. Example similarity evaluation.

As a consequence, the similarity between two signatures heavily depends on the names assigned to the whole services, available operations, and exchanged parameters. The comparison between terms relies on a term similarity function $termSim : (t_i, t_j) \rightarrow [0..1]$. This function returns a value that reflects how the two terms t_i and t_j are semantically close: 1 if t_i and t_j are synonym, 0 if they are antonym.

To achieve this goal, *termSim* relies on two kind of ontologies: a *domain specific ontology* and a *general purpose ontology*. The first includes terms related to a given application domain. We assume that this ontology can be built by domain experts who analyze the terms included in the Web services published in the registry. The latter includes all the possible terms (at this stage we adopt Wordnet[7]).

The domain specific ontology offers more accuracy in the relationships among terms, while the general purpose one offers wider coverage. This happens because in a general-purpose ontology, a word may have more that one *synset*, each

[7] `http://wordnet.princeton.edu/`

corresponding to a different meaning. In contrast, we assume that in a domain-specific ontology each word has a unique meaning with respect to the domain itself. URBE can be configured to only use Wordnet to obtain a better coverage, to only use the domain specific ontology to obtain better precision, and to use both of them to gain the two advantages contemporarily.

Name similarity depends on the way two names are connected inside the ontology [17]. If we assume that WSDL descriptions are generated automatically, for example from Java classes, it is possible that the names of operations and parameters reflect the naming convention usually adopted by programmers: `getData` or `currencyExhange` are more frequent than the simple names directly included in the ontology. For this reason, if the terms are composite words, *termSim* tokenizes the word and returns the average similarity among the terms.

URBE is built on top of a UDDI implementation only for historical reasons, but the similarity engine has wider applicability —as we will see in Section 4. Such a module can also be used as a stand-alone component or be embedded in complex frameworks.

3.3 USQL

USQL (Universal Service Query Language, [10]) is an XML language to express service requirements in a technology agnostic way. The language allows users to abstract the particular protocol and details used by the registry, and focus on *what* services are supposed to offer. USQL, like SQL in the database world, is thus a language for searching services understood by different registry vendors. Its simplicity, expressiveness, and extensibility make USQL a good solution for both experts and unskilled users. For example, users without technical skills can search for services provided by certain organizations, while more skilled users can search for services that offer particular operations.

A dedicated engine translates both the queries from users into the format imposed by the particular registry, and the responses from registry-dependent descriptions into a *generic service model* (GeSMO). This model adopts a layered structure: the lowest level contains the concepts common to different services, while higher layers describe properties specific to particular services. This way, we have an extensible model able to capture different service types (e.g., Web services, Grid services, and P2P services) using orthogonal metrics, like semantics, QoS, trust and security, and management.

USQL queries can exploit *syntactic* information about Web services, for example, their names or the names of the organizations behind them. They can also embed *semantic* data that belong to users' domain knowledge, and QoS elements to predicate on the non-functional requirements that the service is supposed to comply with. Obviously, we can easily mix these data to conceive complex and sophisticated queries to retrieve the services of interest.

The language is based on a simple XML dialect to describe both required services and their QoS properties. In particular, there are elements to select services with a particular name, with a particular service description, or provided by a particular service provider. As for semantics, USQL supports different taxonomy schemes such as the *North American Industry Classification System* or the *United Nations Standard Products and Services Code System.* The user is able to specify requirements on the operations the service should provide. USQL also accepts constraints on the desired quality of service in terms of price, availability, reliability, processing time, and security. These orthogonal aspects fully support the user to retrieve services with the required functional and non-functional properties.

For example, if we want a service to send SMS messages, we might think of different properties. We can specify that interesting services must contain `SMS` in their name. We could also exploit their semantic characterization to discover only services provided by phone companies, or require that the WSDL interface of the service we want must have a `send` method that accepts a phone number and a short message as inputs. Finally we can also say that we are only interested in cheap services by setting a maximum price.

4 Proposed solution

A set of isolated registries would require interested providers to publish their services on each registry separately to proficiently advertise them and foster user awareness. This is exactly why we propose a flexible infrastructure that takes advantage of DIRE, URBE, and USQL to simplify the way services are published over a set of registries and ease their retrieval.

