BeMatch: A Platform for Matchmaking Service Behavior Models

Juan Carlos Corrales
PRiSM, Universite de Versailles Saint-Quentin en Yvelines
45 avenue des Etats-Unis
Versailles Cedex, France
jcc@prism.uvsq.fr

Daniela Grigori
PRiSM, Universite de Versailles Saint-Quentin en Yvelines
45 avenue des Etats-Unis
Versailles Cedex, France
grig@prism.uvsq.fr

Mokrane Bouzeghoub
PRiSM, Universite de Versailles Saint-Quentin en Yvelines
45 avenue des Etats-Unis
Versailles Cedex, France
mok@prism.uvsq.fr

Javier Ernesto Burbano
GIT, University of Cauca
Calle 5 No 4-70
Popayan, Colombia
jaburbano@unicauca.edu.co

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 possibly enhanced with some ontological knowledge. Recent works have demonstrated that this approach is not sufficient to discover relevant components. Motivated by these concerns, we have developed BeMatch platform for ranking web services based on behavior matchmaking. We developed matching techniques that operate on behavior models and allow delivery of partial matches and evaluation of semantic distance between these matches and 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. We exemplify our approach for behavioral services matchmaking by describing two demonstration scenarios for matchmaking BPEL and WSCL protocols, respectively. A demo scenario is also described concerning the tool for evaluating the effectiveness of the behavioral matchmaking method.

1. INTRODUCTION

The capability to easily find useful services 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 main problems arising from the current framework of web services is the need to dynamically put in correspondence service requesters with service suppliers, hence allowing the formers to benefit from the more recent offers or updates of the latter.
- Programs and scientific computations (scientific workflows) which are important resources in the context of Grid systems, sometimes even more important than data. The scientists need to retrieve these programs to determine whether it is worth to reutilize them or rewrite them again with respect to desired characteristics.

In all these cases, users are interested in finding suitable components in a library or a collection of programs described by appropriate models. The user formulates a requirement as a process model; his goal is to use this model as a query to retrieve all components whose respective 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 goal, the models that require minimal modifications may be the most suitable ones as component reuse aims generally to reduce development costs. If the retrieved models have to be tailored to the specific needs, the adaptation effort should be minimal.

Recently there has been a proliferation of web service search engines on the Internet (see [1]). These can be classified into two types. The first type accepts as input keywords, which they use to search within WSDL descriptions of services. The second type goes beyond naive keyword matching of WSDL contents by performing a similarity search on WSDL operation names and input/output parameters (see [2, 3]). These approaches can only match simple services, thus do not handle the execution aspects of services.

By describing service capabilities using ontologies, the service discovery can be improved by doing a semantic matching between the required service and the publisehd services. Several web service matchmaking prototypes have been implemented using this approach (see [4, 5, 6]).

The need to take into account the service behavior in the retrieval process was underlined by several authors and some recent proposals exist ([7, 8, 9]). The few approaches that

EDBT'08, March 25–30, 2008, Nantes, France.

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. Moreover, they assume that services have a common semantics on message names.

In the database community, different approaches for automatic matching schemas have proposed. In general, these methods try to capture clues about the semantics of the schemas, and suggest matches based on them (see [10]). These matching techniques can not be directly applied to web service description matching because the granularity of the search is different: services matching can be compared to finding a similar schema, while schema matching looks for similar components in two given schemas that are assumed to be related (see [11]). Moreover, each web service in isolation has much less information than a schema.

Motivated by these concerns, we have developed BeMatch platform for ranking web services based on behavior matchmaking. We developed 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.

The innovative feature of BeMatch, compared with existing approaches for services matchmaking, is that it allows an approximate match. That is, even if companies do not use a common vocabulary for operation names and model differently the sequencing and the granularity of operations of a service, the platform allows to retrieve among a list of services offering the same functionality as the requested one, the service having the most similar behavior model.

