A Semantic Matching Algorithm for Discovery in UDDI

Unai Aguilera
Tecnológico – Fundación Deusto
Avda. Universidades, 24
48007 Bilbao, Spain
uaguiler@tecnologico.deusto.es

Joseba Abaitua, Josuka Díaz,
David Buján, Diego López de Ipiña
Universidad de Deusto
Avda. Universidades, 24
48007 Bilbao, Spain
abaitua@fil.deusto.es
{josuka, dbujan, dipina}@eside.deusto.es

Abstract

One of the key objectives of web service technology is to construct processes that enable service providers to interconnect with their clients. The industry has developed the UDDI registry to enable the register of services and to make them available to their users. However, the keyword-based search of UDDI frequently return no results because there is no relation between the concepts used by the providers and those used by the clients. This paper presents an UDDI registry extended with semantic functionality by means of a generic semantic discovery algorithm that is not bound to any specific ontology or concepts from it. It enables to match instances and properties without specifically knowing the ontologies being matched. Furthermore, we present an ontology which represents the information associated with a business entity in UDDI, and which enables users to register and search business information semantically also using the former algorithm.

1. Introduction

In recent years, the web service technologies have considerably grown in their application on the enterprise world. The goal of web services is to enable heterogeneous applications to intercommunicate easily. The application of this technology is very adequate for B2B (Business to Business) solutions where providers and clients, with very different characteristics, need to communicate among themselves to perform various tasks. In this scope, where a client can select among many different providers, a process is necessary to help in the selection of those services that are more adequate to perform the desired task. The industry (guided by OASIS Consortium [1]) has developed a standard that enables providers to register their services and make them available to their clients. This system is called UDDI (Universal Description Discovery and Integration) [2]. UDDI is similar to a “yellow pages” where providers advertise themselves and which are used by the clients to search services that could fulfill their needs. When the provider performs a registration the service is described using a set of keywords. During the search process the client specifies some keywords that describe the service he is searching for, and a string based comparison is performed to retrieve the services that meet these requirements.

However, the possibilities of this approach are limited. As explained in [3] the UDDI string-based search is not adequate in most cases because it only permits keywords combined with a classification system to be used in the search process. The keyword based search frequently returns no results because it is not able to recognize the similarities and differences between the concepts used to describe the web services. For example, the search for services described as “car rent” will not return those services registered as “vehicle rent” by the providers because the registry does not known any relation between the two concepts. Therefore, the user of the services needs to know the correct words, used by the service provider when registering, in order to find the desired service. A further discussion of the problem of UDDI keyword search is provided in Section 2.1.

During recent years some solutions to this problem have been proposed. Some of this solutions improve the registry introducing semantics into the registration and search processes by means of the use of shared ontologies to represent the concepts and the relations between them.

The goal of these approaches is to avoid the limitations that exist in the string-based search of the UDDI registry. The introduction of semantics can add new possibilities to the matching process. These semantics are introduced using ontologies. An ontology is a con-
ceptualization of a specific domain that is shared among all the participants.

Ontologies enable participants to know the significance and relation between the terms and concepts being used. In the case of a UDDI registry, the use of ontologies enables the matching algorithm to discover the differences and similarities between the concepts used by the clients in their searches and those used by the service providers when describing their services. This way, in their searches, users can employ concepts that are not exactly equal to the concepts used by providers in their descriptions but are related to them thanks to the ontology. These solutions are presented in Section 2.3.

An UDDI registry not only stores information about the services, but it stores other information related to them (i.e. Business Entity). For this reason, we think that a completely semantic registry must semantically manage all the entities involved in the process. So, we have constructed an ontology to represent the information related with a Business Entity, also enabling the semantic register and search of these entities.

We have also developed a semantic matching algorithm that enables to perform the matching between then user request and the registered descriptions in a general form and without knowing the specific ontology being compared. The algorithm works for any ontology developed in OWL [4, 5], so it can be applied to the search processes of service and business entities described with any OWL ontology (e.g. OWL-S).

The remainder of the paper is structured as follows: Section 2 presents the related technologies and previous work done in the area of adding semantic capabilities to UDDI registries. Section 3 presents the SemB-UDDI architecture explaining the connection between the elements. Section 4 discusses the new algorithm proposed to perform the semantic matching process. Section 5 presents the ontologies that could be used in the register including the ontology developed to represent the Business Entity information. Finally, Section 6 summarizes and concludes the paper.

2. Background

2.1 UDDI

The Universal Description Discovery and Integration is an open industry initiative guided by OASIS that enables businesses to publish their own services and discover other published services. The clients can search by name, description, the business that offers them and other related information. This mechanism based on a keyword comparison implies that the client has to use the exact words that the services provided included when they described their services.

