Noname manuscript No.
(will be inserted by the editor)

A Distributed Architecture for Efficient Web Service Discovery

Luciano Baresi · Matteo Miraz · Pierluigi Plebani

the date of receipt and acceptance should be inserted later

Abstract Although the definition of Service Oriented Architecture (SOA) included the presence of a service registry from the beginning, the first implementations (e.g., UDDI) did not really succeed mainly because of security and governance issues. This article tackles the problem by introducing DREAM (Distributed Registry by ExAMple): a publish / subscribe based solution to integrate existing, different registries, along with a match-making approach to ease the publication and retrieval of services.

DREAM fosters the interoperability among registry technologies and supports UDDI, ebXML Registry, and other registries. The publish / subscribe paradigm allows service providers to decide the services they want to publish, and requestors to be informed of the services that satisfy their interests. As for the matchmaking, DREAM supports different ways to evaluate the matching between published and required services. Besides presenting the architecture of DREAM and the different match making opportunities, the article also describes the experiments conducted to evaluate proposed solutions.

Keywords Distributed Service Registries · Service Discovery · Service Matchmaking · Similarity measure

Luciano Baresi, Matteo Miraz, and Pierluigi Plebani
Dipartimento di Elettronica, Informazione e Bioingegneria - Politecnico di Milano
Piazza Leonardo da Vinci, 32 - 20133 Milano, Italy
E-mail: [luciano.baresi, pierluigi.plebani]@polimi.it

1 Introduction

The development of software systems more and more often relies on the principles of Service Oriented Computing to improve the flexibility and interoperability of resulting solutions [1]. These systems open their boundaries by providing accessible programmatic interfaces (services) that ease the integration with other systems. In some cases, this openness is limited to a well-defined set of possible users (usually in business-to-business scenarios); in some other cases, these services can be used by anyone. As a result of this process, lots of services are available on the Internet:for instance, Programmable Web[1] has more than 12,000 registered APIs, and we have no figures about all those services that are not publicly available.

This considerable amount of services imposes suitable discovery and selection capabilities to allow for the identification of the services of interest. During the early days of SOA (Service Oriented Architectures), UDDI (Universal Description Discovery and Integration) was conceived to support the discovery of services, and some public UDDI repositories were created to host their descriptions [2]. These public registries did not work. Besides some security issues, the high amount of information required to register a service, and the absence of control led to incomplete and buggy service descriptions. The management of these registries was not easy because of the number of involved parties and the distributed ownership of services. The result was that nobody checked the sta-

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

tus of these services, and when these registries were closed, they were full of non-working services and incomplete information [3]. Public initiatives were substituted by private UDDI registries and domain-specific ebXML (e-business XML) registries [4], implemented in closed environments where the number of involved actors is limited and suitable governance can be applied.

As evidenced by the continuous effort done to offer private registry implementations in commercial platforms (e.g., Oracle and BEA AquaLogic Service Registries) and to standardize the implementation and the access to such registries (e.g., ebXML RegRep 4.0 specification has been approved as OASIS standard [2]), more and more private service registries are and will be made available soon.

The increasing number of available services, the evolution towards more private, controlled solutions, and the advent of cloud infrastructures let us foresee a service ecosystem where service descriptions are hosted on a set of diverse registries: some of them will be freely accessible while others will be more controlled. DREAM (Distributed Registry by ExAMple) is the contribution of this article to the ecosystem. DREAM is a solution for interconnecting heterogeneous registries and for easing the discovery of services. This solution originates from previous work by the authors: *DiRe* (DIstributed REgistry, [5]) and *Urbe* (UDDI Registry By Example [6]). The former contributes the communication framework among registries and the facet-based [7] description of services; the latter provides the match-making capabilities and semantic awareness. The integration of the two proposals defines a semantically-enabled replication infrastructure that supports different registry technologies (UDDI, ebXML, and others [3]).

A shared service description model provides the basis for the homogenous distribution and retrieval of information about services. A publish / subscribe middleware allows DREAM to let different parties share services and, at the same time, be informed about new, useful services. Each registry can decide the services it wants to publish, that is, the services it wants to share with others. Similarly, it can declare its interests —the services it would like to be aware of— 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 party can accommodate and tailor its interests (i.e., those of its users) while in operation. The approach introduced in this paper improves and validates the solution discussed in [8].

The discovery mechanisms implemented in DREAM support different granularities: service requestors can express their interests by referring to any element from complete facets to single attributes. In addition, the match-maker exploits reference ontologies to evaluate the similarity between the terms used in the requests and in available service descriptions and return service that match requests at different degrees of precision.

To summarize, the key and novel contribution of this paper is a distributed framework enabling the interoperability among different service registries and different service description models. This allows the service requestor to express his/her preferences using the preferred service description model, as the interoperability among the different service registries is ensured by an abstract service model composed of facets. As demonstrated by a series of experiment, this integration, along with the possibility of integrating semantic-enabled service discovery mechanisms, increases the effectiveness of the service retrieval in terms of precision and recall.

Moreover, the proposed approach contributes to make service publication and retrieval easier and more flexible as DREAM does not rely on a single standard for describing provided services and for defining queries. Currently, from the service provider perspective, DREAM is able to support any kind of service description, i.e., WSDL, SAWSDL, OWL-S, with the only limitation that the language has to be XML-based. From the service requestor perspective, the service retrieval supports queries expressed using XPath, XPath with an additional operator (proposed in this work), and WSDL documents used to specify the requestor needs. Although DREAM is open to be extended with additional query languages, developers of such

[2] `https://lists.oasis-open.org/archives/regrep/201201/msg00011.html`

[3] `http://www.secse-project.eu/`

extensions are in charge of implementing the related match-making algorithms.

The rest of the paper is organized as follows. Section 2 discusses the motivations behind a distributed architecture for service registries and how semantic analysis can improve the precision and recall of service retrieval. Section 3 introduces the proposed architecture and explains how it can be used by both service providers and requestors. Section 4 presents the different match-making capabilities offered by DREAM. Section 5 describes the experiments conducted to evaluate the performance of DREAM in terms of response time, precision, and recall. Section 6 surveys related approaches and Section 7 concludes the paper.

2 Motivations

As discussed in the introduction, the idea of having a worldwide service registry failed after a while as the governance of this kind of solutions is hard to manage. To overcome this problem, the SOA community developed alternative approaches. On the one side, they proposed a more agile publication process. For instance, Web sites like XMethods[4] or ProgrammableWeb index services by only considering a description and tags freely assigned by the developers. The resulting tag-based retrieval, as usually occurs with keyword-based information retrieval mechanisms, is not effective. On the other side, the classical solutions, like UDDI and ebXML Registry, are still used but either in a more controlled context, for example, inside a company, or by focusing on a specific type of services (e.g., geospatial services, healthcare).

Although these new approaches eased the publication, they had a negative impact on the retrieval. Indeed, reducing the information required when a new service is made available also means reducing the information that can be used to retrieve it. The use of a private service registry implies that all the indexed services can only be viewed by a limited set of users. For this reason, companies maintain two registries: (i) a public one, which is freely accessible and contains all the services that can be invoked externally, and (ii) a private one, with control policies that restrict the access to the services that are only available internally.

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

Finally, despite the initial intention of having a standard registry technology, that is, UDDI, interoperability is now an issue. Different registries, with different communication protocols are available and can be required to exchange information and cooperate.

The lack of a winning solution pushed us to concentrate on the distributed publication of services as a means to improve both exposition and retrieval. These are the resulting requirements for a service registry architecture:

- Distribution: services can be published on different registries managed by different subjects.
- Interoperability: service registries can be based on different technologies and communicate using different protocols.
- Controlled publication: the service provider can decide the visibility of the published services (e.g., private and public) no matter the registry on which they are published.
- Customized retrieval: the service request can be formalized in different ways and all the service registries must be queried against such a request.
- Scalability: services are continuously published and removed and the number of services to be stored can be significantly high and not foreseen at design-time.

Figure 1 shows our vision with respect to the publication of services (on the left) and their retrieval (on the right). When a provider develops a service, this is published on the internal (proprietary) registry, and it can be declared as either a private or public service. In the former case, the service description is only stored in the internal registry and it is not accessible outside. In the latter case, the service is automatically published on the external registries that manifested interest on it. For instance, a repository that hosts services related to books can be interested in services that provide information on books, reviews, and stores. Any service about this topic would be stored in the registry. The communication infrastructure provided by DREAM ensures a seamless integration between the internal registry and the public ones is provided even if they are based on different technologies.

