Using Semantic Web Services to Automatically attend to Educational Requests

Heitor Barros[1], Ivo Calado[2], Marlos Silva[1], Ig Ibert Bittencourt[1], Evandro Costa[1]

[1]Instituto de Computação – Universidade Federal de Alagoas (UFAL)
Av. Lourival Melo Mota, Km, 14 - Maceió - AL – Brasil

[2]Departamento de Sistemas e Computação- Universidade Federal de Campina Grande, Campina Grande, Paraíba, Brasil

[1]{heitorjsbarros, marlos.tacio, ig.ibert, ebcosta}@gmail.com, [2]ivocalado@embedded.ufcg.edu.br

Abstract. *With the emerging growth of Web Services technology, the Web is now evolving to become a service provider. In this context, a major challenge is the Semantic Web Services discovery and composition problem. This task is hard because injecting context into adaptive service integration and management raises a number of significant difficulties, i.e. defining and using effective and practical metrics to manage adaptation in terms of comparing ability of different services to adapt. This paper proposes an algorithm for Semantic Web Services discovery and composition that uses similarity metrics to achieve such goals. Additionally, an illustrative scenario in the context of education is presented to demonstrate the use of the proposed approach.*

1. Introduction

Web services are loosely coupled software components, published, located and invoked through the web. As they rely on open standards for interfaces and protocols definitions, its main use is as basic blocks of software construction on service-oriented architecture. The prosperity of such approach has gained a variety of domain applications and the research community have identified two major investment areas: Web Services discovery and Web Services composition.

Once the services interface is already standardized in the sense that all them are expressed in terms of input, output and a description of the internal processing, the integration of such services is therefore encouraged. This integration, also known as Web Services composition, has the purpose of providing the final user with a single unified service, hiding the distribution and heterogeneity of services offered by service providers.

However, since the services description is usually done as free text in WSDL [Christensen 2008], adding context-based semantic description to services has gained importance and has been driving the research on the ability of semantically matching services instead of doing that syntactically - which naturally leads to mechanisms for retrieving semantics from the services' syntactic description [Ma 2008]. The Web Service that use an ontology language to specify its description is called Semantic Web Service and an approach for automatic discovery and composition of such elements is the focus of this work.

2. Related Work

As a very active field of research, the Semantic Web Services discovery and composition problem is approached by several initiatives from some groups worldwide.

Not every time the problem of discovery is faced along with the composition one, as seen on OWL-S/UDDI Matchmaker and OWLS-MX [Klusch 2006], which are tools for web services semantic discovery that support OWL as the domain ontologies description language and OWL-S [Martin 2004] for the services description. The difference between both approaches is that while the first one makes a simple matching between the input and output parameters, the second one uses a hybrid strategy combining logic and concept similarity metrics for service matching, which empowers the input and output parameters analysis by adding similarity degrees between them as well as several methods to reach such values. Additional related work regarding similarity metrics on services matching can be also found on WSMO-MX [Kaufer 2003].

Automatic composition has been mainly viewed as a planning problem [Narayanan 2003] and usually treated separately from service discovery [Kuster 2007]. However, there are some works that, together with this one, clearly see the benefits of automated service composition after a service discovery phase. One of such proposals is the IBM STWS (Semantic Tools for Web Services) [IBM 2005], which is able of handling services semantically described in WSDL-S whose domain ontologies are expressed in OWL. Such tool uses an algorithm based on backward chaining procedures to combine services starting from the given output parameters until reaching a chained combination that fits the required service. Through its description, one notices a lack of concerning on the quality of created compositions since there is no mention to any evaluation strategy.

The DIANE approach, like the one proposed in this work, is meant for automatically discover and compose semantic web services. Its main features include fuzzy sets to analyze similarity between required and on-the-fly obtained services - be them composed or not - and dealing with user preferences, covered by the DSD service description language, to find the best suitable service instead of a set of input and output parameters. To the best of our knowledge, the differences between both approaches are the use of W3C's recommended OWL-S language for semantic services descriptions and the similarity metrics used by OWLS-MX and WSMO-MX for matchmaking.