Therefore, when a user starts a search in UDDI he needs to have some knowledge of the words that were used to describe the services or businesses. In most cases, this knowledge is partially available to the client, or is not available at all. It is very likely that in his search the client will be use words that were not included in the description of the service by the provider and, as result, the keyword based search will discard many results that might be useful for the client.

This occurs because the search algorithm is not able to recognize the relationships between the significance of the keywords used by the client and the keywords used by the provider when the service was registered. In other words, the search algorithm does not know if the client and the service provider are using common semantics when referring to the same kind of services. UDDI only works well when the client knows some information about the services, but it fails as a discovery strategy in other cases.

2.2 OWL-S

OWL-S [6] is an ontology for web services constructed using OWL and it has been developed to enable the following tasks: automatic service discovery, automatic service invocation and automatic service composition. A computer can access the description of a web service and it can know exactly what the service does thanks to the shared concepts contained in the ontologies used in the description.

A service described using OWL-S provides three types of knowledge: Service Profile, Process Model and Service Grounding. The ServiceProfile describes what the service does, including functional information such as inputs, outputs, and other non-functional information (category, classification, ...). It is normally used during the automatic discovery of web services. The Process Model describes how the service works and it is an abstract vision of the service operation. Finally, the ServiceGrounding tells how to access the service; it contains all the information related to the real implementation of the service and is used to invoke it automatically.

2.3 Related work

In the last few years, some work has been done in the area of semantic web services discovery. In this section we present a short review of these works.

One proposal for enhancing UDDI with semantic information is made in [3, 7]. In these papers the au-
Authors present an architecture to augment UDDI registries with additional semantic information. They add a new layer to the UDDI architecture that performs the semantic matching between service records. Whenever an advertisement of a service containing semantic information is received the information is extracted and stored in the registry. The services are described using OWL-S, and only the ServiceProfile is used to perform the discovery process. The authors use a one-to-one mapping if the information contained in the OWL-S profile has an equivalent in the UDDI registry. For those OWL-S profile elements that do not have a correspondence with UDDI registry elements a tModel based mapping is used. The authors also present a matching algorithm which they argue is efficient for semantic web service discovery [8] and is an evolution of the algorithm presented in [9]. The authors define four degrees of match for the result of the discovery process: exact, plugin, subsume and fail, ordered from the best to the worst result. The matching algorithm also contains some optimizations, such as the indexation of the registered services, to improve the discovery process.

Another solution is presented in [10], where the authors propose a similar solution to the former, but they add a filtering mechanism that progressively reduces the set of registered services being matched to improve the matching algorithm. The filtering mechanism used is similar to that developed in [11].

An approach for a semantic UDDI registry is presented in [12] and is based on the proposal made in [13], but it extends the UDDI API to support the semantic based inquires and a planning algorithm to help users in web service composition is introduced.

Another work related to combination of UDDI and the semantic web is presented in [14]. It proposes an external matching mechanism to enhance the search in a service registry and to permit the integration of multiple external engines.

A matching algorithm is presented in [18] where the authors propose a more grained ranking of results for the semantic matching providing more relevant results than those that can be obtained performing only a subsume matching. The proposed algorithm performs a matching of the ServiceProfile as a whole, taking into account the service classes in addition to the inputs and outputs of the service.

In this paper we present an enhanced UDDI registry that uses some of the algorithms explained here as roots of the algorithm we have developed. However, we have developed a more general algorithm that not only performs semantic matching between inputs and outputs, but also the semantic matching between other concepts present in the OWL ontology. Therefore, this algorithm can be used to match other concepts present in the OWL-S profile or other ontologies that represent other semantic information. The proposed algorithm is explained later in Section 4.

![Figure 1. SemB-UDDI architecture](image)

### 3. SemB-UDDI architecture

The architecture of the SemB-UDDI registry is shown in Figure 1. It extends the functionality of the standard UDDI registry by adding semantic capabilities. It adds new functions that enable the clients to register advertisements and search among them using the new semantic capabilities of the registry in similar way to the solution presented in [7].

When the client wants to discover a service or business entity he must construct a semantic description of the entity that he wants to find, e.g. the OWL-S ontology for semantic service or the ontology we have developed for the description of UDDI business entities. The description is sent to the SemB-UDDI registry and processed by the matching process to compute the semantic relationship. The results of the match are returned to the user.