2. BEMATCH: DESIGN AND IMPLEMENTATION

Given an extensive repository of published services, the goal of BeMatch platform is to rank services with respect to their suitability in fitting user requirements. We suppose that user expresses his needs as a service behavior model and the platform will help him identifying the services having the most similar behavior model.

The first step for selecting possible candidates is based on non-operational information. For example, a service requester is looking for a partner that runs his business in a certain domain, that is located in a certain area or that has a certain amount of expertise (references) in providing the required functionality. This discovery step is implemented by means of a hierarchically structured catalogue. After the first discovery step, a list of candidate services are found. Then, the query service is compared with each candidate service and the ranked list of candidates is presented to the requestor. In our platform the service ranking is based on service behavioral matching which is reduced to a graph matching problem (see [12, 13]). More precisely, we use the error correcting subgraph isomorphism (*ecsi*) algorithm in order to allow an approximate matching. This method is based on the idea of graph edit operations that are used for calculating a distance measure for graphs.

The services ranking is constructed taking into account different similarity measures. The first measure is based on the graph distance calculated by the *ecsi* algorithm. The second measure takes into account the number of the mapped nodes. If two target graphs have the same subgraph distance to the query graph but are matched to subgraphs with different number of nodes, the one that matches a subgraph with more nodes will be preferred. The third measure is based on the number of sequences of mapped nodes.

The system is presented in Figure 1 and is composed of the following modules:

- Services repository: This repository stores services descriptions and associated metadata. The data in the repository is organized in a hierarchical manner; this allows to query the repository and implement the first step in the discovery process. We filter the services in order to reduce the number of candidates for the second step of the discovery process, the behavioral matchmaking.

- Services to graphs parser: This module transforms a service behavior description (BPEL and WSCL in the current implementation) to a graph.

- Graph matchmaking: This module takes as inputs the query graph and a single target graph (from the candidates list) produced by the parser presented above and computes the semantic distance between them by applying the algorithm for optimal error correcting subgraph isomorphism detection.

- Cost functions: This module groups the cost functions for the graph edit operations that allow to calculate the distance between graphs. The costs assigned to different graph edit operations reflect the relative importance of dissimilarities between different graph attributes. Thus they depend of service behavior metamodel and on the application domain. For example, for WSCL descriptions, the substitution cost of two interactions is calculated based on the similarity of their names and on their messages names.

- Linguistic analyzer: This component calculates the linguistic similarity between two strings using the following algorithms: NGram, Check synonym, Check abbreviation and tokenization.

- Granularity level analyzer: It checks whether decomposition/ composition operations are necessary and add their cost to the graph edit distance. These graph edit operations are necessary when the same functionality is modeled at different granularity levels in the two graphs (for example, using two nodes in a graph and only one node in the other graph).

- Similarity functions: This module defines the similarity functions that allows to construct the service ranking.

- Tool for evaluating the effectiveness of the behavioral matchmaking method. This tool allows to create a service ranking based on manual comparisons between a query service and a list of target services in the repository. The tool permits to compare the ranking defined by the user with the results of our matchmaking tool. The cost functions for graph edit operations use some

user defined parameters (weights reflecting the importance of different service attributes in evaluating the similarity). Given the fact that the parameterization of the cost function is domain dependent and very important for the effectiveness of the matchmaking method, this tool helps the user determine the optimal parameters to apply for a given domain and similarity criteria.