As for service retrieval, we assume the presence of two types of service requestors: the typical user, who browses a service directory looking for a particular service, and the registry

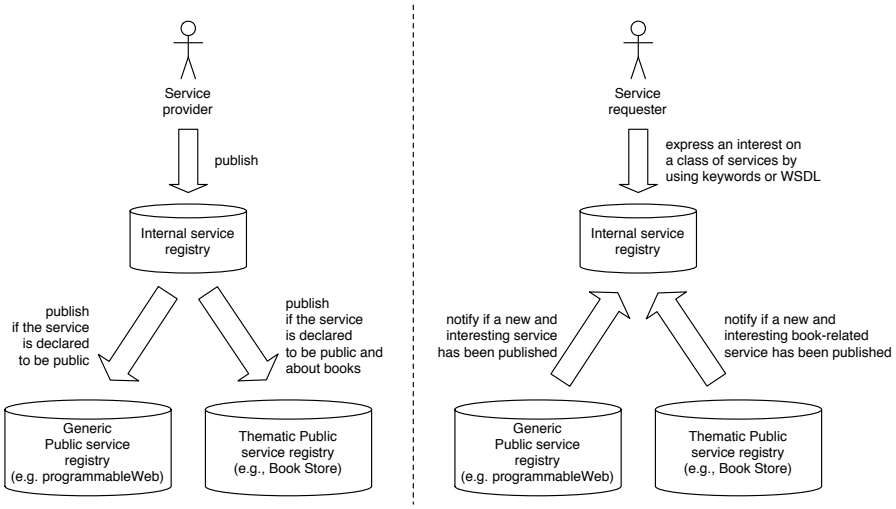


Fig. 1 Example scenario.

maintainer, who wants to be aware of as many services as possible. These two types of users might have different skills and thus require different methods to define their interests. For example, a non tech-savvy requestor can express an interest by means of keywords (e.g., "book price"), while a more skilled user may submit a WSDL to specify the interface that the requested service has to expose. The advantage of DREAM is that it supports both types of service requestors. As DREAM enables the integration of different service description models and different service retrieval mechanisms, it is open to support different types of queries. Service requestors can express their query according to their skills, without adopting any new standard.

Finally, the communication between the service registries and the service requestor is mainly based on a publish / subscribe middleware: a solution that can ensure good performance even in case of a high number of published services and service requests.

3 DREAM Architecture

To satisfy the architectural requirements introduced in the previous section, this paper proposes a distributed service registry based on a common service description model and a publish-subscribe middleware. Each node of the infrastructure can be used by both providers to publish their services, and by users who want to find the services able to match their needs. As a consequence, the goal of the middleware is twofold: it collects the requests and finds the services able to satisfy them. When a new service is published, its description must be sent to all the nodes interested in it.

From a technical standpoint, Figure 2 shows that DREAM provides a *communication infrastructure* (composed of *delivery manager*s and *dispatcher*s) and a *similarity engine*. Each delivery manager collects the information about new requests and new available services in a node and the *dispatcher* distributes that information to the interested parties. Since such information should only be exchanged if a party is interested in that particular new service, the similarity engine is used to match the published service description to the different requests.

3.1 Service Description Model

A user can choose the implementation of the *Registry* that fits best its requirements. These implementations mainly adhere to two main specifications: *UDDI* [9] and *ebXML* [4], but they could also be proprietary solutions. Sadly, they have a different data model, and thus they are not compatible. The heterogeneity of registries and the need for a flexible approach that fits most of the user scenarios suggested us to develop a new, lightweight model to describe services. The goal of this service description model is to become a neutral language that is easily mappable onto the different registry models. To this end, the service description model uses a faceted approach: each service is described

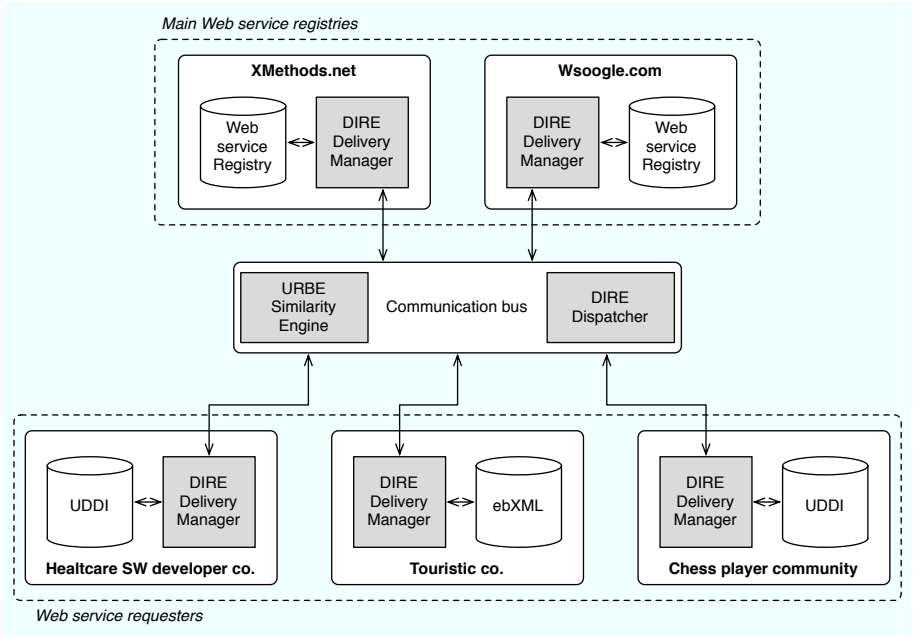


Fig. 2 DREAM architecture.

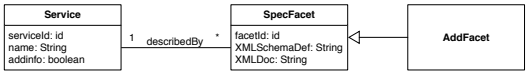


Fig. 3 Service model using facets.

by means of facets, each addressing a different characteristic of the service or point of view.

A `Facet` is the key constituent of the proposed model (see Figure 3). It describes a web service from a particular point of view. For example, a facet may characterize a service from a functional point of view and describe the operations it provides —by means of a WSDL document, or it can describe the quality of services guaranteed by the service and detail its availability, reputation, and response time. The model only requires that facets be self-contained, and the information they provide rendered in XML. The first requirement ensures that once a subject retrieves a facet, there is no need to retrieve additional documents to get the information it contains. The latter eases document management and is justified by the wide adoption of XML languages by the service community. As usual, XML schema documents characterize the facets with the type of information they contain. For example, *WSDL* facets describe the interface of services, *RDF* facets add semantics, and XMI facets specify complex service behaviors by means of *UML* diagrams. Additionally, each *Facet* has a unique identifier, used to ease its management. Hereafter, $\{f_j^{w_i}\}$ identifies the facets of a Web service w_i.

Each `Service` comprises a set of specification facets (`SpecFacet`). The service provider is in charge of writing these facets, and they contain properties that it guarantees for its service. For example, a service can be described through facets that specify its interface, the company that provides it, and the guaranteed qualities of service. Users can reassemble the complete specification of a service by collecting and analyzing all its specification facets. For example, Figure 4 shows the excerpts of two possible facets associated with the book example. We assume that the service providers define for the same service two different WSDL and OWL-S description, respectively.

The users of a service may also create some additional facets (`AddFacet`) and describe the service from their point of view. For example, users can describe a service by specifying its rating, its level of customization, and the measured quality of service. These facets do not specify the service, but they contain information that may be useful in the service-selection phase to rank retrieved services.

In addition, the user can also rely on existing standards and frameworks like UDDI and ebXML and can use parts of that specifications to proper identify the services in DREAM. For example, possible usages could be to create additional facets that are linked to a `tModel` or

WSDL facet: **book_price_service.wsdl**

```
<wsdl:definitions ...>
  <wsdl:types>
    ...
  </wsdl:types>
  <wsdl:message name="get_PRICEResponse">
    <wsdl:part name="_PRICE" type="tns:PriceType" />
  </wsdl:message>
  <wsdl:message name="get_PRICERequest">
    <wsdl:part name="_BOOK" type="tns:BookType" />
  </wsdl:message>
  <wsdl:portType name="BookPriceSoap">
    <wsdl:operation name="get_PRICE">
      <wsdl:input message="tns:get_PRICERequest" />
      <wsdl:output message="tns:get_PRICEResponse" />
    </wsdl:operation>
  </wsdl:portType>
...
</wsdl:definitions>
```

