Behavioral matchmaking for service retrieval

Daniela Grigori, Juan Carlos Corrales*, Mokrane Bouzeghoub
PRiSM, Universite de Versailles Saint-Quentin en Yvelines, France
*Telematics Engineering Group, University of Cauca, Colombia**
Daniela.Grigori@prism.uvsq.fr; jcorral@unicauca.edu.co; Mokrane.Bouzeghoub@prism.uvsq.fr

Abstract

The capability to easily find useful services (software applications, software components, scientific computations) becomes increasingly critical in several fields. Current approaches for services retrieval are mostly limited to the matching of their inputs/outputs. Recent works have demonstrated that this approach is not sufficient to discover relevant components. In this paper we argue that, in many situations, the service discovery should be based on the specification of service behavior (in particular, the conversation protocol). The idea behind is to develop matching techniques that operate on behavior models and allow delivery of partial matches and evaluation of semantic distance between these matches and the user requirements. Consequently, even if a service satisfying exactly the user requirements does not exist, the most similar ones will be retrieved and proposed for reuse by extension or modification. To do so, we reduce the problem of behavioral matching to a graph matching problem and we adapt existing algorithms for this purpose. A prototype is presented (available as a web service) which takes as input two conversation protocols and evaluates the semantic distance between them; the prototype provides also the script of edit operations that can be used to alter the first model to render it identical with the second one.

Keywords: web services, services retrieval, behavioral matching

1 Introduction

The capability to easily find useful services (software applications, software components, scientific computations) becomes increasingly critical in several fields. Examples of such services are numerous:

- Software applications as web services which can be invoked remotely by users or programs. One of the problems arising from the model of web services is the need to put in correspondence service requesters with service suppliers, especially for services which are not yet discovered or which are new, taking into account the dynamic nature of the Web where services are frequently published, removed or released.
- Programs and scientific computations which are important resources in the context of the Grid, sometimes even more important than data [12]. In such a system, data and procedures are first rank classes which can be published, searched and handled. Thus, the scientists need to retrieve procedures with desired characteristics, to determine if a required calculation was already carried out and whether it is more advantageous to carry out it again or to retrieve data generated previously.
- Software components which can be downloaded to create a new application. To reduce the development, test and maintenance costs, a fast solution is to re-use existing components.

In all these cases, users are interested in finding suitable components in a library or collection of models. User formulates a requirement as a process model; his goal is to use this model as a query to retrieve all components whose process models match with a whole or part of this query. If models that match exactly do not exist, those which are most similar must be retrieved. For a given task, the models that require minimal modifications are the most suitable ones. Even if the retrieved models have to be tailored to the specific needs of the task, the effort for the tailoring will be minimal.

In this paper we argue that, in many situations, the service discovery process requires a matchmaking phase based on the specification of the component behavior. The idea behind is to develop matching techniques that operate on behavior models and allow delivery of partial matches and evaluation of semantic distance between these matches and the user requirements. Consequently, even if a service satisfying exactly the user requirements does not exist, the most similar ones will be retrieved and proposed for reuse by ex-

tension or modification. To do so, we reduce the problem of service behavioral matching to a graph matching problem and we adapt existing algorithms for this purpose.

In the next section we present several motivating scenarios. Section 3 presents existing approaches for service retrieval and shows their drawbacks for the presented scenarios. In section 4 we show how the behavioral matching is reduced to a graph matching problem; a similarity measure is defined based on the graph edit distance for which two new graph edit operations are introduced. Section 5 shows how the graph matching algorithm can be used for conversation protocol matchmaking. In section 6 we present an experimental study of the matchmaking algorithm. Finally section 7 present ongoing work and conclusions.

2 Motivating scenarios

In this section we present two scenarios requiring behavioral matchmaking. The first example situates in the context of web services integration and consists in retrieving services having compatible behavior. The second example is delta analysis which consists in finding differences between two models.