Once a new Web service becomes available, this information is not only stored in the registry used by the provider to publish the new service, but it is also forwarded to all the other registries interested in the same kind of services. This way, the provider can reduce the set of target registries to ideally a single one. In turn, even service retrieval becomes more effective: we move from a scenario where requesters have to browse different registries to find what they want, to a scenario where requesters only express their needs once, their requirements are spread around, and the information about interesting Web services is automatically moved onto their registries.

The proposed infrastrcture is shown in Figure 3. Its core is similar to the one adopted in DIRE, where a *communication bus*[8] connects all the companies that own a registry. Generally speaking, every registry can be used to both publish new Web services and retrieve interesting ones. Each registry is connected to the bus by means of a *delivery manager*, which is in charge of the different registry technologies and also manages the information flow in the two directions.

The first significant addition of this paper is that the *communication bus* also relies on an extended version of URBE's *similarity engine* for the comparisons between requests (subscriptions) and available services.

The figure also shows how the proposed solution works with our running example. For the sake of simplicity, we assume that XMethods and Wsoogle

[8] We can easily assume secure and reliable interactions since the P/S communication infrastructure is in charge of it.

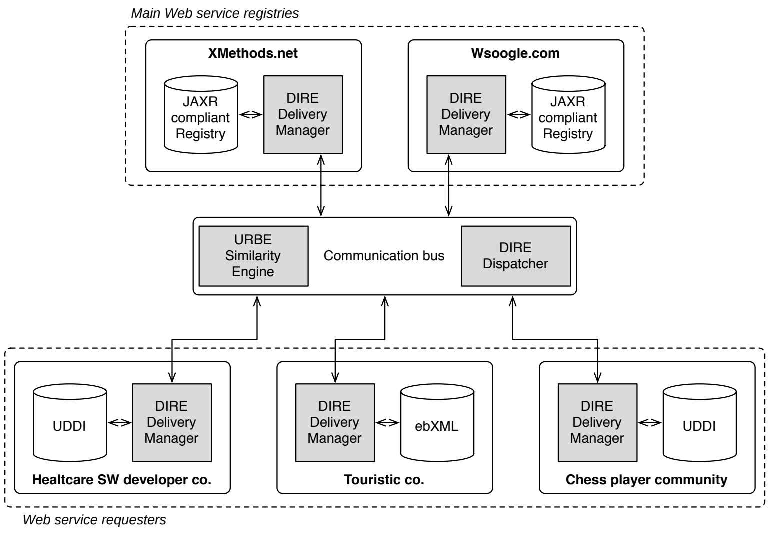


Fig. 3. Proposed infrastructure exemplified on the example scenario.

are connected to the communication bus via a *delivery manager*. Since we are considering general purpose registries, with a high number of services, the P/S infrastructure could become the bottleneck of the entire system. If this were the problem, "thematic" buses (e.g., about games, health, and so on) would help split the traffic, and therefore manage the performance of the communication infrastructure. A thematic bus may also be organized around a domain ontology and, in this case, such an ontology could be used by URBE to compute the similarity among service interfaces. Said this, the explanation of the approach can easily consider a single bus without losing any significant detail. In addition, we also assume that all the three actors (already introduced in Section 2), interested in new Web services, have a proprietary service registry, along with a delivery manager properly connected to the communication bus.

The introduction of USQL and URBE aims at (i) affecting the way subscriptions can be expressed and (ii) improving the effectiveness of the filtering performed by the *dispatcher* when it has to decide whether to forward the information about a new service or not. The next sections illustrate these two aspects in detail.

4.1 USQL-based subscriptions

This section explains how users can interact with the *delivery manager* using USQL. As stated before, the main benefits are the independence of any particular technology and the openness towards non technical users. Our additional goal is to leverage these features inside our distributed environment, which means translating USQL queries into appropriate subscriptions to support service lookup.

Since USQL queries are nothing but verbose XML documents, the presentation in this section is organized around increasingly complex examples.

Moving back to the scenario of Section 2, let us suppose that the chess players had discovered an interesting set of services provided by an organization called AcmeChess. Now, the community is looking for other services, and given the good reputation, they would like to know whether there are new services provided by AcmeChess. For example, if we think of a simple facet with tag `serviceprovider`, the filter (i.e., the XPath expression) could be: `//serviceprovider = "AcmeChess"`.