OWL-S facet: **book_price_service.owls**

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<rdf ...>
  <service:Service rdf:ID="BOOK_PRICE_SERVICE">...</service:Service>
  <profile:Profile rdf:ID="BOOK_PRICE_PROFILE">
    <service:isPresentedBy rdf:resource="#BOOK_PRICE_SERVICE"/>
    <profile:serviceName xml:lang="en">BookPriceService</profile:serviceName>
    <profile:hasInput  rdf:resource="#_BOOK"/>
    <profile:hasOutput rdf:resource="#_PRICE"/>
    ...
  </profile:Profile>
  <process:Input rdf:ID="_BOOK">
    <process:parameterType>books.owl#Book"</process:parameterType>
  </process:Input>
  <process:Output  rdf:ID="_PRICE">
    <process:parameterType>concept.owl#Price</process:parameterType>
  </process:Output>
  <grounding:WsdlGrounding rdf:ID="BOOK_PRICE_GROUNDING">
    <service:supportedBy rdf:resource="#BOOK_PRICE_SERVICE"/>
  </grounding:WsdlGrounding>
</rdf:RDF>
```

Fig. 4 Example of facets.

to a `categoryBag`, in case of UDDI, or to a `Collaborative Partner Profile`, in case of ebXML.

At this stage, DREAM implements facets linked to WSDL descriptions, and in the rest of the paper we discuss how DREAM can support different kinds of service requestors expressing different kinds of queries. Future releases will include other types of services description models, i.e., OWL-S or REST-based, as well as facets expressing quality of service capabilities.

3.2 Communication infrastructure

Differently from other approaches such as METEOR-S [10] and PYRAMID-S [11], which create a single logical registry spread among several physical nodes, we propose a really distributed registry. DREAM exploits the inherent distribution of registries to provide the user with a finer control over published information. DREAM assumes that each party manages a *private registry*. This registry manages information regarding the services being used. Being the registry private to the corporation, it glues together the different parts of its IT infrastructure by allowing a blackboard communication style between the various components. The registry may be used from parts of system to retrieve, add, or modify information on services. For example, a component may monitor the execution of services, measure the average response time, and store it in the registry as additional facets. Another component, which retrieves services, may use that information to select the fastest service for a given task.

The party that manages a registry has the full control on the information published on the registry. This means that it can perform a preliminary selection of the services in the registry, and ensure that it only contains services that are of interest. Each query performed on the registry works on pre-approved services. Accordingly, the results of these queries have a high precision (i.e., almost only relevant services are found), at the price of a lower recall (i.e., not all relevant services may be found).

DREAM improves the recall by introducing a *marketplace* mechanism to exchange service facets among registries. Like "real" marketplaces, DREAM allows a loosely-coupled cooperation among service providers and potential customers. On one side, service providers are allowed to *share* their services and to broadcast their descriptions. On the other side, DREAM provides clients with the ability to analyze shared facets (i.e., service descriptions). The client can decide whether a service meets its requirements and, if it is the case, acquire the relevant facets and insert them in its registry.

To support this cooperation style, DREAM introduces a global *communication bus* and a *delivery manager* to connect each registry (see Figure 2) to it. The *communication bus* acts as a common reference to all the delivery managers, and allows them to efficiently exchange messages in a peer-to-peer manner. The main element of the communication bus is the *dispatcher*, which follows the publish / subscribe paradigm. When a node wants to deliver a message, it contacts the dispatcher and *publish*es the message. Conversely, *subscription*s allow nodes to declare what messages are relevant for them. The dispatcher forwards messages being published to nodes with a proper subscription.

The core of this communication system is based on REDS [12], a distributed publish / subscribe system [13]. REDS splits the dispatcher

among several nodes, and guarantees logical integrity. Consequently, it is able to create a scalable infrastructure that can manage very large networks. Moreover, REDS is able to adjust its internal structure, react to node failures, optimize its performance, and ensure a reliable and efficient communication system.

The *delivery manager* acts as *façade* for the registry. It allows the party to both publicize and discover services, by managing the information flow from the local registry to the other registries and vice-versa. For this purpose, the delivery manager is able to perform the adequate conversions between the faceted service model and the one used by the particular registry.

When it is asked to publish the information about a service, the delivery manager accesses the local registry, fetches the information on the service, converts it into the proper facets, and delivers them as a message to the bus. The dispatcher forwards that message to all the interested parties by using a best effort delivery. In fact, the bus operates on an unreliable network: nodes can crash, have temporary failures, and the whole network may have problems.

Additionally, parties may join the marketplace after the information on a service has already been shared. To solve these issues, DREAM uses the *lease* mechanism, which guarantees a global consistency even if some messages are lost. The lease mechanism is typical of many distributed systems (e.g., Jini [14]) and requires that each sent message (information) has an expiration date. When a message expires, the information is not considered valid anymore, and it can be deleted, unless a *renew* request is sent. These renews ensure that all customers receive the information, even if they enter the system after the first distribution. Moreover, since messages are retransmitted more often than they expire, the infrastructure can tolerate a certain amount of lost messages. The *delivery manager* automatically performs this operation and renews the information about published services.

The delivery manager can also help who wants to discover new services. The party must specify the *interest*, that is the query to be used to find the new services. DREAM allows interests to analyze the content of facets, and supports different match-making solutions: `XPath`, `R-XPath`, and `WSDL-based` (see Section 4). The delivery manager uses the interests created by the party to perform subscriptions on the communication bus. The dispatcher will then forward to the delivery manager all the messages whose content matches the interests. As soon as these messages are received, the delivery manager converts them into the local format, and inserts them in the limbo zone of the local registry.

If one wanted to extend the set of matchmakers, the key element is interface *Interest* which declaration follows:

```
interface Interest {

    public boolean matches(Deliverable msg);
}
```

To introduce a new matchmaker, one must implement such an interface by implementing method *matches*. This method receives a *Deliverable* object as input that contains the references to the facets that must be exploited to answer the query. DREAM comes with three implementations of this interface for the matchmakers that are discussed in the next section.

4 Match-making

As previously discussed, one of the main goals of DREAM is to provide a flexible way for retrieving services by allowing users to submit their requests in different ways. Since all services are described through facets, the request (a.k.a *interest*) can be expressed in three different ways. They differ in the way the query is formulated and in the accuracy provided by the similarity engine. One can:

- Use an *XPath* expression:
  $mmX(xpath, \{f_j^{w_i}\})$.
  This is to state that one or more keywords have to *exactly match* a given element in the service description. Since users could be unable to write XPath expressions, we assume they may use directories of pre-selected XPaths, or tools for translating keywords into them.
- Adopt a relaxed XPath (*R-XPath*):
  $mmR(xpath, \{f_j^{w_i}\})$.
  It is similar to the previous XPath-based approach, but it also allows for a *relaxed match* between the terms included in the expression and the service description. It means that the match is satisfied when the terms,

even if they are not equal, are however connected in a reference ontology.
- Exploit any facet of the service description and compare its terms:
  $mmW(facet, f^{w_i}_{facet})$.
  For example, given a *WSDL* description of the desired Web service, the match-making would compare the operations, messages, and parameters as defined in the published WSDL facet.

Given this flexibility, DREAM can deal with different scenarios. For instance, one can assume that a user looks for a service to buy books on line (see Figure 5). The user's interests can be defined by an XPath expression stating that `buy` is the name of the portType and `getBook` is the name of the operation, and these two names must be included in the service description [5]. Other users can be less restrictive on names and also be interested in services that use similar terms like, for instance, `purchase` and `getPaperback`. Experienced users, like programmers, may also provide a WSDL to specify the interface of the service they would like. This situation is also similar to when a service already used by an application becomes unavailable and it must be replaced: the closer the new service interface is, the lesser the work needed to adjust the client is.

4.1 XPath-based match-making

The first approach leverages the `XPath` language to inspect facets and determine whether they contain valuable information. The use of XPath fits perfectly the service model proposed in Section 3.1, where facets are defined as XML documents.

Queries expressed by using the XPath-based match-making require the *XML schema* of the facet they target and the *XPath* expression that states the properties of interest. As for XPath expressions that concern name matching, function $mmX(xpath, \{f^{w_i}_j\})$ returns true if the XPath expression is satisfied. For example, if one considers the example of Figure 5, and assumes the availability of a WSDL facet (whose namespace is `wsdl`), the query corresponds to the following XPath expression:

$xpath =$
```
//wdsl:portType[@name=''buy'']
&& //wdsl:operation[@name=''getBook'']
```

$mmX(xpath, f^{w_i}_{wsdl})$ returns true only if the `portType` attribute is equal to `buy` and the name of the operation is `getBook`. Note that, along with the name matching evaluation, the user can exploit the full power of XPath to specify more complex conditions.

4.2 R-XPath-based match-making

The second approach extends the previous one by allowing for a more flexible comparison: a service is considered to be relevant even if its description does not exactly correspond to the terms specified in the query. To do this, we evaluate the similarities between words by means of function $wSim(w_1, w_2) \rightarrow [0, 1]$, where w_1 and w_2 are the two words to be compared, and the higher the result is, the more similar the two words are.

Before discussing how $wSim$ works, we introduce the bipartite graph assignment problem since it provides the basis for our similarity function. Given a graph $G = (V, E)$, where V is the set of vertexes and E the set of edges, $M \subseteq E$ is a matching on G iff no two edges in M share a common vertex. If M covers all the nodes of the graph, G is bipartite. This also means that each node of the graph has an incident edge in M. Let us suppose that the set of vertices are partitioned in two sets Q and P, and that the edges of the graph are weighted according to function $f : (Q, P) \rightarrow [0..1]$. The function $maxSim : (f, Q, P) \rightarrow [0..1]$ returns the maximum weighted assignment, that is, an assignment such that the average of the weights of the edges is maximum. Fig. 6 shows a graphical representation of the problem, where the bold lines constitute the matching M.

If we expressed the assignment in bipartite graphs according to a linear programming model, we would have:

$$maxSim(f, Q, P) = max \sum_{i \in I}^{j \in J} f(q_i, p_j) \cdot x_{i,j}$$

$$\sum_{j \in J} x_{i,j} \leq 1 \quad \forall i \in I$$

$$\sum_{i \in I} x_{i,j} \leq 1 \quad \forall j \in J$$

[5] For the sake of simplicity, in the example, we use only the WSDL facets, even if it is possible to apply the XPath and R-XPath matchmaking to any XML-based document.

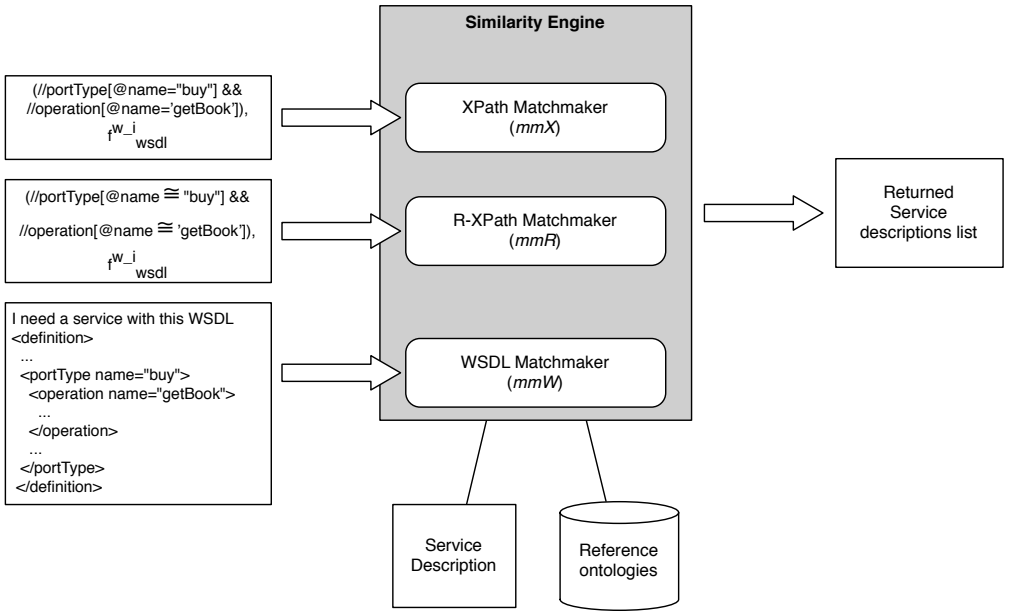


Fig. 5 Query types in DREAM.

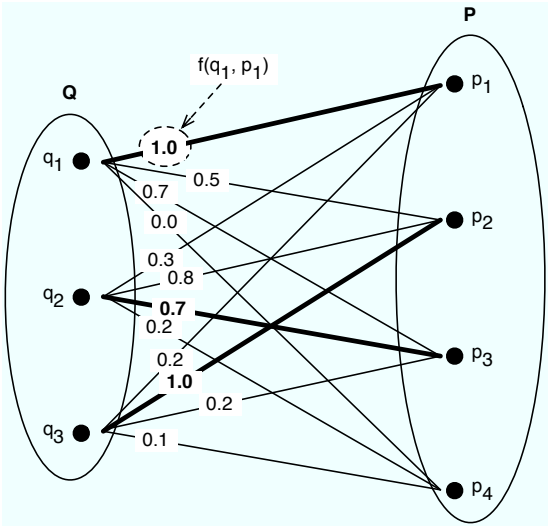


Fig. 6 Graphical representation of the assignment in bipartite graphs problem.

$$I = [1..|Q|], \quad J = [1..|P|] \tag{1}$$

□

The similarity between two words w_1 and w_2 is computed in three steps: tokenization, stemming, and distance evaluation. Tokenization decomposes w_1 and w_2 in two bags of terms: $w_1 = \{t_{1,i}\}$ and $w_2 = \{t_{2,i}\}$. It considers case changes, underscores, hyphenations, and numbers. The terms resulting from the tokenization are also stemmed (second step). Thus, words like *sending* or *exchanged* are transformed into their stemmed version: *send* and *exchange*. The stemming process is a well-known process adopted by several information retrieval approaches [15] [6]. The third step is about the computation of the similarity $wSim$, obtained by exploiting the assignment problem in bipartite graphs:

$$wSim(w_1, w_2) = \frac{maxSim(termSim, t_{1,i}, t_{2,i})}{|t_{1,i}|} \tag{2}$$

The inputs are the two sets of tokens $\{t_{1,i}\}$ and $\{t_{2,i}\}$ that compose the two words to be compared, and function

$$termSim : (term', term'') \rightarrow [0..1]$$

that returns the similarity of two tokens. This way, $wSim$ returns the word similarity as the sum of the similarities between the pairs of tokens that maximize such a sum. On this basis, $termSim$ holds a central role in the computation of the similarity.

The literature proposes several approaches to state the similarity and relatedness between terms [16]. These algorithms usually compute such a similarity by relying on the relationships among terms defined in a reference ontology (e.g., *is-a*, *part-of*, *attribute-of*). Our approach computes the similarity between terms by adopting the solution proposed by Seco et al. [17]: they rely on the assumption that concepts with

[6] For the sake of simplicity, we use the terms t_i for both the original and stemmed versions.

many hyponyms [7] convey less information than concepts that have less hyponyms or any at all (i.e, they are leaves in the ontology).

Note that $wSim$ returns the maximum sum divided by the number of terms composing w_1. Indeed, in the application of the assignment problem in bipartite graphs to our context, set t_1 represents a query, whereas t_2 is what we compare against the query to evaluate the similarity. $|t_1| < |t_2|$ means that the number of elements required in the query t_1 is lower than the number of elements made available in t_2: for each element in t_1 we may find a corresponding element in t_2. In contrast, $|t_1| > |t_2|$ means that we are asking for more elements than those that are actually available. As a consequence, we consider that the situation in which $|t_1| < |t_2|$ is in general better than the case $|t_1| > |t_2|$. For this reason, we divide the result of the maximization by the cardinality of $|t_1|$. So, if $|t_1| < |t_2|$ then $wSim : (t_1, t_2) \rightarrow [0..1]$, whereas if $|t_2| < |t_1|$ then $wSim : (w_1, w_2) \rightarrow [0..\frac{|t_2|}{|t_1|}]$. This way function $wSim$ is asymmetric, that is, $wSim(w_1, w_2) \neq wSim(w_2, w_1)$. If all the tokens composing w_1 has a correspondence with one token in w_2, then the similarity will be higher than in the case in which some "requested" token is not associated with any token on the other side.