Web services integration Consider a company that uses service S to order office suppliers. Suppose that the company wants to find retailers (say WalMart or Target) having compatible web services (a new retailer or replacing the current partner). The allowed message exchange sequences are called conversation protocols and can be expressed for example using BPEL abstract processes, WSCL, or other protocol languages (see, e.g., [5]). The specification of the conversation protocol is important, as it rarely happens that service operations can be invoked independently from one another. Thus the company will search for a service having a compatible conversation protocol. Among retailer services, the most compatible one has to be found. If the service is not fully compatible, the company will adapt its service or will develop an adaptor in order to interact with the retrieved service. In both situations, the differences between the business protocols have to be automatically identified. In the former case, finding the most similar service allows to minimize the development cost. In the latter case, identifying automatically the differences between protocols is the first stage in the process of semi-automatically developing adapters (see [3]).

Delta analysis consists in finding the differences between two models. For example, the first one is the model specified by a standard and the second one is the model as it is implemented in an enterprise. Conversation definitions can be specified by industry specific standards groups in the same way that, for example, RosettaNet PIPs are specified by RosettaNet and used by participating enterprises. Enterprises need to verify if their services follow the guidelines prescribed by the standards. Thus, they need to compare the conversation model of their existing service with that prescribed by the standards. Ideally a tool should identify all the differences between the two models. Based on these differences the cost of reengineering of the existing service could be evaluated.

3 Related work

Currently, the algorithms for Web services discovery in registers like UDDI or ebXML are based on a search by key words or tables of correspondence of couples (key-value). Within the framework of the semantic Web, description logics were proposed for a richer and precise formal description of services. These languages allow the definition of ontologies, such as for example OWL-S, which are used as a basis for semantic matching between a declarative description of the required service and descriptions of the services offered ([16, 7, 6]). In [16, 6], a published service is matched with a required service when the inputs and outputs of the required service match the inputs and outputs of the published service (i.e., they have the same type or one is a generalization of the other). In [13], independent filters are defined for service retrieval: the name space, textual description, the domain of ontology that is used, types of inputs/outputs and constraints. The approach presented in [10] takes into account the operational properties like execution time, cost and reliability.

Service retrieval based of key words or some semantic attributes is not satisfactory for a great number of applications. The tendency of recent work is to exploit more and more knowledge on service components and behavior. The need to take into account the behavior of the service described by a process model was underlined by several researchers [22, 19, 7, 18, 24]. In [7], in order to improve precision of web service discovery, the process model is used to capture the salient behavior of a service. A query language for services is defined which allows to find services by specifying conditions on the activities which compose them, the exceptions treated, the flow of the data between the activities.

Very recently, authors in the academic world have published papers that discuss similarity and compatibility at different levels of abstractions of a service description (e.g., [4, 8, 11, 24]). In terms of protocols specification and analysis, existing approaches provide models (e.g., based on pi-calculus or state machines) and mechanisms to compare specifications (e.g., protocols compatibility checking).

In [24], authors give a formal semantics to business process matchmaking based on finite state automata extended by logical expressions associated to states. Computing the intersection is computationally expensive, and thus does not scale for large service repositories. To solve this problem,

the authors of [23] present an indexing approach for querying cyclic business processes using traditional database systems. The choice of finite state automata as a modelling formalism limits the expressiveness of the models, for instance representing parallel execution capabilities can lead to very large models.

A new behavior model for web services is presented in [21] which associates messages exchanged between participants with activities performed within the service. Activity profiles are described using OWL-S (Web Services Ontology Language). Web services are modelled like nondeterministic finite automatons. A new query language is developed that expresses temporal and semantic properties on services behaviors.

To summarize, the need to take into account the service behavior in the retrieval process was underlined by several authors and some very recent proposals exist ([21],[24]). The few approaches that exist give a negative answer to the user if a model satisfying exactly his requirements does not exist in the registries, even if a model that requires a small modification exists. Our objective is to propose an approach for service retrieval based on behavioral specification allowing an approximate match. To the best of our knowledge, there is not another approach allowing to retrieve services having similar behavior and defining a behavior-based similarity measure.

4 A graph-based approach to behavior matchmaking

In this section we show how the behavioral matching is reduced to a graph matching problem. Section 4.1 recalls the principles of the graph matching method that we use, the error correcting subgraph isomorphism, which is based on the idea of graph edit operations. Next sections show how we adapt it to our problem: we extend the set of graph edit operations and we define a similarity measure for behavior matchmaking.

A conversation protocol describes the observable behavior of a web service. It complements the web service interface definition by imposing constraints on the order of exchanged messages. Most of existing proposals (standards and research models) are graph based. For this reason, we choose to use a graph representation of conversation protocols in order to compare two models.