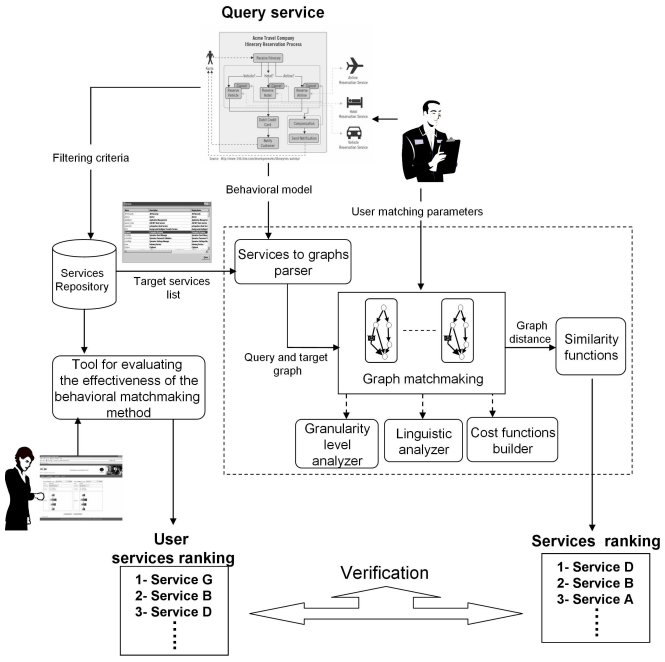

Figure 1: Architecture

The platform was built using the Java programming language (JDK 1.6.0) and the Netbeans 5.0 Integrated Development Environment (IDE). The desktop implementation uses the Xerces Application Program Interface (API) for processing the BPEL and WSCL documents within the Parser Module. The JGraph and JGraphLayout APIs were used for the visual representation of service models. The Linguistic analyzer accesses the Wordnet Dictionary trough the Java Word Net Library (JWNL) API for verifying the semantic relationship between two words.

We use the service repository presented in [14] that supports storing and finding BPEL business processes (and other types of XML documents). In the repository, data is represented as EMF objects (Objects of Eclipse Modeling Framework) that are Java objects. Data can be queried as Java objects using an object-oriented query language, namely the Object Constraint Language (or other query engine based on other query language). This repository together with its querying capabilities (and its possible extensions) allow us to support the first phase of the discovery process, while the main focus of our work is the matchmaking phase based on behavioral models.

The services matchmaker (dotted square in figure 1) is also available as a web service that takes as input two WSCL files and calculates the similarity between them. It returns also the script of edit operations required in order to transform the first conversation protocol to conform with the second one (`http://ariadna.unicauca.edu.co/matching/`).

3. DEMO SCENARIOS

We choose to exemplify our approach for behavioral services matchmaking (dotted square in figure 1) by using the BPEL and WSCL protocols (see [13] for WSCL matchmaking and [12] for BPEL processes matchmaking). BPEL has emerged as a standard for specifying and executing web services-based processes. It supports the modeling 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), without revealing the internal behavior. An *executable process*, in contrast, specifies the actual behavior of a participant. The BPEL demo scenario will compare two *BPEL abstract processes* (chosen from the repository) and will find out automatically the similarity between them. On the other side, 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. The demo scenario for WSCL will show the similarities between two WSCL protocols identifying also the differences in the granularity level descriptions. Finally, a demo scenario is presented concerning the tool for evaluating the effectiveness of the behavioral matchmaking method. This tool is available on the web (`http://ariadna.unicauca.edu.co/pertinence`) and allows to evaluate the usefulness of behavioral based similarity measures. The scenario will show the process followed by user for manually comparing two services.

3.1 BPEL matchmaking scenario

We will compare two BPEL processes for hotel reservation. Suppose that the first service has the following activities: first, the customer should place his hotel selection *Reservation Request* (Activity type: Receive). Then, either ShowCatalog or ShowAvailability message are expected via *Hotels information* (Activity type: Pick). Next, the (*RequestCatalog*, Activity type: Invoke) or ShowAvailability information (*RequestAvailability*, Activity type: Invoke) activities are executed respectively. Afterwards, a confirmation (*UserConfirmation* Type: Reply) with the reserve information is sent to the user. Finally, the hotel reservation service expects the credit card payment *PaymentCC* (Type: Receive). The second service model has the following activities sequence: first, the customer should place his *Reservation* (Activity type: Receive) preferences. Then the hotel reservation service receives the customer reservation dates (*Show Availability* Type: Receive) and verifies the hotels availability (*CheckAvailability* Type: Invoke); if there are no rooms available for the proposed dates, the last two operations are repeated until finding available rooms. Next, a confirmation (*Confirmation* Type: Reply) is sent to the 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 the BPEL documents into graphs (query graph and target graph in Figure 2). Next, the graphs are compared by the graph matchmaking module taking into account the matching parameters defined by user