The matchmaker module performs provides functionality to register the semantic descriptions in its knowledge base, to compare them with a user request and to unregister them when necessary. It contains the algorithm used to perform the matching process. This algorithm, which will be explained later, performs the semantic comparison between the user request and the registered descriptions and obtains a result that expresses the degree of semantic similarity.

Internally, the matchmaker has various components that contribute to obtaining the functionality. A reasoner that performs the logic reasoning over the information contained in the ontologies stored in the knowledge base. It is used by the semantic matching al-
algorithm to perform the comparison between user request and the registered descriptions. In the current implementation of the matchmaker module we are using Pellet as the reasoner for the matching operations [15]. The reasoner also contains a knowledge base that stores the ontologies and the related information about them. When a new instance (service description or business description) is stored the related ontologies are also downloaded and stored in the knowledge base to allow the reasoner to work with them.

The knowledge base of the reasoner is not persistent so the ontologies and instances need to be reloaded every time the reasoner is restarted. For this reason, the matchmaker module includes a database that is used in order to maintain persistence. Whenever a new ontology or instance is stored in the matchmaker module the database stores the same information to provide persistence. When the semantic matching module is restarted this information is retrieved from the database and loaded enabling the reasoner to start with the information stored in previous sessions.

3.1 Extending UDDI with semantics

The SemB-UDDI project extends the UDDI standard API to enable the registration and search of registered descriptions (services and business). These functions have been added by means of modifying an existing implementation of UDDI called jUDDI [16]. In order to enable the clients to use these new functions we have extended an implementation of a UDDI client called UDDI4J [17]. This way a user can access the new capabilities of the semantic registry. Functions to add and remove service and business entities descriptions have been added with the SemB-UDDI registry.

4. Matchmaker algorithm

The semantic matching process has been implemented using an algorithm based on the ideas proposed in [7] and [18]. In the first paper the authors propose a solution for semantic service discovery where the matching process is only performed between the inputs and outputs of the services being matched. In the second one, an algorithm that matches the Service Profile as a whole is proposed, but it is restricted to the OWL-S ontology.

We propose an algorithm where the matching is performed with all the concepts included in the user request. The goal of the proposed algorithm is to compare instances (registered descriptions and user requests) in a generic form without specifically knowing the ontologies and concepts being compared. That is, the proposed algorithm is able to match OWL instances of any class and it is not only restricted to OWL-S or other specific ontologies. Therefore, it can be used to match other ontologies like the business ontology that has been developed to represent the UDDI Business Entity information or other OWL ontologies used in the description of the registered entities in the registry.

The algorithm is summarized as follows: whenever a user sends a request, which consists on a OWL instance describing the element desired to find in the registry, it is matched against all the stored descriptions. The matching process is performed in a recursive manner using the user request as a query and trying to find the relation between all the terms in the request with the terms of the registered description being compared.

For example, let’s suppose that the client wants to find services of class InformationService, see the profile hierarchy shown in Figure 2, which have an output of type ZipCode. The algorithm will match the user request against those service descriptions, if they exist, registered as WebService, InformationService and BookInformationService. For each registered instance of this classes the process:output property is matched to compute the relation of the desired property, i.e. ZipCode, with the same property, if exists, that was contained in the description of the web service when it was registered. The relationship between the properties is computed and combined with the value obtained in the classes matching and the result is returned to the client. If the property process:output is not found in the registered description the result will be fail because the user wants to find only those services returning a ZipCode, or some concept related to it thanks to an existing ontology. Therefore, those services which does not have an output related to this concept are no valid results for the user because can not fulfill their needs.

Our approach has some advantages over other the approaches explained before in Section 2.3. One advantage is the possibility of change the ontology being used to describe an element in the registry without needing to modify the algorithm. Another one is the reutilization of the algorithm for various ontologies, in the case of our project OWL-S and the SemB-UDDI ontology explained later. And, the last one is that all the concepts in the user request are used to compute the matching process producing a result more closed to the user needs.

In the SemB-UDDI registry all the advertisements (services and businesses) are stored in the knowledge base of the matchmaker module. The semantic advertisements consist of a description constructed using OWL (OWL-S for web services and the SemB-UDDI ontology for business entities explained in section 5).
In the case of a semantic web service this description is usually an OWL-S Service Profile but it could be other OWL ontology. As explained above, the service profile contains all the information needed to perform the discovery of a web service, so this information can be used to perform the discovery of the web services in the registry. In the other case, when registering business entities, they can be described using the ontology constructed for this goal.