We assume the presence of both *domain specific* and *general purpose* ontologies. The former include terms related to a given application domain, and can be built by a domain expert, for example, by analyzing the terms included in the Web services published in the registry. The latter include all the possible terms —and we adopt Wordnet. We decided to rely on both types of ontologies since the domain specific ontology offers more accuracy in the relationships between terms, while the general purpose one offers wider coverage. This happens because in a general purpose ontology a word may have different meanings, and thus different sets of synonyms (*synsets*) in different contexts. In contrast, we assume that in a domain specific ontology, each word has a unique meaning with respect to the domain itself. For instance, *currency* has two synsets in WordNet. The first is about the financial domain, that is, the system of money used in a country; the second is about the fact of being generally accepted. This means that if we compared the terms *currency* and *money* [8], we could realize that they are strictly related only if we consider the financial domain. On the other hand, if we considered the other synset, the relationship would be looser. Therefore, in case of general purpose ontologies, it is hard to identify the correct domain to consider: our solution is to use the average similarity evaluated over the different synsets.

According to the definition of $wSim$, the match-making function for the relaxed XPath $mmR(xpath, \{f_j^{w_i}\}) \rightarrow [true; false]$ is defined by starting from the previously defined mmX, where the similarity operator $\cong$ can be used in the XPath expression. In this case $(A \cong B = true) \Leftrightarrow w(A, B) > th_r$. Having the $\cong$ operator, the user can enrich the XPath expression with relaxed name matching, that is, the names included in the service descriptor not necessarily need to be equal to the names specified in the query. We assume that the threshold $th \in [0..1]$ is defined by the DREAM administrator after a training session as its value is critical for the reliability of the match-making function. Indeed, if the value is too low, the number of false-positive might increase. The number of false-negatives increases if the threshold has a too high value.

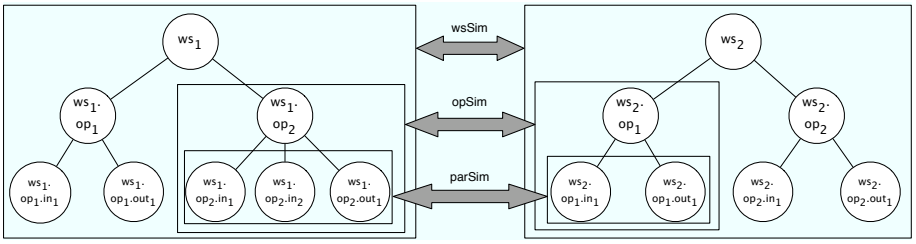
4.3 WSDL-based match-making

The third match-making algorithm considers only the WSDL facets and it is based on the functionality provided by *Urbe* (Uddi Registry By Example) [6] that evaluates the similarity between two WSDLs. *Urbe* proposes a match-making algorithm aimed to identify similar (substitute) services by analyzing the WSDL descriptions of the different services. Retrieved Web services must expose an interface that is equal to or richer than the required one. In particular, *Urbe* computes the similarity degree of two WDSL descriptions by computing the effort —in terms of changes to the code— requested to a client to use the service(s) retrieved by the system.

The algorithm computes the relationships between the main elements of the WSDL descriptions, that is, of their portTypes, messages,

[7] A hyponym is a word that conveys a more specific meaning than a general term applicable to it. For example, spoon is a hyponym of cutlery.

[8] see http://marimba.d.umn.edu/cgi-bin/similarity.cgi



$$\begin{aligned} wsSim(ws_1, ws_2) = {} & wPTNameSim \cdot wSim(ws_1.name, ws_2.name) + \\ & + (1 - wPTNameSim) \cdot \\ & \cdot \frac{1}{|ws_1.\{op_{k1}\}|} \cdot maxSim(opSim, ws_1.\{op_{k1}\}, ws_2.\{op_{k2}\}), \end{aligned} \tag{3}$$

where:

$$\begin{aligned} opSim(ws_1.op_{k1}, ws_2.op_{k2}) = {} & \\ & wOpNameSim \cdot wSim(ws_1.op_{k1}.name, ws_2.op_{k2}.name) + \\ & +(1 - wOpNameSim) \cdot \\ & \cdot[0.5 \cdot \frac{1}{|ws_1.\{op_{k1}.\{in_{l1}\}\}|} \cdot \\ & \quad maxSim(inParSim, ws_1.op_{k1}.\{in_{l1}\}, ws_2.op_{k2}.\{in_{l2}\}) \\ & +0.5 \cdot \frac{1}{|ws_2.\{op_{k2}.\{out_{m2}\}\}|} \cdot \\ & \quad maxSim(outParSim, ws_2.op_{k2}.\{out_{m2}\}, ws_1.op_{k1}.\{out_{m1}\})] \end{aligned} \tag{4}$$

and

$$\begin{aligned} & inParSim(ws_1.op_{k1}.in_{l1}, ws_2.op_{k2}.in_{l2}) = \\ & \quad wSim(ws_1.op_{k1}.in_{l1}.name, ws_2.op_{k2}.in_{l2}.name) \end{aligned} \tag{5}$$

$$\begin{aligned} & outParSim(ws_1.op_{k1}.out_{l1}, ws_2.op_{k2}.out_{l2}) = \\ & \quad wSim(ws_1.op_{k1}.out_{l1}.name, ws_2.op_{k2}.out_{l2}.name) \end{aligned} \tag{6}$$

Fig. 7 Structure of the service similarity function $wsSim$.

operations, and parameters. If available, semantic annotations associated with the candidate service as a SAWSDL (Semantic Annotated WSDL [18]) facet can be used to improve the retrieval process.

Since semantic annotations are rare, DREAM usually computes the similarity function $wsSim : (ws_1, ws_2) \rightarrow [0, 1]$ between two WSDL service descriptions and returns their similarity degree, where ws_1 represents the user query, whereas w_2 represents a service included in the registry that needs to be compared to the query. Also in this case, the higher the returned value is, the better the similarity between the services is. Function $wsSim$ considers the number of operations and parameters and the similarity between the names used for portTypes, operations, and parameters.

The hierarchical structure of a WSDL description impacts the structure of $wsSim$. More in detail, as reported in Figure 7, the similarity between two web service descriptions computed

by $wsSim$ depends on the similarity among their portTypes, as they represents the k_1, k_2 operations made available by the services. This similarity at portType level is computed by function $opSim$. In turn, the similarity between two portTypes depends on the similarity between the l_1, l_2 input and m_1, m_2 output parameters that characterize each operation, which is computed by using functions $inParSim$ and $outParSim$, respectively.

Equations 3-6 detail the structure of these four functions where the same pattern is adopted. On the one side, the name similarity $wSim$ is used to compare the names of analyzed elements (i.e., the service names for $wsSim$, the portType names for $opSim$, and the parameter names for $inParSim$ and $outParSim$). On the other side, as a service is composed of several portTypes that, in turn, are composed of several parameters, function $maxSim$ is used to identify the best matching between elements of the comparing services that maximizes the similarity value. Finally, the result of $maxSim$ is divided by the number of elements included in the service representing query ws_1 (i.e., the number of portTypes in ws_1 for $wsSim$, the number of parameters defining a portType belonging to ws_1 for $opSim$). This aspect introduces an asymmetry in the similarity function that is justified by the need for distinguishing between a query ws_1 asking for more elements than service ws_2 offers, and a query ws_1 asking for less elements than ws_2 offers. In the first case, the similarity will be lower as the query is not fully satisfied, whereas in the second case, even if the service can offer more than requested, the query is satisfied.

To balance the importance of these two aspects —while computing the overall similarity—, weights $wPTNameSim$, $wOpNameSim$ are introduced. More specifically, $wPTNameSim \in [0..1]$ defines the importance of the name of the portTypes with respect to the similarity between the operations these portTypes offer. Similarly, at operation level, parameter $wOpNameSim \in [0..1]$ weights the importance between the similarity of operation names and the similarity of related parameters.

The following properties hold for the similarity function $wsSim$:

- $wsSim(\sigma_i, \sigma_i) = 1$: a Web service is totally similar to itself;
- in general, $wsSim(\sigma_i, \sigma_j) \neq wsSim(\sigma_j, \sigma_i)$: the similarity depends on the Web service description used as query.

Based on this algorithm, function $mmW(wsdl, \{f_j^{w_i}\}) \rightarrow [true; false]$ returns $true$ if $wsSim(wsdl, f_{wsdl}^{w_i}) > th_w$. Similarly to the case of the relaxed X-Path, the threshold needs to be set by the administrator after a training session. In addition to that, for this similarity function, the administrator is also in charge of tuning the values of $wPTNameSim$ and $wOpNameSim$.

5 Validation

The efficiency and effectiveness of DREAM has been assessed through a set of experiments. In particular, we started from a set of queries issued at different nodes and a set of registries distributed over the network to evaluate (i) the efficiency by measuring the response time to return the result of the comparison, and (ii) the effectiveness by measuring the precision and recall of such a result.

The benchmark adopted for both tuning and evaluating the performance of the similarity algorithm has been obtained from the SAWSDL [18] service retrieval test collection (SAWSDL-TC1) [9]. SAWSDL semantically enriches the WSDL-based service definition by annotations that contain concepts organized in a reference ontology: the benchmark of WSDL services used for evaluating our approach is obtained by ignoring these annotations. More in detail, the benchmark consists of 894 Web services that cover different application domains: communication, economy, education, food, medical care, travel, and weaponry. The benchmark also includes 26 test queries, represented as SAWSDL documents; the list is reported as Appendix A.

To have a fair comparison among the three approaches, that is, XPath, R-XPath, and WSDL, we started from the queries suitable for the WSDL case and we derived those for the other two cases. Figure 8 shows how given a WSDL-based query (included in the benchmark), the related XPath expression requires that the names of portTypes and (input/output) messages be the same as those of the initial query. Yet, the R-XPath expression also indicates the similarity

[9] `http://projects.semwebcentral.org/projects/sawsdl-tc/`

XPATH query based on **book_price_service.wsdl**

