BPEL Processes Matchmaking for Service Discovery

Juan Carlos Corrales, Daniela Grigori, and Mokrane Bouzeghoub

Prism, Universite de Versailles Saint-Quentin en Yvelines
45 avenue des Etats-Unis, 78035 Versailles Cedex, France
Juan-Carlos.Corrales-Munoz@prism.uvsq.fr,
Daniela.Grigori@prism.uvsq.fr,
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. 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 which takes as input two BPEL models 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 [1]. 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 it out again or to retrieve data generated previously.

R. Meersman, Z. Tari et al. (Eds.): OTM 2006, LNCS 4275, pp. 237–254, 2006.

– 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 extension 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 graph edit distance for which two new graph edit operations are introduced. We show also how to combine two graph models in order to satisfy user requirements. Section 5 shows how the graph matching algorithm can be used for BPEL process 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 of 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 business protocols and can be expressed for example using BPEL abstract processes, WSCL, or other protocol languages (see, e.g., [2]). The specification of the business 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 business 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. 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. Business 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 business 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 ([4,5,6]). In [4,6], 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 [7], 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 [8] takes into account the operational properties like execution time, cost and reliability. The authors of [9] provide a lightweight semantic comparison of interfaces based on similarity assement methods (lexical, attribute, interface and QoS similarity).

In the context of the Grid [1], the search of procedures is based on a high-level language which expresses the relationships among procedures and their input/output data.

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 [10,11,5,12,13,14]. In [5], 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.

In [11], authors argue that the matchmaking based on service input and output is not sufficient as some output data may be produced only under certain internal conditions. Thus, they propose an algorithm that matches output data taking into account the process structure, for instance conditional branching.

In [10], authors underline the importance of including behavior aspects in matchmaking process in the B2B environment and mention it as a future work. The authors of [13], which propose a model for dynamic service aggregation, stress also the capability to automatically verify the behavioral compatibility of various processes as a requirement in electronic marketplaces.

In [12], authors deal with the equivalence of two processes modelled using Petri nets. It is supposed that partners discover each other by searching in business registry, and then agree on a common protocol. Their work verifies the compatibility between the agreed protocol and the process existing in the enterprise. We take a different approach, by allowing to find a partner that is fully or partially compatible to an existing enterprise process.

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., [15,16,17,14]). 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 [14], 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 [18] present an indexing approach for querying cyclic business processes using traditional database systems; they introduce an abstraction function that removes cycles and transforms a potentially infinite set of message sequences into a finite representation, which can be handled by existing 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 [19] 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 non-deterministic 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 ([19],[14]). 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, we define a similarity measure for behavior matchmaking and we show how to compose two library graphs to satisfy user requirements.

A business 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 business 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 there 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 ([20,21]). 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 [22].

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 [22].

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 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 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 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. 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 corresponds 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 ($v1$ followed by $v2$), a matching algorithm based on the classical edit distance would map v to $v1$ and suppress $v2$. It would not be possible to discover that v is mapped to a composition of $v1$ and $v2$.

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.

Depending on the application, the similarity measure can be defined in different ways. The total distance between the two graphs can be 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. The second possibility is to define it as the subgraph edit distance (ed) divided by the number of nodes of model graph (N_M): $D = ed/N_M$. The similarity measure is the inverse of this distance ($S = 1/D$).

4.4 Composing Fragments to Match the Input Graph

If two models are matched into 2 subgraphs of the input model that are disjoint (the set of nodes are disjoint), then it is possible to combine them to form a graph that will be matched to a larger subgraph of the input graph.