Using graphs as a representation formalism for both user requirements and service models, the service matching problem turns into a graph matching problem. We want to compare the process graph representing user requirements with the model graphs in library. The matching process can be formulated as a search for graph or subgraph isomorphism. However, it is possible that it does not exist a process model such that an exact graph or subgraph isomorphism can be defined. Thus, we are interested in finding process models that have similar structure if models that have identical structure do not exist. The error-correcting graph matching integrates the concept of error correction (or inexact matching) into the matching process ([20, 9]). To make the paper self-contained, in the next section we briefly recall the principle of this graph matching method and the basic definitions as given in [14].

4.1 Background and basic definitions

In order to compare the model graphs to an input graph and decide which of the models is most similar to the input, it is necessary to define a distance measure for graphs. Similar to the string matching problem where edit operations are used to define the string edit distance, the subraph edit distance is based on the idea of edit operations that are applied to the model graph. Edit operations are used to alter the model graphs until there exist subgraph isomorphism to the input graph. For each edit operation, a certain cost is assigned. The costs are application dependent and reflect the likelihood of graph distortions. The more likely a certain distortion is to occur the smaller is its cost. The subgraph edit distance from a model to an input graph is then defined to be the minimum cost taken over all sequences of edit operations that are necessary to obtain a subgraph isomorphism. It can be concluded that the smaller the subgraph distance between a model and an input graph, the more similar they are.

In the following we give the definitions of error correcting graph matching as given in [14].

A directed labelled graph is defined by a quadruple $G = (V, E, \alpha, \beta)$ where V is the set of vertices, $E \subset V \times V$ is the set of edges, $\alpha : V \rightarrow L_V$ is the vertex labelling function and $\beta : E \rightarrow L_E$ is the edge labelling function.

Definition 4.1 Graph isomorphism *Let g and g' be graphs. A graph isomorphism between g and g' is a bijective mapping $f : V \rightarrow V'$ such that*

- $\alpha(v) = \alpha'(f(v))$ for all $v \in V$

- for any edge $e = (u, v) \in E$ there exists an edge $e' = (f(u), f(v)) \in E'$ such that $\beta(e) = \beta'(e')$ and for any edge $e' = (u', v') \in E'$ there exists an edge $e = (f^{-1}(u'), f^{-1}(v')) \in E$ such that $\beta(e) = \beta'(e')$.

If $f : V \rightarrow V'$ is a graph isomorphism between graphs g and g', and g' is a subgraph of another graph g", i.e. $g' \subset g$", then f is called a subgraph isomorphism from g to g".

Given a graph G, a graph edit operation δ on G is any of the following:

∘ substituting the label $\alpha(v)$ of vertex v by l

∘ substituting the label $\beta(e)$ of edge e by l'

∘ deleting the vertex v from G (for the correction of missing vertices). Note that all edges that are incident with

the vertex v are deleted too.

◦ deleting the edge e from G (for the correction of missing edges).

◦ inserting an edge between two existing vertices (for the correction of extraneous edges).

Definition 4.2 Edited graph *Given a graph and an edit operation δ , the edited graph $\delta(G)$ is a graph in which the operation δ was applied. Given a graph G and a sequence of edit operations $\Delta = (\delta_1, \delta_2, \cdots \delta_k)$, the edited graph $\Delta(G)$ is a graph $\Delta(G) = \delta_k(\cdots \delta_2(\delta_1(G)))..)$.*

Definition 4.3 Ec-subgraph isomorphism *Given two graphs G and G', an error correcting (ec) subgraph isomorphism f from G to G' is a 2-tuple $f = (\Delta, f_\Delta)$ where Δ is a sequence of edit operations and f_Δ is a subgraph isomorphism from $\Delta(G)$ to G'.*

For each edit operation δ, a certain cost is assigned $C(\delta)$. The cost of an ec-subgraph isomorphism $f = (\Delta, f_\Delta)$ is the cost of the edit operations Δ, i.e., $C(\Delta) = \sum_{i=1}^{k} C(\delta_i)$. Usually, there is more than one sequence of edit operations such that a subgraph isomorphism from $\Delta(G)$ to G' exists and, consequently, there is more than one ec-subgraph isomorphism from G to G'. We are interested in the ec-subgraph isomorphism with minimum cost.