3. The Proposed Algorithm

This section presents the details of the proposed algorithm. It allows the automatic discovery and composition of semantic Web Services described in OWL-S and uses similarity metrics to evaluate the quality of the proposed solution. This work was developed as an extension of [Giv 2004], from which the following features can be listed:

- Direct and indirect matching of services;
- Weighted directed graph representation for the services relationships;
- Shortest path algorithms to locate the best path in the weighted graph, increasing composition accuracy.

The algorithm implementation was done using the Java language with the MindSwap Application Programming Interface (API) to create and manipulate objects which represent semantic services.

3.1. The proposed algorithm

The proposed algorithm is composed of two execution steps, Figure 1. The first one is executed during the insertion of the service into the search directory. This step is called “preprocessing step”. The second one, performed during discovery events, is called “processing request step”.

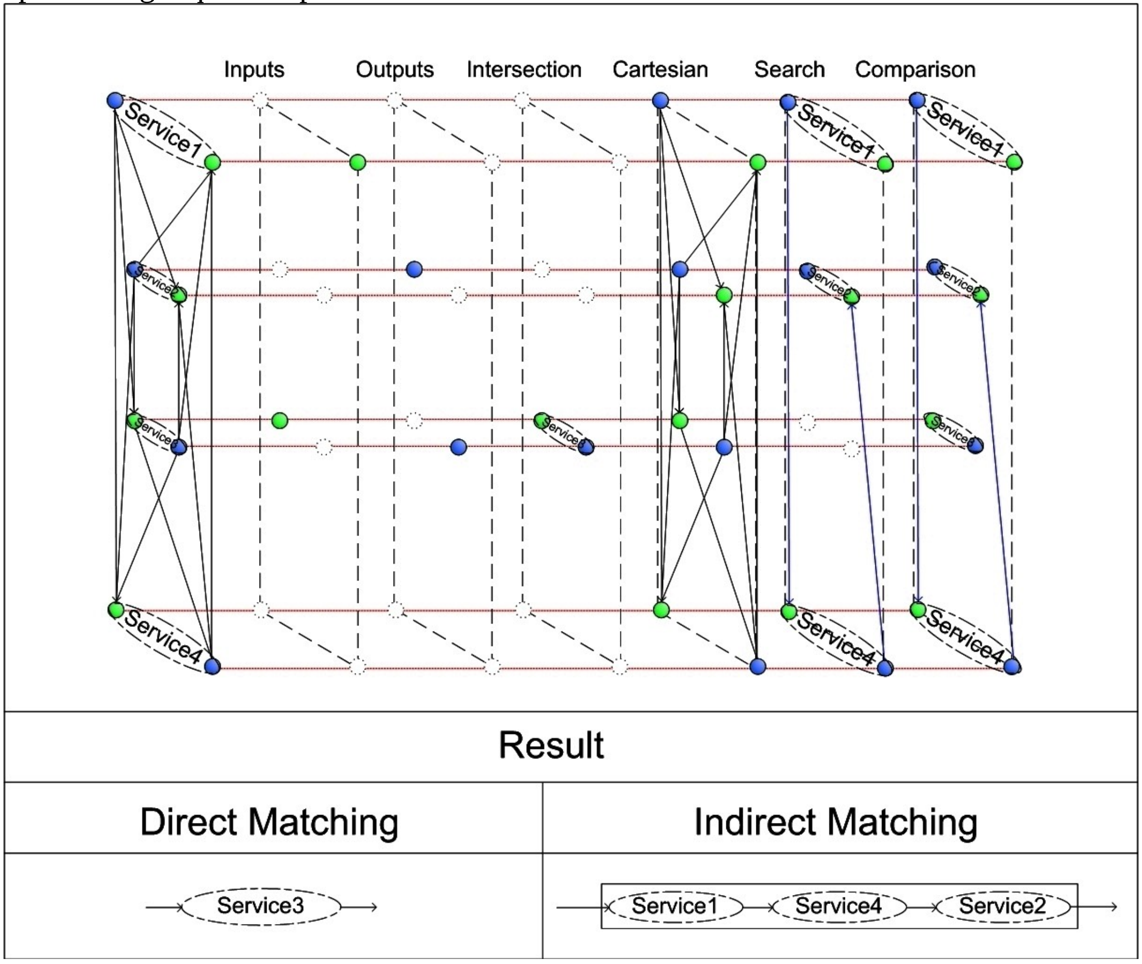

Figure 1. Algorithm's steps.