Suppose two model graphs G_1 and G_2 that are disjoint. Let $f_1 = (\Delta_1, f_{\Delta_1})$ and $f_2 = (\Delta_2, f_{\Delta_2})$ be two ec-subgraph isomorphism from G_1 and G_2 to G_I, respectively. The problem is to find an ec-subgraph isomorphism from $G = G_1 \cup G_2$ to G_I that is based on f_1 and f_2. f_1 and f_2 can be combined if no two vertices in $\Delta(G_1)$ and $\Delta(G_2)$ are mapped into the same input vertex. More precisely, the intersection of the images of f_{Δ_1} and f_{Δ_2} must be empty, i.e., $f_{\Delta_1}(V_1) \cap f_{\Delta_2}(V_2) = \emptyset$.

The construction of an ec-subgraph isomorphism $f = (\Delta, f_\Delta)$ from f_1 and f_2 requires that a set of edit operations Δ and a subgraph isomorphism f_Δ are generated on the basis of Δ_1, Δ_2, and f_{Δ_1}, f_{Δ_2} respectively such that Δ is a subgraph isomorphism from $(G_1 \cup G_2)$ to G_I. Let $f_1 = (\Delta_1, f_{\Delta_1})$ and $f_2 = (\Delta_2, f_{\Delta_2})$, $\Delta_1(G_1) = (V_{\Delta_1}, E_{\Delta_1}, \alpha_{\Delta_1}, \beta_{\Delta_1})$. Then :

$$f_\Delta(v) = \begin{cases} f_{\Delta_1}(v)\ if\ \ v \in V_{\Delta_1} \\ f_{\Delta_2}(v)\ if\ \ v \in V_{\Delta_2} \end{cases}$$

and $\Delta = \Delta_1 + \Delta_2 + \Delta_E$. Δ_E is constructed as follows. For each pair $(v_i, w_j), v_i \in V_1, w_j \in V_2$, if there is an edge $e_I = (f_{\Delta_1}(v_i), f_{\Delta_2}(w_j)) \in G_I$, then an edge (v_i, w_j) must be inserted in Δ_E.

If C_1 and C_2 are the cost of the ec-subgraph isomorphism f_1 and f_2 respectively, then the cost of the ec-subgraph isomorphism f is : $C(f) = C_1 + C_2 + C(\Delta_E)$.

To summarize, if two models G_1 and G_2 cover disjoint subgraphs in the input graph, it is possible to construct a model that is their composition in order to find a better match for the input graph.

5 BPEL Processes Matchmaking

In this section we illustrate the use of the error-correcting graph matching algorithm for BPEL processes 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 BPEL model. The same approach can be applied for other models, as long as the business protocol can be transformed to a graph in a unique way (equivalent representations of a business protocol are reduced to the same process graph).

BPEL [23] has emerged as a standard for specifying and executing web services-based processes. It supports the modelling of two types of processes: executable and abstract processes . An *abstract process* is a business protocol, specifying the message exchanges between different parties from the perspective of a single organization (or composite service), whitout revealing the internal behavior. An *executable process*, in contrast, specifies the actual behavior of a participant. A BPEL process is composed of a set of activities, that can be either primitive or structured. Primitive activities include operations involving web services like the *invoke*, the *receive* and the *reply* activity. There are further activities for assigning data values for variables (*assign*) or *wait* to halt the process for a certain time interval. Structured activities are used for defining the control flow, e.g. to specify concurrency of activities using *flow*, alternative branches (*switch*) or sequential execution (*sequence*). These structured activities can be nested. Beyond that, *links* can be used to specify order constraints between activities composing a *flow*, similar to control flow arcs.

BPEL builds on IBM's WSFL and Microsoft's XLANG and combines thus the features of a block structured language (XLANG) with those for directed graphs (WSFL). As a result there are sometimes two equivalent ways to implement a desired behavior. For exemple, a sequence can be realised using a *sequence* or a *flow* with a link between

activities, a choice based on certain data values can be realised using the switch or flow elements, etc.

For this reason, we transform a BPEL process to a process graph and thus equivalent constructs that are syntactically different will be transformed to the same graph fragment. In the following we concentrate on the matchmaking of BPEL abstract processes, but the approach can be adapted to matching executable processes.

