Structure Matching for Enhancing UDDI Queries Results

Giancarlo Tretola
University of Sannio
Department of Engineering
Benevento,82100 Italy
tretola@unisannio.it

Eugenio Zimeo
University of Sannio
Research Centre on Software Technology
Benevento,82100 Italy
zimeo@unisannio.it

Abstract

To enhance UDDI query capability, typically based on taxonomic classification, semantic matching is assuming a key role. Even if we recognize the great importance of semantics in the discovery process, structure-based matching can be very useful in many situations where semantic annotations are not provided at publish time or ontologies are not yet well defined. Moreover, structure matching has a potential application in dynamic binding and invocation to perform correct parameter passing based on syntactic elements obtained from the mapping returned by the structure matching algorithm. The paper discusses the problem of similarity structure matching and proposes and compares different implementations of the algorithm introduced by Wang-Stroulia with the aim of obtaining better performance. We integrated the algorithm in a matchmaking framework based on multiple cascade filters that are able to combine several matchmaking techniques in order to improve precision and recall in a flexible and effective way.

1. Introduction

Web Services based distributed applications in B2B environments are now dealing with an important step in the improvement of the paradigm: discovery and binding of published services. The main problem in this domain is related to the growing diffusion of Web Services deployment that makes harder to find the most apt service for the specific needs of the business process to implement. Web Service Description Language (WSDL) is oriented to interface description and is not aimed to support functional matching and selection when binding is performed. However it is possible to consider the use of WSDL to define the mapping between syntactic elements in service invocation.

In the Service Oriented Architecture (SOA) a registry has the role to collect information about published services, making them accessible to the client that is looking for the more appropriate service to invoke. The published services are catalogued with taxonomic criteria (NAICS or UNSPSC) and the Universal Description, Discovery and Integration (UDDI) provides an API to interact with the registry. The API offers functionality for issuing queries, aimed to discover the services, based on taxonomic categorization. This discovery approach is clearly not sufficient, because it does not distinguish the different services in the same category to enable selecting the best fitting one.

To overcome the limitations of UDDI registry, many researchers are proposing several techniques, either based on structural or semantic information [14][15]. Even if semantic allows for more expressive service descriptions and more precise selection of desired services, it still requires a lot of burden in the publishing phase and a lot of computational effort at discovery time. An alternative, useful approach that improves keyword-based search and syntactic matching is called "structure matching". With this approach, a basic semantic of services is inferred from their structure and in particular from operations and parameters types. The approach could be used to provide an indication of the degree of similarity between a requested service and the set of services returned by the taxonomic based queries.

One interesting proposal is described in [1] through the definition of an algorithm aimed to compute the structural similarity degree between two service interface descriptions. In this paper, we present an implementation of that approach proposed with the aim of improving performance. Our implementation of the algorithm represents a part of a matchmaking framework, which is able to use one or more matching techniques with a cascading pipe and filter architecture to improve matching results. Therefore, with this work, we extend and improve UDDI performances enabling more accurate queries, even if semantic information is not available. Our opinion is, in fact, that structure matching could be used by the side of semantic matching in a complex matchmaking system for two main reasons. The principal reason is the possibility to perform a matching procedure in fields where there is not any ontology available for semantic description or if only partial descriptions are possible. As example, let's consider services described without an available ontology for data semantics. Services, then, are described only with functional semantics. Structure matching could be used, after the functional semantic matching, to perform the matching of input and output data structures that should be not executed using semantic matching. That operation could be useful if there is the necessity to make automatic selection and a subsequent invocation: structure matching could be used to make a matching of data structures involved in the operation, performing a mapping between data types useful for finding the correct order of parameters passing and data conversion.

Moreover, using structure matching could be possible to improve overall performance, filtering the services before executing semantic matching, for example, in order to consider only the services with a required degree of similarity.

The paper is organized as follow. In Section 2, we briefly describe the operative environment: the Service Oriented Architecture, the discovery phase, and the matchmaking strategies. In Section 3, we analyze more in detail the structure matching and related work in schema and structure matching fields. In Section 4, we present the Wang-Stroulia algorithm. Section 5 describes our design and implementation activity. Section 6 shows the experimental results, while Section 7 concludes the paper highlighting the potential exploitation of the implemented version of the algorithm.