3.1.1 Preprocessing step

This step includes the tasks insertion and removal from the service directory. The search tool keeps updating to always provide consistent information about the stored services. The second one, performed during discovery events, is called 'request processing step'.

- The nodes represent the services;
- The edges represent the relationship among services.

Every relation that is established between two services has a determined weight that is equivalent to the compatibility of the first service's input parameters to the second one's output parameters. Such compatibility measurement is defined through similarity metrics, as defined in [Klusch 2006].

From the graph-weighted model, it is possible to construct paths and consequently enable services composition. For each newly inserted service, similarity metrics between this service and the ones already in the graph structure will be calculated. The results will vary on the adopted similarity metric.

3.1.2. Processing request step

The format of a request is a 'perfect' service described in OWL-S. This service presents all the desired input and output parameters of the real service to be discovered. The return of that execution is a Java object (*Service* class) representing the desired service. This object can represent either atomic or composite services. This feature creates a layer of abstraction to the client application, since the service composition becomes transparent.

Summarizing, here are the service discovery process steps.

1. Create two empty sets: the first one contains the services whose inputs match the request, called *I*, while the other set contains the services whose output match the request, this one is called *O*;
2. Scan all nodes of the graph, and using the compatibility filter as basis, perform a search for services that fit for *I* and *O* sets;
3. Try to establish an intersection between the sets *I* and *O* through selecting the requests is most compatible services. If such operation is successful, then there is a direct matching;
4. Establish a Cartesian product between *I* and *O*;
5. Input and Output weights are assigned to the graph edges as defined by the adopted similarity metric;
6. A graph search algorithm is applied to find the shortest path between the Cartesian product nodes. The Dijkstra algorithm is being used in this implementation, but that is not a restriction. Other similar and improved algorithms could be used instead;
7. The results of direct and indirect matchings are compared and the service, or chain of services, with the highest similarity value, according to the request, is selected;
8. If a chain of services was chosen, then a composite service is created. A bind between the input and output parameters is done and the sequential execution of all bound services is automatically performed;
9. The service is returned;
10. If neither a direct matching nor an indirect one satisfies the request (the sets *I* and/or *O* are empty or there is no path at all between an element of *I* and an element of *O*) then nil is returned.

3.2. Extensions of the original algorithm

As previously mentioned, the algorithm presented in this work is based on the one presented in [Giv 2004]. This way, additional features were implemented to improve the discovery process. The list of implemented improvements is presented below:

- Direct and indirect matching are defined, then the best set of services that meet the request is identified, since that depends on the services characteristics, a composite service can be more suitable to the request than a single service;
- Weight measurements were added to the edges of the graph. This enables verifying services relationship strength, which means semantic proximity. The weight in each edge is based in the use of similarity metrics among the parameters of the related services. The original algorithm does not comprise handling a weighted graph;
- A Cartesian product is created between the input and output set of services (see 3.1.2) instead of just choosing a service in each set, as it occurs in the original algorithm. This change, although prone to increase processing, allows a more refined service composition.
- A graph search algorithm has been used to find the shortest paths in a weighted graph.

4. Similarity metrics

Similarity metrics has been used by a large number of communities, such as statistics, artificial intelligence and databases. Measuring similarity between entities like internet documents, letters, images, software source code are some of its application.

The analysis of such similarities consists in comparing two entities while aiming to determine how similar they are. A big part of the similarity functions are metrics that treat data instances in a metric space, establishing distances in this space to define a degree of similarity. For such calculations, the concepts of vectorial spaces are used, where entities that are being compared are converted into vectors, and algebraic expressions (Euclidean distance, cosine etc) a measurement of distance between these vectors can be calculated.