The BPEL matchmaking process is composed of the following steps. First, the BPEL processes to be compared are transformed to graphs. Next, the graph matching algorithm is applied taking into account the comparison rules and possibly the nodes decomposition. Finally the similarity result is shown.

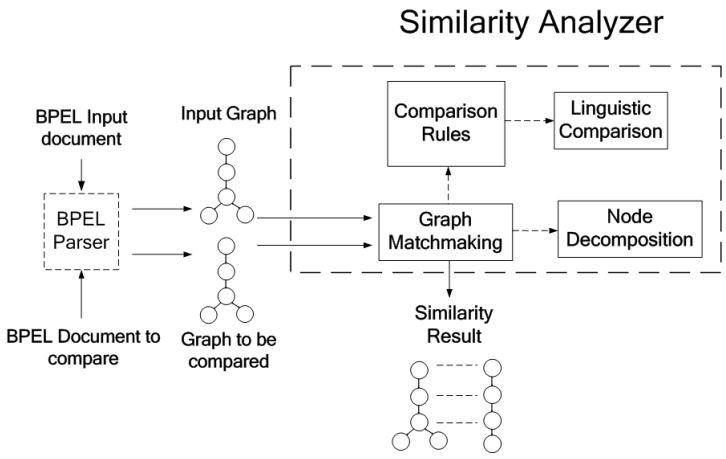

Fig. 1. Architecture

The architecture of the behavior matchmaking system is presented in Figure 1. The system is composed of a parser that transforms a BPEL document into a graph and a similarity analyzer module that evaluates the similarity between the graphs. The similarity analyzer is composed of the following elements (Figure 1):

- *Graph matchmaking*, that takes as inputs the graphs produced by the BPEL parser and finds out the error correcting sub-graph isomorphism with minimal cost.
- *Comparison rules module*, that groups the cost functions for the graph edit operations.
- *Decomposition module*, that applies the decomposition operation if it is necessary in order to have the same level of granularity in both models.
- *Linguistic comparison*, that implements different algorithms useful to find the similarity between two strings.

In the next sections we present in detail the functionalities of each module.

5.1 Transforming BPEL to Graph

The parser function transforms a behavior model into a process graph. A process graph has at least one start nodes and can have multiple end nodes. The graph has two kind of nodes : regular nodes representing the activities and connectors representing split and join rules of type XOR or AND. Nodes are connected via arcs which may have an optional guard. Guards are conditions that can evaluate to true or false.

We implemented the flettening strategy presented in [24] to transform a BPEL to a process graph. The general idea is to map structured activities to respective process graph fragments. The algorithm traverses the nested structure of BPEL control flow in a top-down manner and applies recursively a transformation procedure specific to each type of structured activity.