2. SOA and matching operation

Service Oriented Architecture is a paradigm applied to develop distributed application in B2B environment, using functionality supplied by external enterprises, exposed as Services [4]. We can say, briefly, that SOA is an architecture aimed to model systems and components designed for interoperability, collaboration and reuse. An important feature of SOA is the possibility to look for and retrieve the needed services also at run-time through a process known as *dynamic binding* [3]. In more details, SOA defines three main components and roles that perform the fundamental interactions: Provider, Registry and Requestor.

In a typical scenario, a provider hosts the implementation of a service and publishes on the registry the interface that describes the service. A requestor may query the registry to discover a service useful for enacting an operation in its process. It obtains the information needed to bind to the provider and invokes the functionality.

The Web Service implementation of SOA is based on three main technologies: WSDL, SOAP, and UDDI. WSDL is an XML format for describing services as a set of endpoints operating on messages, described abstractly, and then connected to concrete endpoints [5].

The information contained in a WSDL description could be divided in two logical sections: the interface, which defines the service operation and messages involved in invocation, and the implementation description that identifies a concrete instance of the interface and a protocol binding to it.

In a WSDL interface, information is hierarchically organized. At lowest level there are data types, still defined hierarchically in XML. Then there are input/output messages, which use data types for the definition of the parameters list. At top layer there are the operations that refer to messages.

Simple Object Access Protocol (SOAP) is the interaction protocol, based on XML, used to perform the request/response messages exchange between requestor and provider [6]. The Universal Description, Discovery & Integration (UDDI) [7][8] is an initiative of the OASIS [9] to define a registry able to perform the operation of collecting published information, allows for discovery with query and eases the integration of the services in a business process. The classification and the inquiry of the registry are based on taxonomy classification of the published services.

The discovery process is the most critical step in the lifecycle of Web Service based application: once identified the service requirements, there is the need to select the one that fits well with them [10]. The overall operation to perform is the matching, that is the comparison of a desired service description, called *template*, with a list of several candidate service interface descriptions, called *targets*.

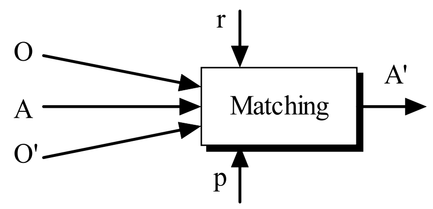

Figure 1. The matching process

The matching operation may be defined as a function that receives two descriptions as input and returns as result a mapping between the two descriptions, called mapping elements, composed of five elements: (id, e, e', n, R). **id** is an univocal identification number, **e** and **e'** are the entities of the first and of the second descriptions, **n** is the match result that expresses numerically the matching between **e** and **e'**, and **R** is a relation that states the similarity degree.

The matching process, depicted in Figure 1, is aimed to produce an alignment of mapping elements. **A** is the input alignment, to be completed by the matching process with the match result, i.e. mapping and matching score. **O** and **O'** are the descriptions to match, representing, for example, the couple of schema or ontology to be matched. **A'** is the resulting alignment. The **p** parameter is used as boundary threshold for **n** or **R**. **r** models a resource used for the matching, for example a data dictionary.

The matching process can be classified in syntactic and semantic. The syntactic matching is aimed to determine the linguistic correspondence of the elements in the two matched descriptions, **R** is a syntactic relationship computed between entities and is measured with **n,** which is a numerical quantification of the syntactic relationship [11]. The semantic matching expresses the conceptual similarity between the elements in the two descriptions by means of a semantic relationship **R** and typically lacks a numerical score [13]. The semantic matching, in fact, returns as a result a conceptual relationship between the matched descriptions based on the ontology used to annotate the services. Then, this relationship is qualitative and is not expressed in numerical form. In [13], for example, the matching of two descriptions may have four possible results: exact, plug-in, subsume and disjoint. However, a ranking process can be built by considering the semantic distance of target descriptions from the template.

3. Related works

The discovery process that we are going to examine in this work is aimed to identify a service that could satisfy the functional requirements specified by a requestor using WSDL interface descriptions. The matching procedure, based on the structure of the descriptions, is an important operation in other traditional IT systems, such as data warehousing, information integration, etc. [11]. There are, for example, procedure in the e-commerce domain aimed to XML Schema Matching, as preliminary operation to the definition of automatic expressions that translate data instance from a schema to another one [12]. Summarizing, in Web Services domain, during the discovery and integration phases there is the need for an operation useful to assign the correspondence among the various description terms of the services. To perform this operation, in situation where it is not available an ontology that supports semantic descriptions and considered that WSDL is an extension of XML, it is possible to use a particular type of schema matching: *structure matching*.