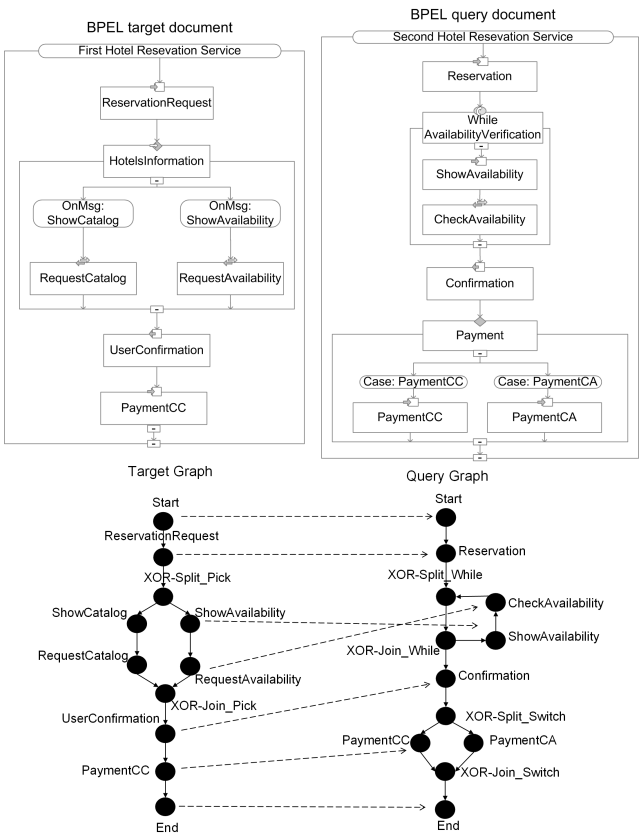

Figure 2: BPEL Example

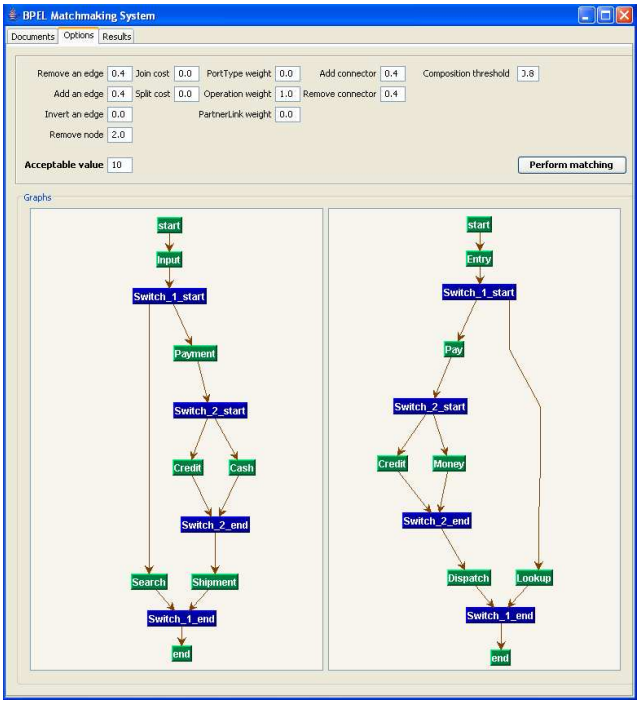

Figure 3: Services matchmaking interface

(see Figure 3). The dotted lines in Figure 2 represent the mappings found by the platform between the two graphs. The edit script will show that the two graphs have some common activities, but the activities *ShowAvailability*, *CheckAvailability* and *PaymentCC* of the query graph belong to different structured activities in the target graph. However, the matchmaking algorithm will find similar activities for the right-hand branch of the target graph (*Start*, *ReservationRequest*, *ShowAvailability*, *RequestAvailability*, *UserConfirmation*, *Payment* and *End*). Finally, the similarity between the query and the target service is calculated taking into account the graph total distance and the user similarity criteria.

