A Generic Matching Algorithm for Semantic Discovery

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

Joseba Abaitua, Josuka Diaz-Labrador,
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

This paper presents a generic semantic discovery algorithm, based in OWL, for UDDI registries 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 and it enables to use all concepts contained in the request resulting in a matching closer to the user needs. Also, the generality of the proposed algorithm enables to change the ontology being used to describe an element in the registry without needing to modify the algorithm and therefore, it enables to use other OWL ontologies to describe the information stored in the registry.

1. Introduction

The UDDI, (Universal Description Discovery and Integration) [1], is a registry similar to a “yellow pages” directory where providers advertise themselves and which is used by the clients to search services that could fulfill their needs. 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 [2] 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.

During recent years some solutions to this problem have been proposed. Some of these 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 [2, 6, 7, 8, 9, 10, 11, 12, 13, 14], 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 [3, 4] instances of any class and it is not only restricted to OWL-S [5] 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.

2. Matchmaker algorithm

The semantic matching process has been implemented using an algorithm based on the ideas proposed in [6] and [14]. In the first paper the authors present 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.

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. 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.

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. 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 descriptions 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 [8], are: exact, which means that the concepts being compared are exactly the same; plugin, which means that the request concept subsumes the registered concept; subsumes, when the registered concept subsumes the concept contained in the request, and fail when no one of the former relationships occurs.

The following section explain the algorithm in full detail.

2.1 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.

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 1.

```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 1. Matching algorithm main loop**

The **matchInstance** function, see Figure 2, 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 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.

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.

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

**Figure 2. Matching instances**

2.2 Matching classes

The matching of the classes is performed by the **matchClasses** function in the following way, see Figure 3: 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.

```
machineClasses(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 3. Matching classes

### 2.3 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.

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.

When the values being matched are URLs that point to concepts of an ontology, another case of literal-values match occurs. An example of this is the match 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 2.2.

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.

### 2.4 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 instances 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. This is so because we believe that all requirements that must be satisfied by a registered description should be contained on the user request. 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.

### 3. Conclusions and future work

A generic matching algorithm has been proposed that allows the discovery of registered entities with high search-flexibility through a comparison between all concepts in the user-request. 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 is currently being tested.

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 enhance the matching algorithm with the inclusion of wildcards in the specification of property values, so that the search mechanism improves.

4. 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).

5. References