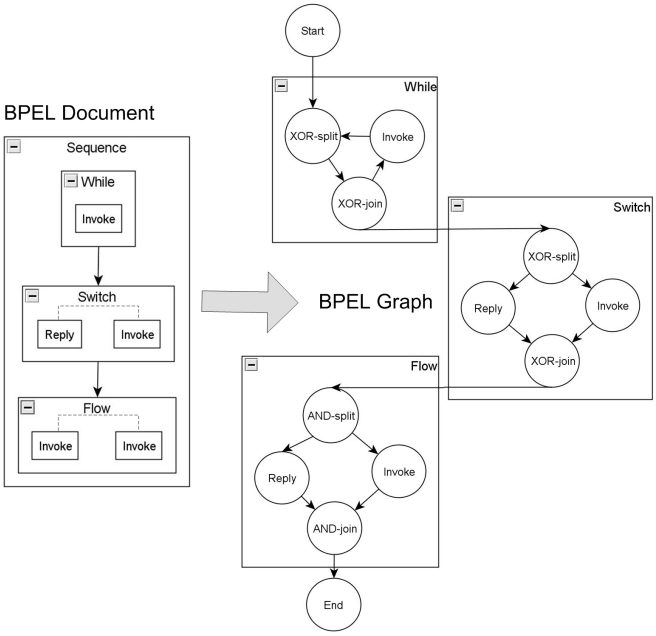

Fig. 2. Correspondences between BPEL elements and graph elements

A *BPEL basic activity* is transformed to a node. The *sequence* is transformed by connecting all nested activities with graph arcs; each sub-activity is then transformed recursively. For the *while* activity a loop is created between an XOR join and an XOR split, the condition is added to the edge. The graph representation of *switch* consists of a block of alternative branches between an XOR split and an XOR join. The branching conditions are associated to edges. The *flow* is transformed to a block of parallel branches starting with an AND split and synchronized with an AND join.

The nodes that represent the activities have the following attributes: Operation and PortType. The connector nodes have two attributes: ConnectorType (AND-split,

AND-join, XOR-split, XOR-join) and ActivityType (the BPEL structured activity from which it was transformed). Figure 2 shows the correspondence between BPEL constructs and graph elements.

5.2 Graph Matchmaking

This module implements the algorithm for error-correcting sub-graph isomorphism detection ([22]). The sub-graph isomorphism detection is based on a state-space searching using an algorithm similar to A* [20]. The basic idea of a state-space search is to have states representing partial solutions of the given problem and to define transitions from one state to another, thus, the latter state represents a more complete solution than the previous state. For each state s there is an evaluation function $f(s)$ which describes the quality of the represented solution. The states are expanding themselves according to the value of f. In the case of each sub-graph isomorphism detection, given a model graph G and an input graph G_I, a state s in the search space represents a partial matching from G to G_I. Each partial matching implies a number of edit operations and their cost can be used to define the evaluation function $f(s)$. In other words, the algorithm starts by mapping the first node of G with all the nodes of G_I and chooses the best mapping (with minimal cost)(Algorithm1, line 1). This represents a partial mapping that will be extended by adding one node at a time (line 7). The process terminates when either a state representing an optimal ec-subgraph isomorphism from G to G_I has been reached or all states in the search space have edit costs that exceed a given acceptance threshold.

Algorithm 1. Error-correcting sub-graph isomorphism detection $(G(V), G_I(V_I))$

1: Initialize OPEN: map the activity node in V onto each activity node in V_I (call Activity-Match), i.e. create a mapping p. Calculate the cost of this mapping C(p) and add p to OPEN.
2: IF OPEN is empty THEN Exit.
3: Select p of OPEN such that C(p) is minimal and remove p from OPEN
4: IF C(p)> Accept threshold THEN Exit.
5: IF p represents a complete mapping from G to G_I THEN output p. Set accept threshold = C(p). Goto 2.
6: Let p be the current mapping that maps k nodes from G.
7: FOR *each activity node w in V_I that has not yet mapped to a corresponding node in V*
  7.1: extends the current mapping p to p' by mapping the k+1 node of V to w and calculate the cost of this mapping C(p')
  7.2: add p' to OPEN
8: Goto 2

The cost of the mapping $C(p')$ (line 7.1) represents the cost of extending the current mapping p with the next node in the model graph. Extending the mapping by mapping a vertex v (in the input graph that has not yet mapped) to a vertex w in the model graph (that does not belong to the current mapping) implies node edit operation and edge edit operations. First, the label and attributes of v must be substituted by label attributes of

w, and secondly, for each mapping $(v', w') \in p$ it must be ensured that any edge (v', v) in the model graph can be mapped to an edge (w', w) in the input graph by means of edge edit operations.

A process graph has two kind of nodes: activities and connectors. In contrast with activities, connectors do not represent business functions, they express control flow constraints. For this reason, in the matching process, we compare them in a manner similar to edges. That is, when mapping edges between two activity nodes, we map also the possible connectors binding directly to the nodes and calculate the corresponding edit cost.