Another example considers the tourist operator that uses USQL in a smarter way and exploits the semantic facets. If we assume the existence of a standard facet that represents a shared taxonomy about travels, we can easily select all the services related to it. The subscription behind this query would predicate on the relationship between the standard facet `traveling` and the various services to allow the delivery manager to retrieve all the services in the domain of interest. Otherwise, if the ontology were more dynamic and lightweight, it could be embedded into a specific facet used to describe a particular service, and the filter would be: `contains(//service/ontology, "traveling")`.

The third actor, that is, the health-care software development company, selects services by analyzing the interface they offer. For this reason, the first query they create analyzes the operations exposed by the different services to select the ones relevant for their goals. For example, the following USQL query:

```
<USQL version="1.0" xmlns="urn:sodium:USQL">
 <USQLRequest>
  <Where>
   <Service serviceType="WebService P2PService GridService">
    <Operation minDegreeOfMatch="0.75">
     <Inputs>
      <input>
       <type valueIs="contain">integer</type>
       <semantics ontologyURI="http://sodium/ontologies/healthcare">
        http://sodium/ontologies/healthcare#ssn
       </semantics>
      </input>
     </Inputs>
     <Outputs>
      <output>
       <type valueIs="contain">string</type>
       <semantics ontologyURI="http://sodium/ontologies/healthcare">
        http://sodium/ontologies/healthcare#surname</semantics>
      </output>
     </Outputs>
    </Operation>
   </Service>
  </Where>
 </USQLRequest>
</USQL>
```

requires a service that gives the patient's name knowing his/her social security number. It also checks that the service only requires an integer, tagged with taxonomy's node `ssn`, and returns a string, tagged with node `surname`. This query is transformed into a filter with three parts. The first part is an XPath that analyses the WSDL facet to check whether there is an operation with an integer input and a string result. The second part checks whether the input parameter of the operation refers to taxonomy's node `ssn`, while the last part checks the result of the operation, and controls that it represents a `surname`.

When the query is issued at run-time, probably generated by an application component in charge of replacing a faulty service, the infrastructure must guarantee exact results (i.e., only retrieve services that can replace a previous service without human intervention). This is what the standard XPath matching technique provides. When we move the problem at design-time, users create queries to understand what Web services they can exploit. The results of these queries are usually not directly plugged in the system and approximate results better help understand the different alternatives: USQL allows us to specify the degree of matching, and URBE helps the communication bus retrieve services that match the approximation.

The health-care development company can also decide to include QoS requirements in its queries. For example, they can decide to only bind to services with high availability and low processing time. All these elements can easily be translated into both XPath queries directly, for full matches, and complete required facets, then passed to URBE for evaluation, for partial matches.

4.2 Similarity-based subscriptions

The extended version of URBE's *semantic matching* provides two main functions: $termSim$, to evaluate the similarity between two terms, and $facetSim$, to evaluate the similarity between two facets, which is an extension and generalization of the original match making that was limited to WSDL or SAWSDL descriptions. The infrastructure we propose exploits these two functions whenever users want to move from *exact* matches to *relaxed* ones, that is, users are satisfied even if their requirements are not totally fulfilled.

To notify the publication of a new Web service, the dispatcher was used to verify that the new description and the subscription had a perfect match. For example, if the chess player community submits a subscription with an XPath expression as `//service/type="chessgame"`, their registry will never receive Web services with a facet whose field `type` is `chess`. Since we cannot force all the actors to use the same terms, we can take advantage of function $termSim$. We introduce the clause `relaxed[sim]`, and append it to the XPath expression included in the subscription, where $sim \in [0..1]$ is a threshold that specifies the minimum admissible similarity. In the example, the subscription could be `//service/type="chessgame"relaxed[0.5]` to make the dispatcher notify the publication of new Web services with $termSim(\text{'}chessgame', \text{'}chess') \geq 0.5$. Notice that the definition of this similarity threshold is not easy for unskilled users as the average chess player. For this reason, we assume that the `relaxed` clauses can actually be set by transforming a qualitative scale (e.g., high, medium, and low) into the corresponding threshold values.

Function $facetSim$ can be exploited in case the requester is skilled enough to know what a facet is (that is, the structure of an XML document used to describe a service). Thus, the requester wants a Web service that is not only related to a given type, but it is described in a very particular and technical way. To make the substitution possible, the substitute Web service has to expose a facet that is equal to or at least similar to the facet of the failed service.