3.2 WSCL matchmaking scenario

Suppose that we would like to find the similarity between two purchase services whose conversations have been described using WSCL language.

The first conversation (target service) of the figure 4 expects a conversation to begin with the receipt of a *LoginRQ* message (*Login* interaction). The service sends as a response a *ValidLoginRS* or *InvalidLoginRS* document depending on the type and content of the message received. In case of a valid login, the service will expect a *PurchaseRQ* message (*Purchase* interaction). Depending on the content of the received document, the *Purchase* interaction can reply with *PurchaseAcceptedRS*, *InvalidPaymentRS* or *OutOfStockRS*. If the response is a *PurchaseAcceptedRS*, the service will send the shipping information (*ShippingInformation, type Send*) to the user (*Shipping* interaction), otherwise the conversation will end. Similarly, the second conversation (query service) of the figure 4 expects a *LgnRQ* message and can send as a reply a *ValidLgnRS* or *InvalidLgnRS* document (*Lgn* interaction). In case of a valid login, client can send shipping preferences and the service will return a *ShipmentAcceptedRS* or *OutsideZone* document according to the content of document received (*Shipment* interaction, type SendReceive). In the first case, the conversation continues with a *Buy* interaction, otherwise it ends.

Our system converts each WCSL document into a graph (target graph and query graph, Figure 4). Next, the graphs are decomposed (Decomposed target graph and query graph, Figure 4) using the granularity analyzer. The decomposition step allows to flatten the graphs to the same granularity level, given the fact that an interaction could be modeled in WSCL as a SendReceive interaction or as a Send interaction followed by a Receive interaction.

Then, the graphs are compared by the graph matchmaking module. Finally, the cost of granularity differences is added to the total graph distance. In conclusion, the edit script will show that the two graphs are similar, but have the following structural differences : for the mapping (*Shipment,Shipping*) (see figure 4), the *Shipment* is a Send non atomic interaction (was obtained by decomposing a SR interaction) and the *Shipping* is a send atomic interaction. Hence the system will add a granularity cost to the total distance between the two graphs. There is not a corresponding node for the Shipment (R) into the target graph. On the other side, the interactions for purchasing and for shipping are executed in different order in the two models, therefore the system will add to the total distance the costs of necessary

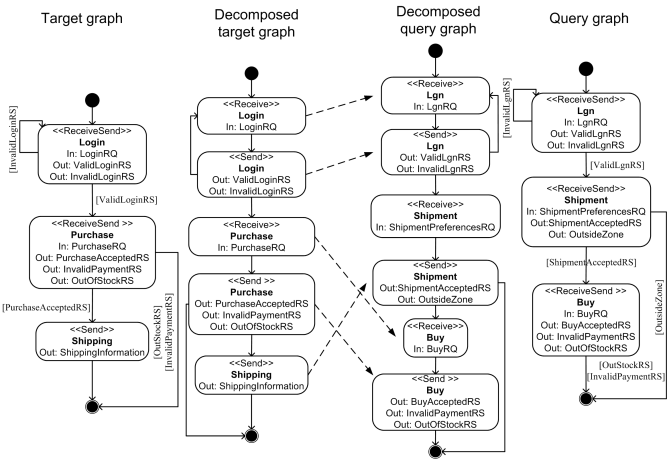

Figure 4: WSCL Example

edit operations for reordering them.

3.3 Scenario for manual evaluation of services similarity

After logging in the evaluation tool, the user selects the services to compare in the interface presented in figure 5. In this demo scenario the user selects the services of figure 4. Then he assigns a similarity score (between 1 and 5) for the two services corresponding to the comparison criteria that he finds relevant.