Definition 4.4 Subgraph edit distance *Let G and G' be two graphs. The subgraph distance from G to G', $ed(G, G')$ is given by the minimum cost taken over all error-correcting subgraph isomorphism f from G to G'.*

4.2 Extension of the sub-graph edit distance

The models to be compared can have different granularity levels for achieving the same functionality. For example, the first service has a single operation (activity) to achieve certain functionality, while in the second service the same behavior is achieved by composing several operations. Thus, new edit operations are required. Given a graph G, we extend the definition of edit operation δ on G by adding two operations:

◦ decomposing a vertex v into two vertices v_1, v_2

◦ joining two vertices v_1, v_2 into a vertex v.

We limit ourselves to a simple case of decomposition, when a vertex is decomposed into a sequence of two vertices. This simple type of decomposition is sufficient for applications that we analyzed. A more general decomposition operation would be to decompose a vertex into a connected subgraph, this is subject of future work.

The operation of decomposing a vertex v into two vertices v_1, v_2 is executed in the following way :

- all the edges having as destination the vertex v will have as destination the vertex v_1;

- all edges having as source the vertex v, will have as source the vertex v_2;

- an edge between the vertex v_1 and v_2 will be added.

The joining operation is executed in a similar way. These two new edit operations allow to model one-to-many dependencies among vertices of two graphs (i.e., a vertex in one graph correspond to two vertices in the second graph). The classical edit operations take into account only one-to-one mappings between vertices of the two graphs. For example, if a vertex v in the first graph corresponds to the composition of two vertices in the second graph (v_1 followed by v_2), a matching algorithm based on the classical edit distance would map v to v_1 and suppress v_2. It would not be possible to discover that v is mapped to a composition of v_1 and v_2.

4.3 Similarity measure for behavioral matching

The subgraph edit distance defined previously expresses the cost of transformation needed to adapt the model graph in order to cover a subgraph in the input model. This distance is asymmetric, it represents the distance from the model graph to the input graph. In order to rank the model graphs, the similarity measure has to take into account the number of vertices in the input graph that were covered by the model graph. If two model graphs have the same subgraph distance to the input graph but are matched to subgraphs with different number of nodes, the one that matches a subgraph with more nodes will be preferred.

For this reason, we propose to calculate the similarity measure based on the total edit distance between the two graphs. The total distance between the model and the input graph is defined as the sum of the subgraph edit distance and the cost of adding the nodes of the input graph not covered by the ec subgraph isomorphism.

5 Conversation protocol matchmaking

In this section we illustrate the use of the error-correcting graph matching algorithm for conversation protocol matchmaking. We first give an overview of the matchmaking process and then we discuss each step in detail; finally, we illustrate it using an example.

We choose to exemplify our approach for business protocol matchmaking by using the WSCL model. The same approach can be applied for other models, as long as the conversation protocol can be transformed to a graph in a unique way (equivalent representations of a conversation protocol are reduced to the same process graph). WSCL is a simple conversation definition language, which offers the basic constructs to model the sequencing of the interactions or operations of one interface. It thus complements the interface definition by specifying the invocations order

of the operations. A conversation in WSCL is specified using the following basic constructs [2]:

◦ Interactions model the actions of the conversation as document exchanges between two participants. WSCL supports five types of interactions: *Send* (the service sends out an outbound document); *Receive* (the service receives an inbound document); *SendReceive* (the service sends out an outbound document and then expects to receive an inbound document in reply); *ReceiveSend* (the service receives an inbound document and then sends out an outbound document); *Empty* (does not contain any documents exchanged, but is used only for modelling the start and end of a conversation.)

◦ Transitions specify the ordering relationships between interactions.

Each interaction specifies the type (schemas) of XML document that is expected as input or is produced as output.

The conversation protocol matchmaking process is composed of the following steps. First, the conversations protocols to be compared are transformed to graphs. Next, the graphs are expanded in order to have the same level of granularity in both graphs and the error corecting graph matching algorithm is applied ([14]). The similarity analyzer module evaluates the similarity between the graphs.. Finally, the granularity levels are compared and the costs corresponding to identified differences are added to the total distance.