Whenever a client wants to search the registry for a semantic web service or a business entity he must construct an instance, using the corresponding ontology, to describe the entity he is looking for. For example, if the user wants to search a semantic web service he must construct a service description (i.e. using OWL-S Profile) to describe the service that he wants to discover. Or, if he wants to discover a business entity he must construct a description using a business ontology. The user description is matched against the stored ones in the knowledge base to search those registered instances that match the request.

During the process of semantic matching the instances, classes, properties and values of the registered advertisements and the user request are compared to determine the degree of match between them. The results of the match are ranked from the best to the worst possible result in ascending order.

The possible results of the match, as explained in [9], are: exact, which means that the concepts being compared are exactly the same; plugin, which means that the request concept subsumes the registered concept; subsume, when the registered concept subsumes the concept contained in the request, and fail when no one of the former relationships occurs.

The Table 1 summarizes the values assigned to each of the possible results of the match. This rank will be used by the algorithm to compute the global result of the matching process.

### 4.1 Algorithm main loop

The matching algorithm is a generic algorithm for the comparison of OWL instances that performs the matching without knowing the ontologies and concepts being compared.

```java
computeMatch(request) {
    matchlist = empty
    regInstances = getInstancesRelated(request)
    for each instance in regInstances {
        result = matchInstance(instance, request)
        if (result != fail)
            add result to the matchList
    }
    return matchlist
}
```

**Figure 3. Matching algorithm main loop**

The algorithm starts retrieving from the knowledge base those instances that have some semantic relationship with the instance contained in the user request. First, it obtains the class of the user request and then it retrieves from the knowledge base instances whose classes are related with it. For example, if the user sends a search request that is an instance of an OWL-S ServiceProfile, the algorithm will retrieve the instances whose class is a super-class, a sub-class or is equal to that of the user request. Each of these retrieved instances is compared with the user request in order to obtain the degree of similarity between them. The main loop of the matching algorithm is presented in the Figure 3.

The matchInstance function computes the degree of similarity between two OWL instances. This similarity is obtained in the following way: first, the relationship between the classes is obtained, and second, the properties of the instance are matched. If the match of the

<table>
<thead>
<tr>
<th>Result</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>0</td>
</tr>
<tr>
<td>Plugin</td>
<td>1</td>
</tr>
<tr>
<td>Subsume</td>
<td>2</td>
</tr>
<tr>
<td>Fail</td>
<td>3</td>
</tr>
</tbody>
</table>
classes returns fail, the matching process stops because the user request cannot be satisfied. If the result of the classes matching is not fail we proceed to compute the properties in a similar way. To obtain the final result of the two processes we compute the worst result of the two semantic comparisons, that is, the result with the highest rank in the table shown before.

If the global result computed during the matching process is different from fail, the result is added to a list of results and will be returned to the user.

    matchInstance(instanceReg, instanceUser) {
        result = matchClasses(instanceReg, instanceUser)
        if result != fail {
            matchResult = matchProperties(instanceReg, instanceUser)
            if matchResult > result
                result = matchResult
        }
        return result
    }

**Figure 4. Matching instances**

### 4.2 Matching classes

The matching of the classes is performed by the `matchClasses` function in the following way: if the class from the request and the class from the advertised instance are equal the result is `Exact`; if the class of the request subsumes the class of the advertised instance, the result is `plugin`, because the user is searching for instances of a more generic class and it can be satisfied by any of its subclasses; if the registered instance subsumes the class of the user request the result is `subsume`, which is a worse result than the former because the user is searching for a more specific class than the one being compared. If none of these cases occurs the result is `fail`.

For example, if the user requests a service of type `VehicleRent` it will be able to use all registered services defined as `CarRent` because these services are subclasses of those contained in the user petition and the result of the process will be `plugin`.

### 4.3 Matching properties

After computing the relation between the classes of the instances, and only when a result different from `fail` is obtained, the algorithm will proceed to perform a comparison between the properties from the user request and the registered instance.

    matchClasses(instanceReg, instanceUser) {
        rClass = instanceReg.getClass()
        uClass = instanceUser.getClass()
        if uClass = rClass return exact
        if uClass subsumes rClass return plugin
        if rClass subsumes uClass return subsume
        otherwise return fail
    }

**Figure 5. Matching classes**

This matching is performed by obtaining each of the properties contained in the user request and comparing them with the properties of the same type that exist in the registered instance. There are two cases when comparing properties from instances in OWL: data properties, which have simple values, and object properties which have values that are other instances.

When comparing literal values the matching is done by the `matchDataProperties` function, and it is performed in the following way: the associated values of the property are obtained from the user request and the registered instance. If the values are not concepts from an ontology, a literal comparison occurs. This literal comparison can only return two possible results `exact` or `fail`, that correspond to the case when the two values are exactly equal or different, respectively.