4.1 Cosine Similarity

Cosine similarity treats entities as n-dimensional vectors. As this metrics is based on the distance between two vectors, Figure 2 which is given by the angle between them, the cosine of such angle will represent the similarity. By definition, cosine is established in the interval 0 <= c <= 1, where we interpret two vectors to be identical when the measure is 1 and the opposite when the result is 0, which means the vectors are orthogonal and thus, are not compatible.

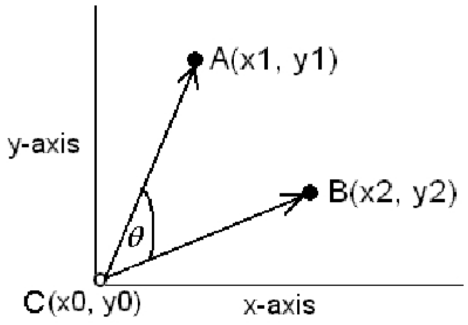

Figure 2. Cosine Similarity.

The proposed algorithm uses this metric to seek for a service that mostly fit the request, directly or indirectly. This approach uses the input and output parameters of each service in order to obtain a similarity level and being able to decide which one adequate to the requester.

4.2. Example

In order to simplify the similarity calculation of the two entities, an Educational ontology is used. To compare the similarity between the Problem and Content entities, the following steps are taken:

4.2.1. Step1 – Defining the entities' ancestor

Going through the ontology a scheme is put together with all the entities' ancestor.

- (Problem) – (Problem, Resource)
- (Concept) – (Concept, Resource)

4.2.2. Step2 – Define all entities involved

Once the each entitie's ancestor re define, the union of these sets is taken in order to define which entities are part of the calculation:

- (Problem U Concept) = (Problem, Concept, Resource)

4.2.3. Step3 – Building comparison vectors

Based on the vectors built on steps 1 and 2, the comparison between the ancestor and the union set is made, building a new vector with the result of this comparison. If one of the ancestor does not contains an entity defined in the union set, 0 is added to the respective index of the results vector, and 1 of the opposite occurs.

- Building vector V1:
    - (Problem) = (Problem, Resource)
    - (Problem U Concept) = (Problem, Concept, Resource)
    - V1 = (1, 0, 1)
- Building vector V2:
    - (Concept) = (Concept, Resource)
    - (Problem U Concept) = (Problem, Concept, Resource)
    - V1 = (0, 1, 1)

4.2.4. Step4 -Calculating the cosine of the angle between the vectors

With the vectors at hand, the cosine between them is computed, and the closer it is to 1 the more similar the entities

are.

:

$$\cos(\theta) = \frac{v1.v1}{|v1||v2|} = \frac{1}{2} = 0{,}50$$

So, the similarity between these entities is 0,50

5. Illustrative Scenario

The aim of this section is to illustrate the features of the proposed algorithm through the development of a case study in e-learning. In this scenario, a student is interacting with the system in a problem solving activity, where the systems use the algorithm to provide a hint to the student based on the problem.

Some semantic web services were developed to provide some pedagogical resources based on domain ontology, Figure 3, such as:

- *ProblemConcept Service*: it gets a concept based on a problem. The input of this service is *Problem* and the output is a *Concept*;
- ConceptContent Service: the input of this service is *concept* and the output is a *content*;
- ContentExplanation Service: the input of this service is *content* and the output is an explanation;
- ShowResource Service: this service indicates which is the next resource to be viewed by the student. It has a user as input and a resource as output.

Inside this illustrative scenario, a software customer possible and desired interaction would be a service that returns an explanation based on a Multiple Choice Problem presented. However, there is no service to fulfill this requirement. From this scenario, services composition would dramatically help increasing process speed and easing process complexity to the user.