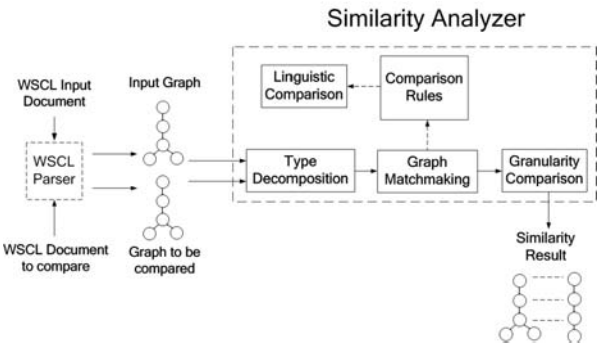

Figure 1. Architecture

The architecture of the behavior matchmaking system is presented in Figure 1. The system is composed of a parser and a similarity analyzer module. The parser transforms a WSCL conversation model into a graph whose vertices represent interactions and whose edges represent transitions. Each vertex has the the following attributes: name, interaction type and documents.

The similarity analyzer module evaluates the similarity between the graphs. In the next sections we present in detail the functionalities of its modules (excepting the graph matchmaking module that implements the algorithm described in [14]; for lack of space we refer the reader to [14]).

5.1 Decomposing interactions

After transforming conversation protocols to graphs, the second step in the behavior matching is graphs expansion. The decomposition operations are applied in order to have the same granularity level in both models. The decomposition operation depends on the metamodel of the protocols to be matched. For instance, for WSCL metamodel, it is possible that in one protocol an interaction is modelled as a *SendReceive* interaction, while in the second protocol the same functionality is achieved by having a *Send* interaction followed by a *Receive* interaction. Thus, the decomposition module will transform interactions of type *SendReceive* or *ReceiveSend* in atomic interactions: *Send* and *Receive*. A *SendReceive* interaction is decomposed into a *Send* interaction followed by a *Receive* interaction in the following way:

- all edges having as destination the *SendReceive* interaction will have as destination the *Send* interaction

- all edges having as source the *SendReceive* interaction, will have as source the *Receive* interaction

- an edge will be added from the *Send* interaction to the *Receive* interaction

- if the *SendReceive* interaction has outbound document a and inbound document b , then the *Send* interaction will have a as outbound document and the *Receive* interaction will have b as inbound document.

In a similar way, a *ReceiveSend* interaction is decomposed into a *Receive* interaction followed by a *Send* interaction.

This decomposition function is specific to WSCL model. For other applications, user can specify a different decomposition function. The decomposition function has always the same signature: it takes as argument a vertex and returns two vertices resulting from decomposition (that are supposed to be sequential). The function behavior is specific to the application (metamodel of the protocols to be matched) and consists in specifying how the labels and attributes of the two vertices are obtained from the decomposed vertex.

5.2 Comparison rules

The *Comparison rules module* contains all the application-dependent functions allowing to calculate the cost of graph edit operations. These functions are used by the graph matching module for calculating the distance between the graphs. In order to support applications with specialized cost function, user-defined cost function can be registered in this module. In the following we explain the cost function used for conversation protocol matchmaking.

The cost for inserting, suppressing edges and vertices can be set to a constant. The cost for editing a vertex is calculated by function VertexMatch (see Algorithm 1). As

Algorithm 1 Function VertexMatch

INPUTS: (Nodei,Nodej)
Nodei: Struct (Idi,Typei,Di), Nodej: Struct (Idj,Typej,Dj)
OUTPUT: $DistanceNode$

if $Typei \neq Typej$ (different types) **then**
  Return $DistanceNode = 1$
else
  Calculate document sets similarity $TotalSD$
  if $TotalSD > 0$ **then**
    Calculate Ids similarity $SimId = LS(Idi, Idj)$

$$DistanceNode = 1 - \frac{w_d * TotalSD + w_i * SimId}{w_d + w_i}$$

    Return $DistanceNode$
  else
    Return $DistanceNode = 1$
  end if
end if

vertices represent WSCL interactions, this cost expresses the distance between two WSCL interactions. Each interaction has a label (Id) and two attributes: the interaction type ($Type$) and documents set (D) (in or outbound documents). The matchmaking gives priority to type comparison, and if two interactions have the same type, it compares the similarity of the set of documents $TotalSD$; if there is a similarity between them ($TotalSD > 0$), it calculates the similarity of the interactions names ($SimId$).