The other case when matching literal values occurs when the values being matched are URLs that point to concepts of an ontology, for example when matching the type of an OWL-S input or output by means of the `process:parameterType` property. In this case the literal values will be matched as classes using the algorithm explained in Section 4.2.

When comparing object properties, the match is performed reusing the same algorithm explained before. The instances are retrieved and compared using the `matchInstance` function again in a recursive manner. This way, it is possible to compare all the user request with the registered instances without knowing the concepts being compared.

The algorithm will navigate through all the user request computing the matching between all the classes and properties with those included in the registered descriptions in the registry.

### 4.4 Computing the final result

The matching process is always performed taking the user request as a description that contains all the minimum requirements that a semantic description registered in the knowledge base must satisfy in order to be returned to the user. For example, if the user request contains a property with a specific value, only those in-
stances that have the same property (or semantically related) with the same value can be returned to the user. If the registered instance does not have an associated value for a property contained in the user petition the result of the matching will be fail, because the user is searching for an instance with specific characteristics that cannot be completely satisfied with the registered instance.

The final result of all the semantic matchings of instances, classes, properties and their associated values are combined to obtain the final matching result. The result obtained by the user after performing a matching request is the worst of all the results obtained when comparing the classes, properties and instances contained in each registered description. The reason is that we think that the user request contains all requirements that must be satisfied by a registered description. Therefore, when a requirement cannot be totally fulfilled it must affect the overall result of the current matching process.

Those registered descriptions that obtain a fail result in one the semantic matching (classes, properties, values of properties) are not returned to the user because they are not able to satisfy the user request.

5. SemB-UDDI ontologies

The information stored in the SemB-UDDI registry is described using various ontologies, some of them reused from existing ones, while others have been constructed expressly to represent specific entities required in the semantic registry. The description of the web services used in the registry and discovery processes can be performed with the ServiceProfile of the OWL-S ontology. OWL-S is a very general ontology created for the description of any kind of web service and, for this reason, it only introduces some concepts and relations that are common and useful for all web services without considering their specific domain of application.

As explained in the introduction we consider that a semantic registry must semantically manage all the information of the entities that are being stored. The UDDI standard defines the entities, the relations between them, and how they are stored in the registry. Those specifications include web services and some other elements as business entities. The business entity, as explained in the UDDI standard specification, represents the business that offers the services.

We think that the semantic description of this entity is also needed to achieve the goal of a complete semantic UDDI registry. This way a user can search all the data contained in the registry semantically without the need to consider the distinction between what information is semantically described and what is not. For this reason, we think that an ontology to describe the information that corresponds to a business entity is needed. We have created an ontology to represent this information based on the specification of a business entity contained in the UDDI standard. We have represented this information because we want the SemB-UDDI registry to be able to maintain compatibility with a standard UDDI registry. This ontology enables the minimum information associated with a business entity to be captured while it enables the possibilities of the semantic web to be applied. Using the proposed ontology to represent the data we are able to capture all this information adding semantic searching capabilities at the same time. The Business ontology is depicted in Figure 6.

![Figure 6. SemB-UDDI Business Ontology](image)

6. Conclusions and future work

In this article we have shown how a UDDI registry can be extended to allow the registry and semantic search of web services and related business entities.

We have proposed a generic matching algorithm that allows the discovery of the registered entities to be made. This algorithm makes a comparison between all the concepts that appear in the user’s request allowing a greater flexibility in the searches. In addition, this algorithm is not only related to a specific ontology and, therefore, the same algorithm can be used to make the semantic discovery of other entities described with OWL. The algorithm has been implemented completely and its currently being tested.

In addition, we have proposed an ontology for the description of the business entities, which allow basic information about business presented in the UDDI standard to be captured and at the same time enables to support the possibilities of the semantic web.
In future work we plan to test the proposed algorithm in discovery tasks based on other parts of the OWL-S ontology, like the ServiceModel sub-ontology. The ServiceModel ontology describes how to use the service, and it contains useful information for service composition tasks. We will try to prove the usefulness of the proposed algorithm in discovery oriented towards automatic composition of semantic web services.

Also, we are planning to match the algorithm to use wildcards in the specification of property values in order to improve the search mechanism.

7. Acknowledgments

This work has been partially supported by Basque Government (under grant number UE2005-11, project “SemB-UDDI”, of the Department of Education, Universities and Research, and under grant number S-PR06R001, project “SEMTEK II”, of the Department of Industry, Commerce and Tourism), and by Codesyntax (http://www.codesyntax.com).

8. References