The schema matching allows for checking syntactic correspondence between elements of two descriptions or XML schema. In Figure 2, a high-level taxonomy classification of the schema matching [15] is shown. The individual matching computes the score basing on a single criterion, which could be further specified: comparison of data instance versus comparison of data schema, or single element matching versus structure of the entity, etc. The "combining matcher" obtains the result using the same strategy with several criteria (hybrid matching) or combining results from different algorithms (composite matcher). A further classification, based on the granularity and kind of inputs, introduces important evaluation criteria: the distinction between approximate or exact techniques; the differentiation from syntactic, extern and semantic; classification in terminological, structural and semantic [12].

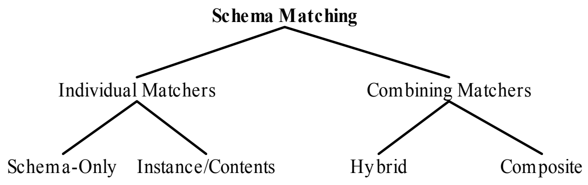

Figure 2. Schema matching taxonomy

We consider, now, the works performed in the field of Structure Matching and the existing prototypes, which are all classifiable as Hybrid Matchers.

In [21], a structure-matching algorithm based on the concept of similarity propagation is proposed. The strategy works on schemas that are converted in labelled graphs. The algorithm performs the element level matching between the labels. The central idea is that two elements are similar when the near elements are similar, so the similarity is propagated in the graphs. The propagation in the graph is performed as flooding does for IP packets in broadcast communication. This strategy has been implemented and tested in Rondo [22]. Another hybrid schema matching prototype is Cupid [1]. In the first phase of the strategy, a linguistic match is performed, using three elements level matching based on names, data types and domain. In the next phase, the schemas are transformed in a tree and there is a structure matching. The matching score is the weighted mean of the two matching, linguistic and structural.

We focalize our interest on a structural matching algorithm to reach the proposed objective: adding a better distinction between services collected through UDDI queries.

4. Wang-Stroulia algorithm

The paper focuses on different implementations of the structure-matching algorithm proposed by E. Stroulia and Y. Wang [1]. The matching criterion used is based on the hypothesis that if two services are conceptually similar, they are in the same taxonomical category, and then they are also structurally similar. The desired service description (template) is matched with the target description to evaluate the similarity level. Since the WSDL service description is based on XML syntax, their elements are compared in a hierarchical way. The conceptual similarity hypothesis was used to develop a heuristic, to assign a degree of structural similarity to WSDL structures.

The matching algorithm is a domain specific implementation of the tree-edit distance algorithm [2], which calculates the similarity of two tree structures as the minimum number of node modifications required to match them.

The matching of two Web Services is based on the match of their composing operations. All possible pairing of each template-operation with each target-operation is matched and a score is assigned. The maximum score of all possible pairings determines the mapping between operations.

The matching of an operation is based on the comparison of the messages. Each operation has input and output messages, and the matching between a couple of operations is performed assigning a score to the pair of input messages and to the pair of output messages. The scoring for messages pairing is assigned performing the matching of parameters data types.

One data type is matched with another comparing the elements of each type. The elements from each type are stored in two lists: the template list and the target list. To consider all pairings, an integer matrix is built. Each matrix element (i,j) contains the score assigned to the similarity between template element **i** with target element **j**. The score is assigned using a heuristic criterion and a MAXSCORE (equals to 10) value is used as defined by Wang and Stroulia. If two elements are primitive and could be converted one into the other without information loss, they are compatible and their score is MAXSCORE. If the transformation could cause information loss they are semi-compatible and their score is MAXSCORE/2. If they are not compatible the score is 0. If at least one of the elements is a complex structure then a new list of elements is created and a recursive invocation to the data types matching function is invoked, to obtain the matching score. A data types matching could receive a bonus, equal to MAXSCORE, if the compared structures have the same grouping style. After the matrix is filled, the final step of the matching is performed using the similarity score to define the mapping between elements of the two data types. All the possible pairings are formed and the result is computed summing all the scores of each pairing. The set of pairing with the highest result is selected as the mapping between the two data types defining the resulting score of the matching.