```
operation@name = getPrice

AND

message/part@name= Book OR _Book

AND

message/part@name= Price OR _Price
```

R-XPATH query based on **book_price_service.wsdl**

```
operation@name = getPrice [relaxed=0.7]

AND

message/part@name= Book OR _Book  [relaxed=0.7]

AND

message/part@name= Price OR _Price  [relaxed=0.7]
```

WSDL query: **book_price_service.wsdl**

```
ws.name = BookPriceSoap
ws.op1    = {ws.op1.name = getPrice,
             ws.op1.in1 = { ws.op1.in1.name = _Book,
                            ws.op1.in1.type = tns:BookType},
             ws.op1.out1 = { ws.op1.out1.name = _Price,
                             ws.op1.out1.type = tns:PriceType}
           }
```

Fig. 8 Example of queries used for the assessment.

threshold that must be reached to obtain a positive match.

5.1 Effectiveness

To analyze the effectiveness of DREAM [19], we used *precision* and *recall* as performance indicators. Each test query is associated with a set of services that the proponents of the benchmark have defined as relevant. This means that given a query, the precision provides the ratio between the number of relevant Web services among those returned by DREAM, where the lower the precision is, the lower the number of false positives is. On the other side, the recall indicates the ratio between the number of relevant Web services returned by DREAM among those defined relevant. In this case, the higher the recall is, the lower the number of false negatives is. The total precision and recall have been computed as the average of the precision and recall of each of the 26 test queries. Note that precision and recall also indicate how DREAM can be beneficial to the user. Indeed, high precision indicates that all the returned services are likely what the user is expecting for. High recall indicates that DREAM returns a significant amount of services that are potentially interesting for the user.

As expected, Figure 9[10] shows that the $wsSim$ similarity algorithm provides the best trend, while the XPath-based similarity has a questionable behavior. Indeed, $wsSim$ deeply analyses all the elements of the WSDL description since the queries are richer than the XPath-based ones. Note that in this last case, a service is considered to be relevant only if the names match exactly.

Although the $wsSim$ algorithm provides the best precision-recall among the three, in the literature [20] there are other matchmaking algorithms that might perform better and, due to the flexibility of DREAM, they can be included in the architecture. Note that the precision and recall obtained by DREAM come from a system that integrates different models for describing a service and that supports different types of query mechanisms. For this reason, even if in the literature there are better approaches, they are specifically studied to support a particular service description model (e.g., WSDL, SAWSDL, or tag-based only). As a consequence, one should consider how to improve the precision and recall without affecting the flexibility of describing services and querying a registry using the languages one prefers.

5.2 Efficiency

To measure the efficiency of the three matchmaking methods, we created a "simple", distributed environment composed of three nodes. Each node used a 550 Mhz Intel Xeon E 5530 processor and 1.5 Gbyte of memory, running Linux. We used the *server* profile of the Oracle Java virtual machine, and the heap was limited to 1 Gbyte. The first node acted as service provider, and periodically published all the services in our benchmark. The second node acted as service consumer: it used the queries defined in our benchmark as subscriptions. The third node acted as broker, and connected the service provider and service consumer. The scalability of DREAM mainly depends on the ability of brokers to route service descriptions to interested nodes. For this reason, we focused on the third node and measured the match-making time.

[10] Precision and recall are calculated by using the SME2 Evaluation tool (`projects.semwebcentral.org/projects/sme2/`).

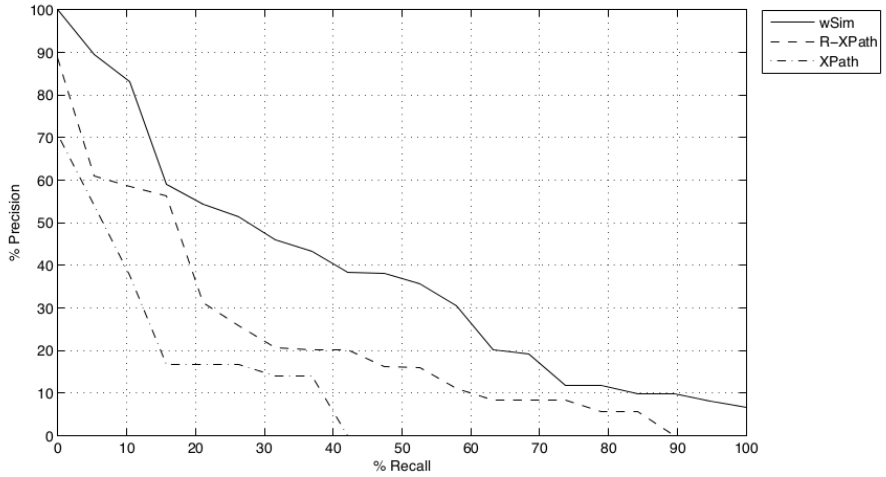


Fig. 9 Precision-Recall chart for XPath, R-XPath, and WSDL-based match-making.

In particular, we run each matchmaking algorithm ten times and we consider the average response time. Each run consisted of publishing all the 894 services of our testbed on the first node at the fastest possibile rate. Before starting the next experiment, we waited until the last node received the last service published, so to be sure that any random fluctuation of an experiment does not interfere also with the following experiments. Each service was described by a single WSDL *facet*, which matches at least one of the queries performed on the second node.

Figure 10 summarizes measured performance. XPath is the fastest match-making method, and it requires on average 5.55 ms. R-XPath is slightly slower, having an average match-making time of 6.90 ms. The method based on WSDL is much slower, requiring an average of 25.91 ms to perform a comparison.

This preliminary analysis shows that both XPath and R-XPath-based match-making mechanisms allow one to create a scalable dispatching network. Its brokers are able to process 10,810 and 8,696 service descriptions per minute, respectively. Note that the whole UDDI Business Registry contained around 50,000 service descriptions before being shut down. Instead, the WSDL-based match-making mechanism only processes 2,316 service descriptions per minute, and thus poses serious scalability issues.

For this reason, we enhanced DREAM by introducing caching mechanisms to speed-up the match-making process. The caching mechanism allowed us to store the similarity values computed in the past and the comparison of two terms, which have been already compared, only requires an access to the cache. At this stage the MRU (Most Recently Used) policy is adopted for caching: i.e., descriptions that are used more are kept longer. Other kind of policies will be implemented in future versions. Based on this, we validated the effects of the caching mechanism by considering two diverse scenarios: a dynamic environment and a static one.

The dynamic environment is characterized by a high ratio of new services and new queries. In this situation, the caching mechanism has limited ability to improve its performance. To simulate this scenario, we subscribed to a query per time, we sent all the services in the benchmark, and we reset the cache before considering the next subscription. Results are reported in Figure 11: the performance of XPath and R-XPath have a slight improvement, and respectively require 5.08 ms and 6.60 ms on average to perform a comparison. Interestingly, the method based on WSDL only requires 11.15 ms (56.97% faster than the version without cache).

In the static environment, service providers publish the same services, and consumers performs the same queries altogether. This is the best-case scenario for caching: after a short initial period in which the system processes the services and the queries for the first time, all the requests can leverage the cache. To measure the performance, we subscribed to all the queries and sent all the services twice. The first time served to fill the cache, and we did not measure the matching time. Instead, we measured the performance when the services were

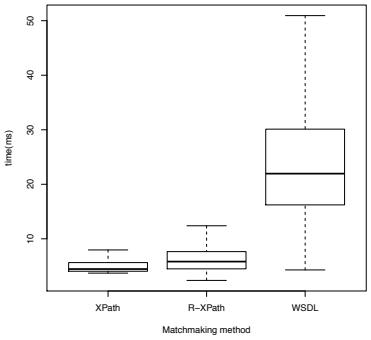


Fig. 10 Performance indicators.

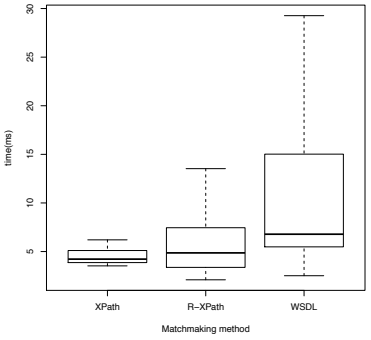


Fig. 11 Performance with caching in the dynamic environment.

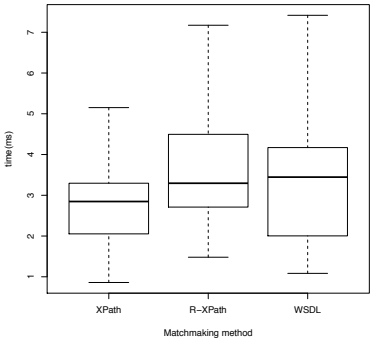


Fig. 12 Performance with caching in the static environment.

sent for the second time. Figure 12 reports the results: the XPath, R-XPath, and WSDL-based methods require 2.74 ms, 3.73 ms, and 3.31 ms for each comparison, respectively. These figures are in line with the requirement of a scalable distributed dispatching network, since brokers can process 21,897, 16,085, and 18,126 service descriptions per minute, respectively.

Interestingly, the architecture of DREAM leverages a lease mechanism, and requires that services be periodically sent through the dispatching network. For this reason, we expect that the real usage scenario is always close to the "static environment", and that the average throughput is thus appropriate. Our experiments showed that DREAM can be used as underlying infrastructure for a scalable dispatching network.

6 Related work

Over the last years, the service community has proposed several approaches for publishing and retrieving Web services. Given the goals of DREAM, we only address two wide classes of approaches: those that concentrate on the architecture of service registries and those that deal with service match-making.

6.1 Registry architectures

The need for a management of service registries in a federated fashion has been recently considered in [21] . In this case, the authors based their approach on the existence of communities that have similar preferences in term of service functionalities. According to those preferences, the services can be organized in different registries where similar services belong to the same registry. According to this scenario, our approach is complementary to what it is proposed in the article as it can be helpful to support the discovery of the services published in the already created registries with the possibility to specify more than one type of query.