The function $SD(Di, Dj)$ where Di, Dj is the set of documents of $Nodei$ and $Nodej$ respectively, computes the best mapping that can be obtained between the documents of the two sets.

$$\begin{cases} SD(Di, Dj) = \\ Max(SD(Di - I, Dj - J) + LS(I, J), & Di \neq \phi, Dj \neq \phi, \\ & I \in Di, J \in Dj \\ 0, & Di = \phi \vee Dj = \phi \end{cases}$$

The number of mappings established is $Min(|Di|, |Dj|)$. Function LS calculates the linguistic similarity between document names and is explained in the next section.

Finally, the total similarity of the document sets is:

$$TotalSD = \frac{SD(Di, Dj)}{k}$$

where, k = No of documents of set Di.

Weights w_d and w_i indicate the contribution of $TotalSD$ (similarity of documents being exchanged) and $SimId$ (similarity of interaction names) respectively in the total DistanceNode score ($0 \leq wd \leq 1$ and $0 \leq wi \leq 1$).

5.3 Linguistic comparison

The *Linguistic comparison module* calculates the linguistic similarity between two labels based on their names [17]. The labels are often formed by a word or by a combination of words and can contain abbreviations. To obtain a linguistic distance between two strings, we use existing algorithms: *NGram*, *Check synonym*, and *Check abbreviation*. The *NGram* algorithm estimates the similarity according to a number of common *qgrams* between labels names [1]. The *Check synonym* algorithm uses a linguistic dictionary (e.g. Wordnet [15] in our implementation) to find out the synonyms between the labels names while the *Check abbreviation* one uses an abbreviation dictionary according to the application domain.

If all the algorithms return 1, there is an exact matching. On the other hand, if all the algorithms return 0, it means that there are no matching between labels. If the *NGram* value and the *Check abbreviation* value are equal to 0, and *Check Synonym* is between 0 and 1, the total linguistic similairty value will be equal to the *Check Synonym* one. Finally, if the three algorithms values are between 0 and 1, the similarity LS ([17]) is the average of them:

$$LS = \begin{cases} 1 & if\ (m1 = 1 \vee m2 = 1 \vee m3 = 1) \\ m2 & if\ (0 < m2 < 1 \wedge m1 = m3 = 0) \\ 0 & if\ (m1 = m2 = m3 = 0) \\ \frac{m1+m2+m3}{3} & if\ m1, m2, m3 \in (0, 1) \end{cases}$$

where, m1 = Sim(NGram), m2=Sim(Synonim Matching) and m3= Sim(Abbreviation Expansion).

5.4 Comparison of granularity level

The ecgm (error correcting graph matching) is applied to graphs that were expanded, i.e., contain only atomic *Send* or *Receive* interactions. The granularity comparison module checks whether the interactions that were mapped by the ecgm algorithm have the same granularity level in both models. For instance, suppose that in the input graph we have a *SendReceive* interaction. This was decomposed by the decomposition module in a *Send* interaction followed by a *Receive* interaction that were mapped with two corresponding interactions in the model graph (by the ecgm algorithm). If these interactions were atomic in the model graph, the cost of joining operation has to be added to the total graph distance (line 5 in the table of Figure 2).

The costs for granularity differences that have to be taken into account for the total distance graph for all cases of figure (atomic versus non atomic interactions in the model and input graph) are summarized in the Figure 2. For the sake of clarity, the table does not present the cases for interactions that have no correspondence in the other graph. If the mapped interactions have the same granularity level (they are both atomic or non atomic) there is no cost to be added to the subgraph edit distance.

Interaction type of input graph	Interaction type of model graph	Granularity Diff. Cost
S atomic	S atomic	0
R atomic	R atomic	0
SR	SR	0
RS	RS	0
SR (or RS)	S atomic + R atomic	c_j
SR (or RS)	S nonat. + R nonat.	$c_d + c_j$
SR (or RS)	S nonat. + R atomic or S atomic + R nonat.	$c_d/2 + c_j$
S atomic	S nonatomic	$c_d/2$
R atomic	R nonatomic	$c_d/2$

S=Send, R=Receive, SR= SendReceive, RS=ReceiveSend

Figure 2. Cost for granularity differences

A more complicated case (line 7 in the table of Figure 2) is when a *SendReceive* interaction SR_I in the input graph is mapped with an atomic *Send* interaction S_M followed by a *Receive* interaction R_M that is non atomic (belongs to a *SendReceive* SR_M) in the model graph. In this case, the cost is $c_j + c_d/2$ (c_j = cost of joining S_M and R_M; $c_d/2$ = cost for obtaining R_M by decomposing SR_M interaction in the model graph).