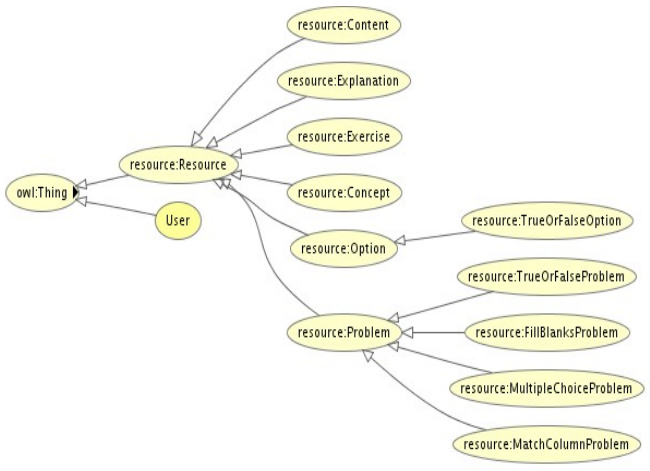
resource:Content
resource:Explanation
resource:Exercise
resource:Resource
owl:Thing
resource:Concept
User
resource:TrueOrFalseOption
resource:Option
resource:TrueOrFalseProblem
resource:Problem
resource:FillBlanksProblem
resource:MultipleChoiceProblem
resource:MatchColumnProblem

Figure 3. Domain Ontology.

The composition evolving these three services (ProblemConcept Service, ConceptContent Service, and ContentExplanation Service.) is fit for solving the described problem with some restrictions. The input parameter for the *ProblemConcept Service* is not the same as the one from the required service. However, due to the services' semantic descriptions, it is possible to infer information about such services parameters to allow the use of non-exact matching services, for instance , subtypes, supertypes and so on. From this property on, there are several propositions that make use of such type of inference. Our proposal, however, uses, as already mentioned, similarity metrics aiming a more accurate evaluation of each parameter's characteristics. The use of such metrics enables the establishment of a relation between *Problem* and *MultipleChoiceProblem* classes to create a composition that fulfills the requirement.

The creation of a semantic description model is mandatory to enable the search for a specific service. The description of the above example will have the MultipleChoiceProblem as input and a Explanation as output, provided that we are searching for a service that executes all the processes inside the E-Learning workflow.

In the pre-processing step, all the E-Learning related services are added to the services repository as well as described in section 3.1.1. Following the services repository population, there is the request processing step. In this step, the service model of the request is used in the search for the most similar service (or service composition). Also, similarity metrics are used to perform the comparison between the requested and the repository services, and the most similar entry is chosen as a possible direct match. After having defined a probable direct matching, the search for a service composition whose similarity is close to the requested service is begun, Figure 4.

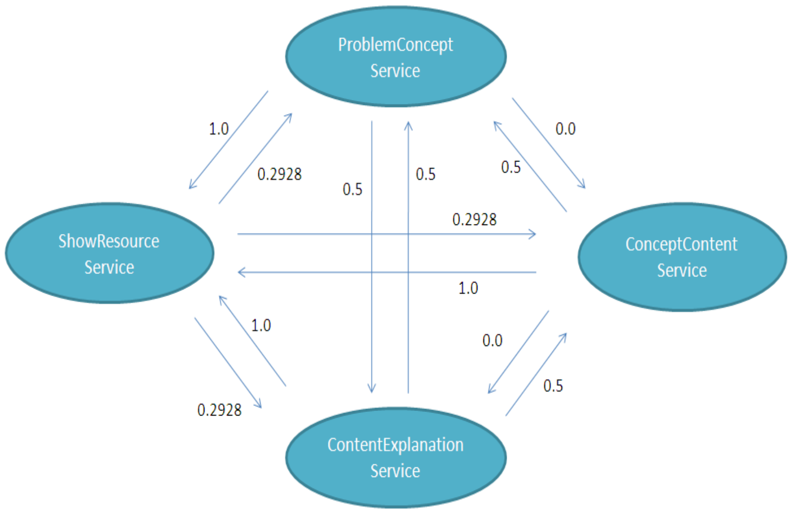

Figure 4. Similarity Graph.