Focusing on the technology, current solutions support the cooperation among registries, but they imply that all registries be of a single type and the cooperation needs a set up phase to manually define the information contributed by each registry. For example, UDDI v.3 [9] extends the replication and distribution mechanisms offered by the previous versions to support complex and hierarchical topologies of registries. It also supports the identification of 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 they cooperate are clearly defined.

Similarly, ebXML [4] is a family of standards based on XML to provide an infrastructure to ease the online exchange of commercial information. ebXML fosters the cooperation through the idea that groups of registries share the same commercial interests or are located in the same domain. One of such groups can then be seen as a single logical entity where all the elements are replicated on the different registries. Service retrieval with ebXML registries results ineffective since users must browse pre-defined taxonomies or submit keywords to find desired services.

METEOR-S [10] and PYRAMID-S [11] 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 reg-

istries using different concrete nodes, where the single node is simply a gateway to the logical, global registry. 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, as the semantic layer imposes too heavy constraints on publication policies and also on the way federations can evolve dynamically.

These approaches 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. In addition to this approaches, [22] adopts semantic-based techniques for implementing an infrastructure able to manage a distributed registry. In the proposed architecture, communication among the actors rely on shared spaces, to provide a flexible and scalable solution.

VISR (View based Integration of heterogeneous web Service Registries) [23] allows the communication among registries by means of ATOM feeds. Service providers publish information and updates regarding their services by means of ATOM feeds. Customers can subscribe to these feeds and get new service descriptions as soon as they are available. Simple matchmaking algorithms are provided, allowing customers to select services by considering provided operations and parameters or XPath expressions.

Besides the "classical" approaches, Sellami et al. [24] leverage information on the customer (e.g., interests, previous invocation history) to enrich service descriptions. This allows them to reduce the query space. When a query is performed, the approach selects the registry that is closest to the customer's preferences: the query is then processed by this system. At a more general level, [25] discusses the idea of an open repository environment and addresses some of the key features of DREAM.

6.2 Service match-making

The approach proposed in this paper is a mix of syntactical and semantic matchmaking algorithms that provides to the users a great flexibility in their querying. The different possible queries that can be adopted are well summarized by Klein and Bernstein [26] that identify four main retrieval approaches: keyword-based, concept-based, table-based, and deductive. The match-making algorithm implemented in DREAM is both table-based (because of the use of name-attribute pairs in the facets) and concept-based (because of the use of semantics).

In the area of table-based solutions, also other approaches in the literature rely on the syntax of the Web service description and compare the signature of the requested service against the signatures of existing ones. This type of approach is closely related to the work on retrieving reusable components [27]. In this field, as stated by Zaremski and Wing, there are two types of methods to address this problem: signature matching [28] and specification matching [29]. In particular, signature matching considers two levels of similarity and introduces the *exact* and *relaxed* signature matching. In our work, signature matching represents the core of the solution. In addition, our similarity algorithm also quantifies how similar a Web service is with respect to another one, instead of simply dividing retrieved Web services in exact matching and relaxed matching ones. Furthermore, as in the case of [30], [31], and [32], our approach takes into account the structure of the service description for the match-making process. However, our approach considers the role of each description element with respect to the resulting compatibility between service descriptions. [33] adopts the same approach where the similarity of WSDL descriptions also considers the composite elements as a whole and not separately.

A further class of similarity algorithms [34–39] retrieves Web services through a reasoning process on a semantic specification; [40] complements it with a structural analysis. Description Logic is the usual formalization adopted and results in languages such as OWL-S [41] and WSMO [42]. Even if these approaches are more effective than the ones based on WSDL, building a logic-based Web service description requires more effort for developers. A recent survey of semantic-based retrieval algorithms is published in [43]. This paper is also interest-

ing for the discussion on the open issues in this field. In particular, the authors claim the need for matchmaking mechanisms that cope with "geographically dispersed and non-interrelated service registries". With DREAM we aim to deal with this situation by providing a flexible retrieval approach that does not stick on a single web service description language and does not impose a specific structure or centralized management.

Our work focuses also on the structure of the Web service, for substitution purposes. In the above mentioned algorithms, the result of the retrieval activity is a set of Web services that achieve the same goal. Nothing can be said about how the goal is achieved. In addition, these approaches are usually able to group Web services in similarity classes, i.e., exact match, partial match, and relaxed match. In contrast, our approach offers a finer grained Web service ranking based on a similarity value. The Semantic Web community also adopts SPARQL [44] (Simple Protocol and RDF Query Language), a query language for RDF (Resource Description Framework) documents [45], as a way to express the characteristics of the required Web service [46]. According to a query-by-example approach, our work imposes that the requested Web service be defined by using the same language adopted to describe published Web services, that is, WSDL or SAWSDL.

7 Conclusions

This article introduces DREAM: an innovative infrastructure for the distributed publication of Web services and for their easy retrieval. The proposal, based on previous experiences of the authors, provides a holistic solution for governing the replication of service information by means of user requests and preferences. It provides users with partial, but acceptable, solutions whose fitness is defined through different match making techniques. The experiments conducted and discussed in this paper demonstrate the capabilities of the proposed solutions in terms of precision and recall. They also assess the impact the complexity of queries has on response time.

The flexibility of both service publication and retrieval makes DREAM suitable for situations with different registries distributed over a network and with a high number of services. The use of facets fosters the interoperability of heterogeneous service registries. The publish / subscribe middleware allows DREAM to continuously inform the parties about new interesting services. The different types of queries provide results with different quality attributes, and thus permit different uses of the infrastructure.

The current implementation of DREAM integrates services described using WSDL. As the nature of the services is actually more diversified, we plan to implement the required modules to have Facets for OWL-S or REST-based service descriptions and to test how these kinds of service description models affect the precision and recall.

In addition, future extensions of DREAM will also provide mechanisms to better manage the non-functional aspects of services. Since in the current implementation, most of the work has been focused on describing the operational aspects, service requestors are also interested in performance and security aspects. Suitable mechanisms for considering these aspects and for validating the feasibility of the solution are then required.

Finally, even if the research on technologies related to service registries has been abandoned over the last years, we think DREAM can provide a significant contribution to enabling scenarios where different technologies coexist and "relevant" service information are distributed (to interested users) in a smart and efficient way. This is one of the key enables of the forthcoming *Internet of Services/Things* [47], where information about multitudes of heterogenous services must be communicated to possible users properly and timely.

References