5.5 Example

Suppose that we would like to find the similarity between two hotel reservation services whose conversations have been described using WSCL. The first service has the following conversation: *ReservationRequest* interaction (Type: Receive), *RequestCatalog* (Type: Receive) followed by *SendCatalog* (Type: Send) and *CheckAvailability* (Type: ReceiveSend). The second service conversation protocol has the following sequence of interactions : *ReservationRQ* (Type: Receive), *Availability* (Type: ReceiveSend), *Catalog* (Type: ReceiveSend). Our system converts each service WCSL document into a graph (input graph and model graph, Figure 3). Next, the graphs are decomposed according to the interaction type (Decomposed input graph and model graph, Figure 3) using the decomposition function. The dotted lines in Figure 3 represent the mappings found by the system between the two graphs using the comparison rules. Finally, the cost for granularity differences is added to the total graph distance. (*Catalog* Interaction in the model graph has to be decomposed into two interactions to match the input graph).

In conclusion, the edit script will show that the two graphs are similar, but have the following structural differences : interactions *RequestCatalog* (Type: Receive) and *SendCatalog* (Type: Send) are modelled as a single *SendReceive* interaction in the model graph; interactions for checking the availability and for asking the catalog are executed in different order in the two models.

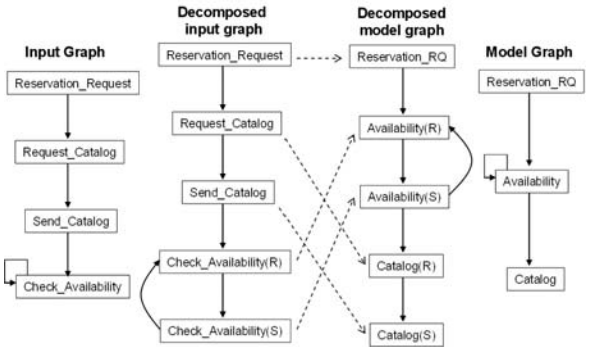

Figure 3. Example

6 Implementation and experiments

In this section, we present an experimental study of the matchmaking algorithm. The theoretical complexity of the graph matchmaking algorithm [14] is ($O(m^2n^2)$ in the best case (when the distance between the model and the input graph is minimal) and $O(m^n n)$ in the worst case (m = the total number of vertices in the input graph; n = the total number of vertices in the graph to be compared). The goal of the experiments is to find how well performs the algorithm for conversation protocol matchmaking. Since most of the conversation protocols have less than 50 interactions, we considered a maximum of 100 interactions.

We implemented a system having the architecture presented in the previous section. The prototype is available at `http://200.21.83.91:8080/matching/input.faces`. The system is also available as a web service that takes as input two WSCL files and calculates the similarity between them (`http://200.21.83.91:8080/matching/services/matching?wsdl`). It returns also the script of edit operations required in order to transform the first conversation protocol to conform with the second one.

The figure 4 shows the system behavior for two graphs with different structures (different edge number and loops) and different values for the identifier of each interaction. For the comparison of the identifiers and documents names, the linguistic comparison is used.

Despite the exponential theoretical cost, the graphic shows that the matchmaking algorithm can be used for WSCL documents having less than 50 interactions.

7 Conclusion

In this paper we proposed a solution for service retrieval based on behavioral specification. First we motivated the

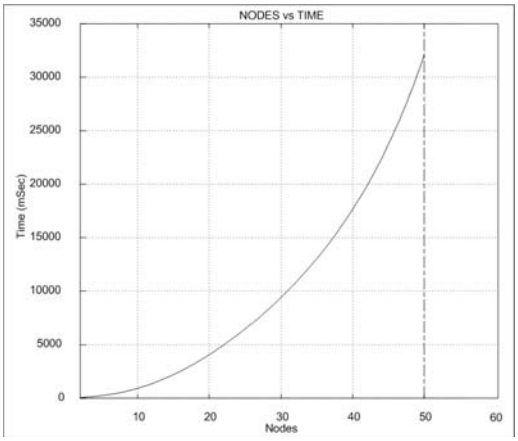

Figure 4. Matchmaking two WSCL documents

need to retrieve services based on their conversation model. By using a graph representation formalism for services, we proposed to use a graph error correcting matching algorithm in order to allow an approximate matching. Starting from the classical graph edit distance, we proposed two new graph edit operations to take into account the difference of granularity levels that could appear in two models. We exemplified our approach for behavior matching for conversation protocols expressed using the WSCL model and we developed a protoype that is available as a web service.