Since a WSDL description is nothing but a particular facet, a subscription of the healthcare software company could be `//service/wsdl='http://www.hcc.org/x-rayPrinter'relaxed[0.8]` where the WSDL corresponds to a Web service able to print and deliver X-rays to patients. When a company develops a new service of this kind and publishes it onto one of the general purpose registries, the dispatcher compares its WSDL with the WSDL at `http://www.hcc.org/x-rayPrinter`. If $facetSim$ returns a value greater than 0.8, the Web service is also published onto the private registry owned by the software company.

5 Related work

Our proposal can easily be compared with two wide classes of approaches: those that concentrate on service publication and discovery and those that deal with term similarity.

As for the first group, Garofalakis et al. in [1] introduce an overview of current Web service publication and discovery mechanisms and also propose a categorization. Registry technologies support the cooperation among registries, but they imply that all registries comply with a single standard and the cooperation needs a set up phase to manually define the information contributed by each registry. For example, UDDI v.3 [18] extends the replication and distribution mechanisms offered by the previous versions to support complex and hierarchical topologies of registries. It also identifies services by means of a unique key over different registries. The standard only says that different registries can interoperate, but the actual interaction policies must be defined by the developers. In our approach, the role of the registries and the way in which they cooperate are clearly defined.

Similarly, ebXML [3] is a family of standards based on XML to provide an infrastructure to ease the online exchange of commercial information. ebXML fosters the cooperation among them by means of the idea that groups of registries share the same commercial interests or are located in the same domain, as the thematic buses do in our approach. One of such groups can then be seen as a single logical entity where all the elements are replicated on the different registries. With respect to our approach, service retrieval with ebXML registries results ineffective since users must browse pre-defined taxonomies or submit keywords to find the desired services.

METEOR-S [7] and PYRAMID-S [8] fall in the family of semantic-aware approaches for the creation of scalable peer-to-peer infrastructures for the publication and discovery of services. These works create a federation of registries using several concrete nodes. Conversely to our approach, the single node become simply a gateway to the logical registry, ensuring higher availability or better response time, but loosing its identity. In particular, the usage of a semantic infrastructure allows for the implementation of different algorithms for the publication and discovery of services, but it also forbids the complete control over the registries. The semantic layer imposes too heavy constraints on publication policies and also on the way federations can evolve dynamically. METEOR-S only supports UDDI registers, while PYRAMID-S supports both UDDI and ebXML registries. They adopt ontology-based meta-information to allow a set of registries to be federated with each registry "specialized" according to one or more categories it is associated with. This means that the publication of a new service requires the meta-information needed to categorize the service within the ontology. Services are discovered by means of semantic templates that give an abstract characterization of the service and are used to query the ontology and identify the registries that contain significant information.

Term similarity has been tackled in several different ways [17]. These algorithms usually calculate such a similarity by relying on the relationships between terms defined in a reference ontology (e.g., *is-a*, *part-of*, *attribute-of*). In contrast, we compute similarity between terms according to the approach proposed by Seco et al. [19], where the authors adapt existing approaches with the assumption that concepts with many hyponyms convey less information than concepts that have less hyponyms or any at all (i.e, they are leaves in the ontology).

About the similarity between whole signatures, our approach is closely related to the approaches studied for the retrieval of reusable components [20]. In this field, as stated by Zaremski and Wing, there are two types of methods to address this problem: signature matching [21] and specification matching [22]. In particular, signature matching considers two levels of similarity and introduces the *exact* and *relaxed* matching of signatures. As for services, Stroulia and Wang [23] propose an approach that also considers the description field usually included in WSDL specifications.

6 Conclusions and future work

The paper presents an innovative infrastructure for the distributed publication of Web services and for their easy retrieval. The proposal leverages previous experiences of the authors, namely DIRE and URBE, and also other initiatives (USQL) to provide a holistic solution able to govern the replication of service information by means of user requests and preferences, and also able to provide users with partial, but acceptable, solutions whose fitness is defined through semantic match making techniques. The overall framework provides users with a wide set of options.

The integrated infrastructure exists as a very first prototype, but more stable solutions are needed for its deployment in realistic settings and also for a thorough empirical evaluation of the approach. Both these directions will govern our future work. The plan is to keep working on a fully functional prototype implementation and design a complete empirical evaluation of the proposal by exploiting a distributed set of registries and the usual collections of public Web services as benchmarks. Such a new prototype will also deal with query optimization and filter maintenance.