5.3 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 functions used for BPEL protocol matchmaking. The cost for inserting, suppressing edges and vertices can be set to a constant. The cost for editing vertices (basic activities and connectors) are presented below.

Algorithm 2. Function BasicActivityMatch

INPUTS: (Nodei,Nodej)
Nodei: Struct (Opi, PTi), Nodej: Struct (Opj,PTj)
OUTPUT: $DistanceNode$

Calculate Operation Similarity $SimOperation = LS(Opi, Opj)$
if $SimOperation = 0$ (different Operations) **then**
  Return $DistanceNode = 1$
else
  Calculate PortType Similarity $SimPortType = LS(PTi, PTj)$
  Calculate $DistanceNode$

$$DistanceNode = 1 - \frac{w_{op} * SimOperation + w_{pt} * SimPortType}{w_{op} + w_{pt}}$$

end if

Matching basic activites. The cost for editing a basic activity vertex (*receive, invoke, reply*) is calculated by function BasicActivityMatch (see Algorithm 2). This cost expresses the distance between two BPEL basic activities. Each activity has two attributes: the Operation name (Op) and the PortType (PT). The matchmaking gives priority to operation comparison, and if two operations are similar ($SimOperation > 0$), it compares the similarity of the $PortType$ and calculates the distance between activities ($DistanceNode$).

Weights w_{op} and w_{pt} indicate the contribution of Op (similarity of Operations) and PT (similarity of PortTypes) respectively in the total $DistanceNode$ score ($0 \leq w_{op} \leq 1$ and $0 \leq w_{pt} \leq 1$).

Matching connectors. The connectors represent the control flow of process. The cost for editing a connector vertex is calculated by function ConnectorMatch (see Algorithm 3). This cost verifies if two nodes depict the same connectors. Each connector has two attributes: the Connector Type (CT) and the Activity Type (AT) that the connector represents.

Algorithm 3. Function ConnectorMatch

INPUTS: (Nodei,Nodej)
Nodei: Struct (CTi, ATi), Nodej: Struct (CTj, ATj)
OUTPUT: $DistanceNode$

if $CTi \neq CTj$ (different Connector Type) AND $ATi \neq ATj$ (different Activity Type) **then**
  Return $DistanceNode = 1$
else
  $DistanceNode = 0$
end if

Matching wait activities. This function (see Algorithm 4) calculates the cost for editing a vertex which represents a *wait* activity. Each wait vertex has two attributes: a delay for a certain period of time (F) or until a certain deadline is reached (U). The function checks if two $ForExpressions$ or two $UntilExpressions$ are similar, and gives a result for $DistanceNode$ respectively. The time similarity function (TS) calculates the resemblance between the time expressions.

Algorithm 4. Function WaitMatch

INPUTS: (Nodei,Nodej)
Nodei: Struct (Fi, Ui), Nodej: Struct (Fj, Uj)
OUTPUT: $DistanceNode$

if $ForExpression$ there exist **then**
  Calculate ForExpression Similarity $SimFor = TS(Fi, Fj)$
  Calculate $DistanceNode = 1 - SimFor$
else
  Calculate UntilExpression Similarity $SimUntil = TS(Ui, Uj)$
  Calculate $DistanceNode = 1 - SimUntil$
end if

5.4 Linguistic Comparison

The *Linguistic comparison module* calculates the linguistic similarity between two labels based on their names [25]. The labels are often formed by a combination of words and can contain abbreviations. To obtain a linguistic distance between two strings, we use existing algorithms: *NGram*, *Check synonym*, *Check abbreviation*, tokenization, etc. The *NGram* algorithm estimates the similarity according to a number of common *qgrams* between labels names [26]. The *Check synonym* algorithm uses a linguistic dictionary (e.g. Wordnet [27] in our implementation) to find synonyms while *Check abbreviation* uses a custom abbreviation dictionary.

If all algorithms return 1, there is an exact matching. On the other hand, if all the algorithms return 0, it means that there is 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 similarity value will be equal to the *Check Synonym* one. Finally, if the three algorithms values are between 0 and 1, the similarity LS ([25]) 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).