1. M. P. Papazoglou, G. Georgakopolous, Service Oriented Computing: Introduction, Communications of the ACM 46 (10) (2003) 1–5.
2. The UDDI Web site, `http://uddi.xml.org`.
3. M. Clark, `http://www.webservicesarchitect.com/content/articles/clark04.asp` (November 2001).
4. ebXML: Electronic Business using eXtensible Markup Language, `http://www.ebxml.org/`.
5. L. Baresi, M. Miraz, A distributed approach for the federation of heterogeneous registries, in: A. Dan, W. Lamersdorf (Eds.), Service-Oriented Computing - ICSOC 2006, Vol. 4294 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2006, pp. 240–251, 10.1007/11948148_20.

6. P. Plebani, B. Pernici, URBE: Web service retrieval based on similarity evaluation, IEEE Transactions on Knowledge and Data Engineering 21 (11) (2009) 1629–1642. doi:10.1109/TKDE.2009.35.
7. P. Sawyer, Specification language definition, Tech. Rep. A1.D2.3, EC SeCSE Project (2006).
8. L. Baresi, M. Miraz, P. Plebani, A flexible and semantic-aware publication infrastructure for web services, in: Advanced Information Systems Engineering, 20th International Conference, CAiSE 2008, Montpellier, France, June 16-20, 2008, Proceedings, Vol. 5074 of Lecture Notes in Computer Science, 2008, pp. 435–449. doi:10.1007/978-3-540-69534-9_33.
9. L. Clement, A. Hately, C. von Riegen, T. R. (eds.), Universal Description, Discovery and Integration version 3.0.2, `http://uddi.org/pubs/uddi_v3.htm` (10 2004).
10. K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, J. Miller, METEOR-S WSDI: A scalable p2p infrastructure of registries for semantic publication and discovery of web services, in: Information Technology and Management, Vol. 6, Jan 2005, pp. 17 – 39.
11. T. Pilioura, G. Kapos, A. Tsalgatidou, 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.
12. G. Cugola, G. P. Picco, REDS: a reconfigurable dispatching system, in: SEM, 2006, pp. 9–16.
13. A. Carzaniga, D. S. Rosenblum, A. L. Wolf, Design and evaluation of a wide-area event notification service, ACM Transactions on Computer Systems 19 (3) (2001) 332–383.
14. Jini, `http://www.jini.org/`.
15. M. Lennon, D. Pierce, B. Tarry, P. Willett, An evaluation of some conflation algorithms for information retrieval, Journal of Information Science 8 (3) (1988) 99–105.
16. T. Pedersen, S. Patwardhan, J. Michelizzi, WordNet::Similarity - measuring the relatedness of concepts, in: Proc. National Conf. on Artificial Intelligence, July 25-29, San Jose, California, USA, 2004, pp. 1024–1025.
17. N. Seco, T. Veale, J. Hayes, 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, pp. 1089–1090.
18. J. Farrel, H. Lausen, Semantic annotations for WSDL and XML schema, http://www.w3.org/TR/sawsdl/ (April 2007).
19. R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, ACM Press / Addison-Wesley, 1999.
20. M. Klusch et al., *Performance Evaluation of Semantic Service Matchmakers. 5th International Semantic Service Selection Contest*, 2013.
21. M. Sellami, O. Bouchaala, W. Gaaloul, S. Tata, Communities of web service registries: Construction and management, Journal of Systems and Software 86 (3) (2013) 835 – 853. doi:http://dx.doi.org/10.1016/j.jss.2012.11.019. URL `http://www.sciencedirect.com/science/article/pii/S0164121212003123`
22. B. Sapkota, D. Roman, S.R. Kruk, D. Fensel, Distributed Web Service Discovery Architecture, in: Proc. AICT-ICIW '06. Int'l Conf. on Internet and Web Applications and Services/Advanced International Conference on Telecommunications, 2006, pp.136-136. doi:10.1109/AICT-ICIW.2006.85.
23. S. Dustdar, M. Treiber, View based integration of heterogeneous web service registries - the case of visr, World Wide Web 9 (4) (2006) 457–483.
24. M. Sellami, W. Gaaloul, S. Tata, Functionality-driven clustering of web service registries, in: IEEE SCC, 2010, pp. 631–634. doi:10.1109/SCC.2010.70.
25. A. Aschenbrenner, T. Blanke, M. W. Küster, W. Pempe, Towards an open repository environment, J. Digit. Inf. 11 (1).
26. A. Bernstein, M. Klein, Towards High-Precision service retrieval, in: Proc. Int. Semantic Web Conf., ISWC'02, 2002.
27. E. Damiani, M. G. Fugini, C. Bellettini, A hierarchy-aware approach to faceted classification of objected-oriented components, ACM Trans. Softw. Eng. Methodol. 8 (3) (1999) 215–262. doi:10.1145/310663.310665.
28. A. Zaremski, J. Wing, Signature matching: a tool for using software libraries, ACM Trans. Softw. Eng. Methodol. 4 (2) (1995) 146–170. doi:10.1145/210134.210179.
29. A. Zaremski, J. Wing, Specification matching of software components, ACM Trans. Softw. Eng. Methodol. 6 (4) (1997) 333–369. doi:10.1145/261640.261641.
30. A. Zisman, G. Spanoudakis, J. Dooley, A framework for dynamic service discovery, in: 23rd IEEE/ACM International Conference on Automated Software Engineering, L'Aquila, 2008, pp. 158–167. doi:10.1109/ASE.2008.26.
31. E. Stroulia, Y. Wang, Structural and semantic matching for assessing Web-service similarity, Int'l J. Cooperative Inf. Syst. 14 (4) (2005) 407–438. doi:10.1142/S0218843005001213.
32. S. Sellami, O. Boucelma, Web services discovery and composition: A schema matching approach, in: Web Services (ICWS), 2011 IEEE International Conference on, 2011, pp. 706 –707. doi:10.1109/ICWS.2011.105.
33. F. Liu, Y. Shi, J. Yu, T. Wang, J. Wu, Measuring similarity of web services based on wsdl, in: Web Services (ICWS), 2010 IEEE International Conference on, 2010, pp. 155 –162. doi:10.1109/ICWS.2010.67.
34. S. Agarwal, R. Studer, Automatic matchmaking of Web services, in: Int'l Conf. on Web Services (ICWS'06), 2006, pp. 45–54. doi:10.1109/ICWS.2006.35.
35. D. Bianchini, V. De Antonellis, M. Melchiori, Hybrid ontology-based matchmaking for service discovery, in: Proceedings of the ACM symposium on Applied computing (SAC'06), ACM Press, Dijon, France, 2006, pp. 1707–1708. doi:10.1145/1141277.1141681.
36. B. Benatallah, M. Hacid, A. Leger, C. Rey, F. Toumani, On automating Web services discovery, The VLDB Journal 14 (1) (2005) 84–96. doi:10.1007/s00778-003-0117-x.

37. M. Klusch, B. Fries, K. Sycara, Automated semantic web service discovery with OWLS-MX, in: Proc. Int'l Conf. on Autonomous agents and multiagent systems (AAMAS'06), ACM Press, New York, NY, USA, 2006, pp. 915–922. doi:10.1145/1160633.1160796.
38. K. Sycara, S. Widoff, M. Klusch, J. Lu, Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace, in: Autonomous Agents and Multi-Agent Systems, Vol. 5, Kluwer Academic Publishers, Hingham, MA, USA, 2002, pp. 173–203. doi:10.1023/A:1014897210525.
39. M. Paolucci, T. Kawamura, T. Payne, K. Sycara, Semantic matching of Web services capabilities, in: Proc. Int'l Semantic Web Conference on The Semantic Web (ISWC'02), Springer-Verlag, London, UK, 2002, pp. 333–347.
40. R. Amorim, D. Claro, D. Lopes, P. Albers, A. Andrade, Improving web service discovery by a functional and structural approach, in: Web Services (ICWS), 2011 IEEE International Conference on, 2011, pp. 411 –418. doi:10.1109/ICWS.2011.14.
41. D. Martin (ed.), OWL-S: Semantic Markup for Web Services. W3C Submission, `http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/` (November 2004).
42. WSMO Working Group, Web Service Modeling Ontology, `http://www.wsmo.org`.
43. H. Dong, F. K. Hussain, E. Chang, Semantic web service matchmakers: state of the art andchallenges, Concurrency and Computation: Practice and Experience 25 (7) (2013) 961–988. doi:10.1002/cpe.2886.
44. E. Prud'hommeaux, A. Seaborne, SPARQL query language for RDF, http://www.w3.org/TR/rdf-sparql-query/ (W3C Candidate Recommendation) (June 2007).
45. D. Beckett (ed.), RDF/XML Syntax Specification (Revised). W3C Recommendation, `http://www.w3.org/TR/rdf-syntax-grammar/` (February 2004).
46. S. Lamparter, A. Ankolekar, Automated selection of configurable Web services, in: 8. Int. Tagung Wirtschaftsinformatik, Universittsverlag Karlsruhe, Germany, 2007.
47. D. Uckelmann, M. Harrison, F. Michahelles (Eds.), Architecting the Internet of Things, Springer, 2011.

A Validation queries

These are the queries in benchmark SAWSDL-TC1 that have been used to validate the approach presented in this paper. As discussed in Section 5, since these queries are meaningful for the WSDL similarity algorithm, to ensure a fair comparison, queries for XPath and R-XPath have been adapted from them.

- `hospital_investigating_service`
- `shoppingmall_cameraprice_service`
- `surfing_destination_service`
- `surfinghiking_destination_service`
- `surfingorganization_destination_service`
- `title_comedyfilm_service`
- `title_videomedia_service`
- `university_lecturer-in-academia_service`
- `userscience-fiction-novel_price_service`
- `novel_author_service`
- `preparedfood_price_service`
- `recommendedprice_coffeewhiskey_service`
- `researcher-in-academia_address_service`
- `country_skilledoccupation_service`
- `dvdplayermp3player_price_service`
- `geographical-regiongeographical-region_map_service`
- `geopolitical-entity_weatherprocess_service`
- `governmentdegree_scholarship_service`
- `governmentmissile_funding_service`
- `grocerystore_food_service`
- `maxprice_cola_service`
- `book_price_service`
- `bookpersoncreditcardaccount__service`
- `bookpersoncreditcardaccount_price_service`
- `car_price_service`
- `citycountry_hotel_service`

To ensure an independent evaluation of our approach, we used the queries adopted by the creators of the benchmark. For each query, they also define the relevance sets, useful for computing precision and recall.