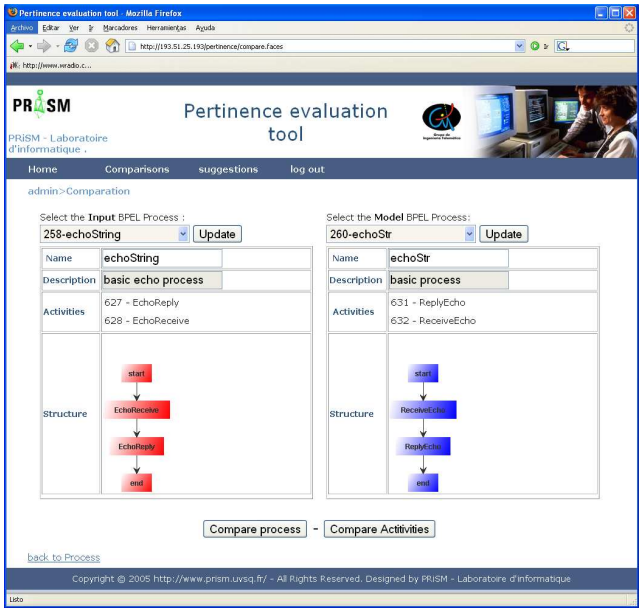

Figure 5: A screenshot of the evaluation tool

The proposed similarity criteria are: service name, service description, activities set and service structure. For the analyzed example the user selects the service description as the most important criteria given that the two services describe hotel reservation processes. On the other side, suppose that user selects as criteria the activities set. In this case the tool allows to analyze each service branch and to compare their interactions. Thus the user can identify a similarity between: *ReservationRequest* interaction on the target service and *ReservationRQ* interaction on the query service; then between *RequestCatalog* and *Catalog* interaction, and finally between *CheckAvailability* and *Availability* interaction. Finally, the tool creates a ranking for each query service analyzed according with the comparison criteria. This list is a *Top n* (1<n<10) ranking, where the first service is the most similar one.

4. ACKNOWLEDGEMENTS

Juan Carlos Corrales is supported by the Program Alban, the European Union Program of High Level Scholarships for Latin America, scholarship No. (E04D042012CO). (`http://www.programalban.org`).

5. REFERENCES

[1] M. Saboua and J. Panb. Towards semantically enhanced web service repositories. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2007.

[2] E. Stroulia and Y. Wang. Structural and semantic matching for assessing web-service similarity. *Int. J. Cooperative Inf. Syst.*, 2005.

[3] N. Kokash, W. van den Heuvel, and V. D'Andrea. Leveraging web services discovery with customizable hybrid matching. In *Proc. of ICSOC*, 2006.

[4] L. C. Chiat, L. Huang, and J. Xie. Matchmaking for semantic web services. In *Proc of SCC*, 2004.

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

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

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

[8] B. Mahleko and A. Wombacher. Indexing business processes based on annotated finite state automata. In *Proc. of The ICWS*, 2006.

[9] Li Kuang, Ying Li, Shuiguang Deng, Jian Wu, Wei Shi, and Zhaohui Wu. Expressing service and query behavior using pi-calculus for matchmaking. In *Proc of WI*, 2006.

[10] Hong-Hai Do and E. Rahm. Coma - a system for flexible combination of schema matching approaches. In *Proc of VLDB*, 2002.

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

[12] J.C. Corrales, D. Grigori, and M. Bouzeghoub. Bpel processes matchmaking for service discovery. In *Proc. of CoopIS*, 2006.

[13] D. Grigori, J.C. Corrales, and M. Bouzeghoub. Behavioral matchmaking for service retrieval. In *Proc. of ICWS*, 2006.

[14] J. Vanhatalo, J. Koehler, and F. Leymann. Repository for business processes and arbitrary associated metadata. In *Proc of BPM*, 2006.