The next step of this work will be to address the problem of comparing a conversation protocol with a set of protocols in a library. Performance issues related to the execution time have to be addressed. We will also experimentally evaluate the performance of the behavior based retrieval method in terms of precision and recall.

8 Acknowledgements

Juan Carlos Corrales is an Alban Program Fellowship recipient (High-level scholarship program for Latin America, http://www.programalban.org).

References

[1] R. C. Angell, G. E. Freund, and P. Willett. Automatic spelling correction using a trigram similarity measure. *Information Processing and management*, 19(4):255–261, 1983.

[2] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, and S. Williams. Web services conversation language (wscl) 1.0. In *W3C*, 2002.

[3] B. Benatallah, F. Casati, D. Grigori, H. R. Motahari Nezhad, and F. Toumani. Developing adapters for web services integration. In *Proc. of CAISE*, 2005.

[4] B. Benatallah, F. Casati, and F. Toumani. Analysis and management of web services protocols. In *Proc. of ER*, 2004.

[5] B. Benatallah, F. Casati, and F. Toumani. Web services conversation modeling: A cornerstone for e-business automation. *IEEE Internet Computing*, 2004.

[6] B. Benatallah, M. Hacid, C. Rey, and F. Toumani. Semantic reasoning for web services discovery. In *Proc. of ESSW*, 2003.

[7] A. Bernstein and M. Klein. Towards high-precision service retrieval. In *Proc. of ISWC*, 2002.

[8] L. Bordeaux and et al. When are two web services compatible? In *Proc. of TES*, 2004.

[9] H. Bunke. Recent developments in graph matching. In *Proc. of ICPR*, pages 117 – 124, 2000.

[10] J. Cardoso and A. Sheth. Semantic e-workflow composition. *Journal of Intelligent Information Systems*, 21:191–225, 2003.

[11] L. Dong, A. Halevy, J. Madhavan, E. Nemes, , and J. Zhang. Similarity search for web services. In *Proc. of VLDB*, 2004.

[12] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proc. of Ssdbm*, 2002.

[13] T. Kawamura, J. De Blasio, T. Hasegawa, M. Paolucci, and K. Sycara. A preliminary report of a public experiment of a semantic service matchmaker combined with a uddi business registry. In *Proc. of ICSOC*, 2003.

[14] B. Messmer. *Graph Matching Algorithms and Applications*. PhD thesis, University of Bern, 1995.

[15] G. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[16] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proc. of ISWC*, 2002.

[17] A. Patil, S. Oundhakar, A. Sheth, and K. Verna. Meteor-s web service annotation framework. In *Proc. of WWW Conference*, 2004.

[18] G. Piccinelli, G. Di Vitantonio, and L. Mokrushin. Dynamic service aggregation in electronic marketplaces. *Computer Networks*, 2(37), 2001.

[19] S. S. Bansal and J. M. Vidal. Matchmaking of web services based on the DAML-S service model. In *Proc. of AAMAS*, pages 926–927, 2003.

[20] L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 3, 1981.

[21] Z. Shen and J. Su. Web services discovery based on behavior signatures. In *Proc. of IEEE SCC*, 2005.

[22] D. Trastour, C. Bartolini, and J. Gonzalez-Castillo. A semantic web approach to service description for matchmaking of services. In *Proc. of SWWS*, 2001.

[23] A. Wombacher, B. Mahleko, and P. Fankhauser. A grammar-based index for matching business processes. In *Proc. of ICWS*, pages 21–30, 2005.

[24] A. Wombacher, B. Mahleko, P. Fankhauser, and E. Neuhold. Matchmaking for business processes based on choreographies. In *Proc. of EEE*, 2004.