To perform such search, a search for the shortest path in the weighted graph is performed to point the chosen composition and the composition-related services are pointed as possible indirect match. The reason for selecting the path with the shortest cost is that the weight of the edges represent the similarity between the output of a service and the input of the adjacent service, which means the smallest lost of information between services. The results of the possible direct and indirect matching are shown on Table 1.

	Services	Similarities
Direct Matching	Concept-Explanation Service	0295875
Indirect Matching	[ProblemConcept, ConceptContent, ConceptExplanation]	0,09175

Table 1: Results

After the definition of the possible direct and indirect matches, the similarity metrics are used again to help choosing the most similar service in relation to the request and the choice is returned as the search result. In the example, the returned result is the composition from figure 5.

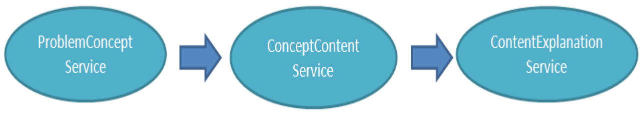

Figure 5. Services Composition.

As seen on table 1, the composition was chosen through the detected similarities since it presents a higher similarity to the request. It is worthwhile quoting that, although the *ConceptExplanation* service is present in both direct and indirect matches - that could lead the reader to think on picking the direct match as a search result because it would be a more direct execution - the indirect match was chosen as final result. Such behavior can be clarified due to the *ConceptExplanation* output parameters have low similarity, which affects its matching degree, while being part of a composition, the whole set has a higher matching degree, which therefore makes it more suitable for an answer.

Finally, the scenario tried to illustrate the developed algorithm work, exhibiting its service discovery capabilities by using services matching, composition and similarity metrics.

6. Conclusion and Future Work

Although there is no widely accepted solution for web services discovery and composition in terms of strategy for selection, composition and metrics for measuring the quality of the delivered service, this paper intended to extend the scope of the discussion to the semantic web environment, which is expected to continue challenging researches worldwide, and presented an algorithm for automatic discovery and composition of Semantic Web Services. As an evolution of a previously existing algorithm for the same purpose, this new approach brought the use of similarity metrics to help improving the quality of composite-services.

As future work, we quote the importance of analyzing which similarity metric gives the best approximation values to work on and compare them. This is intended to be done by exercising the algorithm through the many scenarios and situations available on literature. This way, subsidy for new improvements is expected to arise shortly. Examples of such investment areas are the automatic selection or simultaneous use of all available similarity metrics to better help software to decide and reach adaptation goals on any desired composition scenario.

References

Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl, March 2001. Last access on August of 2008.

Giv, R. D., Kalali, B., Zhang, S., Zhong, N. and Lopez- Ortiz, A. Algorithms for direct and indirect semantic matching of web services. Technical report, University of Waterloo, 2004.

IBM. Semantic tools for web services. http://alphaworks.ibm.com/tech/wssem, June 2005. Last access on August of 2008.

Kaufer F. and Klusch, M. Wsmo-mx: A logic programming based hybrid service matchmaker. *European Conference on Web Services (ECOWS'06)*, pages 161–170, 2003.

Klusch, M., Fries, B. and Sycara, K. Automated semantic web service discovery with owls-mx. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, New York, NY, USA, 2006. ACM Press.

Kuster, U., KunigRies, B., Stern, M. and M. K. . Diane an integrated approach to automated service discovery, matchmaking and composition. *WWW 2007*, pages P 1033–1042., 2007.

Ma, J., Zhang, Y. and He, J. Efficiently finding web services using a clustering semantic approach. In *CSSSIA*, page 5, 2008.

Martin, D., Burstein, M., Hobbs, J. R., Lassila, O., Mc-Dermott, D., Mcilraith, S. A., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N. and Sycara, K. Semantic markup for web services. http://www.w3.org/Submission/OWL-S/, November 2004. Lat access on August of 2008.

Narayanan, S. and McIlraith, S. Analysis and simulation of web services. *Computer Networkds,*, 42(5):675?693., 2003.