Acknowledgments

This work has been supported by the following projects: Tekne (Italian FIRB), Discorso (Italian FAR), SeCSE (EC IP), and ArtDecò (Italian FIRB).

References

1. Garofalakis, J., Panagis, Y., Sakkopoulos, E., Tsakalidis, A.: Contemporary Web service discovery mechanisms. Journal of Web Engineering **5**(3) (2006) 265–290
2. UDDI: Universal Description, Discovery, and Integration. (`http://uddi.xml.org`)
3. ebXML: Electronic Business using eXtensible Markup Language. (`http://www.ebxml.org/`)
4. Martin D. et al. (ed.): OWL-S: Semantic Markup for Web Services. W3C Submission. `http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/` (2004)
5. WSMO Working Group: Web Service Modeling Ontology. (`http://www.wsmo.org`)
6. Farrel, J., Lausen, H.: Semantic annotations for WSDL and XML schema. `http://www.w3.org/TR/sawsdl/` (2007)
7. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services. In: Information Technology and Management. Volume 6. (Jan 2005) 17 – 39
8. Pilioura, T., Kapos, G., Tsalgatidou, A.: PYRAMID-S: A scalable infrastructure for semantic web services publication and discovery. In: RIDE-DGS 2004 14th Int'l Workshop on Research Issues on Data Engineering, in conjunction with the IEEE Conf. on Data Engineering (ICDE 2004). (March 2004)
9. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish / subscribe. ACM Comput. Surveys **35(2)** (2003) 114 - 131
10. Tsalgatidou, A., Pantazoglou, M., Athanasopoulos, G.: Specification of the Unified Service Query Language (USQL). Technical report, (June 2006)
11. Baresi, L., Miraz, M.: A distributed approach for the federation of heterogeneous registries. In: Proc. 4th International Conference Service-Oriented Computing, Chicago, IL, USA, December 4-7. (2006) 240 – 251, Lecture Notes in Computer Science (vol. 4294), Springer Verlag.
12. Sawyer, P.: Specification language definition. Technical Report A1.D2.3, EC SeCSE Project (2006)
13. Plebani, P., Pernici, B.: Web service retrieval based on signatures and annotations. Technical Report 2007.47, Dipartimento di Elettronica ed Informazione - Politecnico di Milano (2007)

14. Najmi (ed.), F.: Java API for XML Registries (JAXR). `http://java.sun.com/webservices/jaxr/` (2002)
15. Cugola, G., Picco, G.P.: REDS: a reconfigurable dispatching system. In: Proc. of the 6th international workshop on Software engineering and middleware. (2006) 9–16
16. Bianchini, D., De Antonellis, V., Pernici, B., Plebani, P.: Ontology-based methodology for e-service discovery. Information Systems **31**(4-5) (2006) 361–380
17. Pedersen, T., Patwardhan, S., Michelizzi, J.: WordNet::Similarity - measuring the relatedness of concepts. In: Proc. National Conf. on Artificial Intelligence, July 25-29, San Jose, California, USA. (2004) 1024–1025
18. Clement, L., Hately, A., von Riegen, C., (eds.), T.R.: Universal Description, Discovery and Integration version 3.0.2. `http://uddi.org/pubs/uddi_v3.htm` (2004)
19. Seco, N., Veale, T., Hayes, J.: An intrinsic information content metric for semantic similarity in Wordnet. In: Proc. Eureopean Conf. on Artificial Intelligence (ECAI'04), Valencia, Spain, August 22-27, IOS Press (2004) 1089–1090
20. Damiani, E., Fugini, M.G., Bellettini, C.: A hierarchy-aware approach to faceted classification of objected-oriented components. ACM Trans. Softw. Eng. Methodol. **8**(3) (1999) 215–262
21. Zaremski, A., Wing, J.: Signature matching: a tool for using software libraries. ACM Trans. Softw. Eng. Methodol. **4**(2) (1995) 146–170
22. Zaremski, A., Wing, J.: Specification matching of software components. ACM Trans. Softw. Eng. Methodol. **6**(4) (1997) 333–369
23. Stroulia, E., Wang, Y.: Structural and semantic matching for assessing Web-service similarity. Int'l J. Cooperative Inf. Syst. **14**(4) (2005) 407–438