5.5 Decomposing Vertices

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 behavior models to be matched. For instance, for BPEL metamodel, it is possible that in one process a message exchange is modelled as an synchronous interaction, while in the second process is modelled as an asynchronous interaction. Figure 3 shows how a message exchange can be modelled as an asynchronous or synchronous interaction for an operation invoked by the process.

Synchronous interaction	**Asynchronous interaction**
Invoke (request/response)	Invoke (one way) + Receive

Fig. 3. Synchronous vs. asynchronous interactions

Therefore, an invoke operation of type request/response (having m_{in} as input message and m_{out} as output message) can be decomposed in an *invoke operation* (one way, having message m_{in}) and a *receive* operation (having m_{out} as message).

Another example of decomposition operation is related to the granularity of the exchanged message. For example, the first process requires messages *submitOrder* and *sendShippingPrefereces* separately, but the second process needs all of this information included in the *submitOrder* message. In this case, an *invoke* operation (having a message composed of two parts $m1$ and $m2$) will be decomposed in two *invoke* operations (having as messages $m1$ and $m2$, respetively).

These decomposition functions are specific to BPEL 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) specifying how the labels and attributes of the two vertices are obtained from the decomposed vertex.

5.6 Example

Suppose that we would like to find the similarity between two hotel reservation services whose models have been described using BPEL.

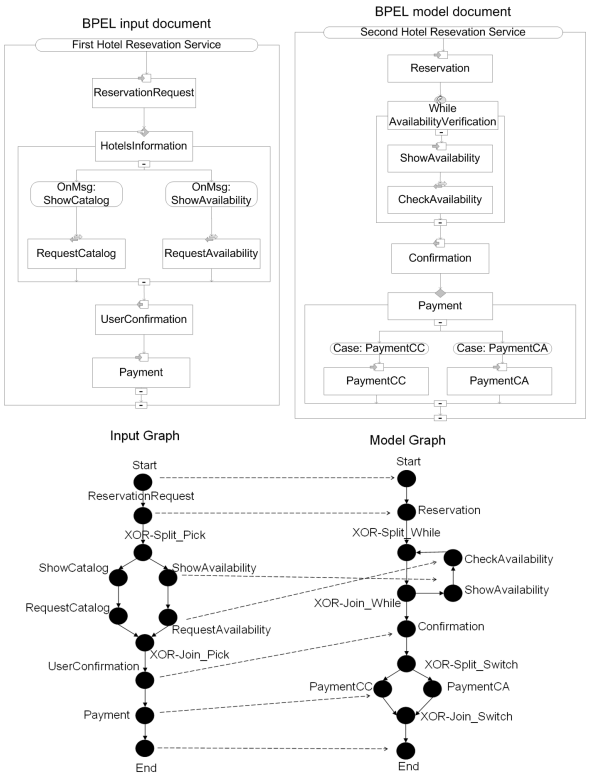

Fig. 4. Example

The first service has the following activities: First, the customer should place his *Reservation Request* (Activity type: Receive). Then the reservation service requires the *Hotels information* (Activity type: Pick), either Catalog (*RequestCatalog*, Activity type: Invoke) or Availability information (*RequestAvailability*, Activity type: Invoke). Next, a confirmation (*UserConfirmation* Type: Reply) is sent to user. Finally, the reservation service finishes the process by receiving the reservation *Payment* (Type: Receive). The second service model has the following activities sequence: first, the customer should place his *Reservation* (Activity type: Receive). Then, the hotel reservation service receives the customer reservation dates (*ShowAvailability* Type: Receive) and verifies the hotels availability (*CheckAvailability* Type: Invoke), until finding availabile rooms. Next, a confirmation (*Confirmation* Type: Reply) is sent to user. Finally, the hotel reservation service requires the customer to pay (*Payment* Type: Switch), either with credit card (are *PaymentCC* Type: Receive) or out of his checking account (*PaymentCA* Type:

Receive). Our system converts each BPEL document into a graph (input graph and model graph, Figure 4). Next, the graphs are compared by the similarity analyzer module. The dotted lines in Figure 4 represent the mappings found by the system between the two graphs using the comparison rules. In conclusion, the edit script will show that the two graphs are similar, but the activities *ShowAvailability*, *CheckAvailability* and *PaymentCC* of the model graph are parts of different structured activities in the input graph. However, the matchmaking algorithm will find similar activities for the right branch of the input graph (*Start*, *ReservationRequest*, *ShowAvailability*, *RequestAvailability*, *UserConfirmation*, *Payment* and *End*).

6 Implementation and Experiments

We implemented the first version of a desktop system having the architecture presented in the previous section. In this section, we present an experimental study of the matchmaking algorithm. The theoretical complexity of the graph matchmaking algorithm [22] 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 the algorithm performs for BPEL process matchmaking. Since most of the existing BPEL process have less than 50 activities, we considered a maximum of 50 activities.

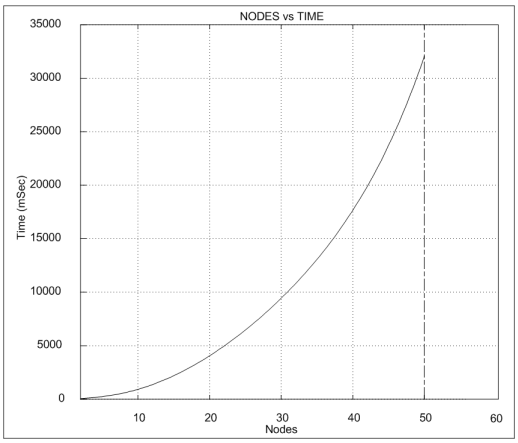

Fig. 5. Matchmaking two BPEL documents

Figure 5 shows the system behavior for two graphs with different structures and different names for activities operations and portTypes. For the comparison of operations and portTypes, the linguistic comparison is used. Despite the exponential theoretical cost, the graph shows that the matchmaking algorithm can be used for BPEL documents having less than 50 activities. The current implementation does not include the two new graph edit operations. We are currently investigating how to efficiently implement them and evaluating the supplementary cost.

7 Conclusion

In this paper we proposed a solution for service retrieval based on behavioral specification. First we motivated the need to retrieve services based on their behavior 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 defined a similarity measure for behavior matchmaking and we showed how to combine fragments in order to satisfy user requirements. We exemplified our approach for behavior matching using the BPEL model. The behavioral matchmaking was implemented as a web service that takes as input the graph representations of two BPEL processes and calculates the degree of similarity between them and outputs also the transformations needed to transform one process into the other.

We are working on generalizing the decomposition and joining graph edit operations to tackle the situation when one node in the first graph corresponds to a subgraph of the second graph. This situation appears when the first service has a single operation (activity) to achieve certain functionality, while in the second service the same behavior is achieved by receiving several messages.

The next step of this work will be to address the problem of comparing a process with a set of processes in a library. Due to the complexity of the matchmaking algorithm, some optimization techniques have to be developped (indexes, clustering to regroup similar services in the library, etc.). We will also experimentally evaluate the performance of the behavior based retrieval method in terms of precision and recall.

Acknowledgements

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

References

1. Foster, I., Voeckler, J., Wilde, M., Zhao, Y.: Chimera: A virtual data system for representing, querying and automating data derivation. In: Proc. of 14th Conf. on Scientific and Statistical Database Management. (2002)
2. Benatallah, B., Casati, F., Toumani, F.: Web services conversation modeling: A cornerstone for e-business automation. IEEE Internet Computing (2004)
3. Benatallah, B., Casati, F., Grigori, D., Motahari Nezhad, H.R., Toumani, F.: Developing adapters for web services integration. In: Proc. of CAISE. (2005)
4. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: Proc. of First International Semantic Web Conference (ISWC). (2002)
5. Bernstein, A., Klein, M.: Towards high-precision service retrieval. In: Proc. of Int. Semantic Web Conference (ISWC). (2002)
6. Benatallah, B., Hacid, M., Rey, C., Toumani, F.: Semantic reasoning for web services discovery. In: Proc. of WWW Workshop on E-Services and the Semantic Web. (2003)

7. Kawamura, T., De Blasio, J., Hasegawa, T., Paolucci, M., Sycara, K.: A preliminary report of a public experiment of a semantic service matchmaker combined with a uddi business registry. In: Proc. of 1st International Conference on Service Oriented Computing (ICSOC). (2003)
8. Cardoso, J., Sheth, A.: Semantic e-workflow composition. Journal of Intelligent Information Systems **21** (2003) 191–225
9. Wu, J., Wu, Z.: Similarity-based web service matching. In: Proc. of IEEE International Conference on Services Computing. (2005)
10. Trastour, D., Bartolini, C., Gonzalez-Castillo, J.: A semantic web approach to service description for matchmaking of services. In: Proc. of Int. Semantic Web Working Symposium (SWWS). (2001)
11. S. Bansal, S., Vidal, J.M.: Matchmaking of web services based on the DAML-S service model. In: Proc. of Int. Joint Conference on Autonomous Agents and Multiagent Systems. (2003) 926–927
12. Zdravkovic, J., P. Johanesson, P.: Cooperation of processes through message level agreement. In: Proc. of Int. Conf. On Advanced Information Systems Engineering (CAISE). (2004)
13. Piccinelli, G., Di Vitantonio, G., Mokrushin, L.: Dynamic service aggregation in electronic marketplaces. Computer Networks **2**(37) (2001)
14. Wombacher, A., Mahleko, B., Fankhauser, P., Neuhold, E.: Matchmaking for business processes based on choreographies. In: Proc. of IEEE International Conference on e-Technology, e-Commerce and e-Service. (2004)
15. Benatallah, B., Casati, F., Toumani, F.: Analysis and management of web services protocols. In: Proc. of ER. (2004)
16. Bordeaux, L., et al.: When are two web services compatible? In: Proc. of TES. (2004)
17. Dong, L., Halevy, A., Madhavan, J., Nemes, E., , Zhang, J.: Similarity search for web services. In: Proc. of VLDB. (2004)
18. Wombacher, A., Mahleko, B., Fankhauser, P.: A grammar-based index for matching business processes. In: Proc. of IEEE International Conference on Web Services. (2005) 21–30
19. Shen, Z., Su, J.: Web services discovery based on behavior signatures. In: Proc. of IEEE International Conference on Services Computing. (2005)
20. Shapiro, L.G., Haralick, R.M.: Structural descriptions and inexact matching. IEEE Trans. Pattern Anal. Mach. Intell. **3** (1981)
21. Bunke, H.: Recent developments in graph matching. In: Proc. of 15th Int. Conf. on Pattern Recognition. (2000) 117 – 124
22. Messmer, B.: Graph Matching Algorithms and Applications. PhD thesis, University of Bern (1995)
23. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services, version 1.1. In: Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation. (2003)
24. Mendling, J., Ziemann, J.: Transformation of bpel processes to epcs. In: Proc. of the 4th GI Workshop on Event-Driven Process Chains (EPK2005). (2005)
25. Patil, A., Oundhakar, S., Sheth, A., Verna, K.: Meteor-s web service annotation framework. In: Proc. of WWW Conference. (2004)
26. Angell, R.C., Freund, G.E., Willett, P.: Automatic spelling correction using a trigram similarity measure. Information Processing and Management **19**(4) (1983) 255–261
27. Miller, G.: Wordnet: A lexical database for english. Communications of the ACM **38**(11) (1995) 39–41