Considering the complexity of the algorithm it is clear that the most onerous operation is the research of the best mapping for all couple of elements in the similarity matrices. The solution proposed in [1] uses the total enumeration of all the possible associations that give the correct solution but with an enormous computational cost. In this paper we analyze others methods to implement alternative solutions for the mapping problem. One possibility is the use of sub-optimal methodology, as the Greedy algorithm, that provides sub-optimal solutions but requires less computational cost. Another possibility is the use of algorithms applicable only in particular situations, as the Kuhn-Munkres algorithm, which gives the correct solution but works only with square matrixes.

5. Design and implementation

Our implementation of structural matching algorithm [1] was designed to work with a brokering system that represents an extension of a matchmaking framework that we defined and we are implementing using Java technologies. The framework sets the architecture and the control logic for the matchmaking of Web Services, which is composed of the following typical operations: description, discovery, matching and selection of services. To this end, the framework works using a registry, where the publishing information about the services are stored to form a single search space.

To obtain a working system through the specialization of the framework, three features are necessary to define: service description languages, service registry and matching strategies. The first feature defines as the services are described using the data and metadata. The second determines the specific registry used to publish the services and to store the data and metadata. The last feature defines the strategies and the algorithms to be used in services comparison and selection.

Service registry is accessed through a Repository Adapter that uses registry API for interaction. Publishers could interact with Matchmaker to retrieve information about services annotation. Requestors interact with the framework using a matchmaking API. The matchmaker conceptual architecture is depicted in Figure 3. The matchmaker core contains the frozen spot of the framework, defining control logic of operations and using the Factory pattern to allocate the concrete implementation of the framework hot spots related to description and matchmaking, that are chosen by defining the three features of specialization. The Matching Manager contains the hot spot used during specialization to plug in the custom code apt to manage the description and the strategies chosen for the specialization. The Repository Adapter performs the adaptation to the selected Repository for Service description.

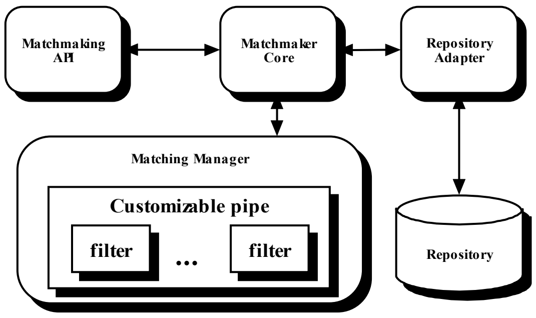

Figure 3. Conceptual matchmaking framework

Regarding the operation of the framework the matching process starts with a request formulated by a service requestor using the Matchmaking API. The Matchmaker Core receives the target description of the desired services and accesses to the repository. The first step is the reduction of the services search space using the functionality offered by the repository to submit a query. This operation returns the list of candidate services. The Matching Manager, performs a fine-grained analysis of the service subspace using a set of matching filters organized in a pipelined structure. A matching filter is a component of the framework that executes a specific matching strategy.

The matching between services description is performed using a specific Matching Function. It is possible to customize the pipeline by implementing different filters, characterized by distinct strategies, choosing which filter to use and defining their sequence. The filters are able to reduce the list of candidate services, step after step, returning a mapping between the target and the templates. We developed a testing specialization of the conceptual framework by defining the three mentioned features: service description, registry and matching strategies. The descriptions used to annotate the services are: taxonomy (NAICS or UNSPSC), tModel, WSDL syntactic description. The registry used is the UDDI registry and specifically we used UDDI4J [16] and JUDDI [17]. The matching strategies used are the functional and data semantics based on the algorithm defined by Wang and Stroulia. However, for other purpose we have also implemented and integrated the semantic matching as proposed by Paolucci [13].

To integrate the algorithm in a filter and this in the matchmaking framework an architecture hot spot has to be extended. The architecture uses the strategy and the template method design patterns. The class diagram that describes the framework adaptation is shown in Figure 4. The framework control logic uses an interface that defines the Matchable type, that has the method *setMatchingFunction* to choose the matching function to use among the possible choices. That interface defines a frozen spot of the framework that allows for selection of the matching strategies.

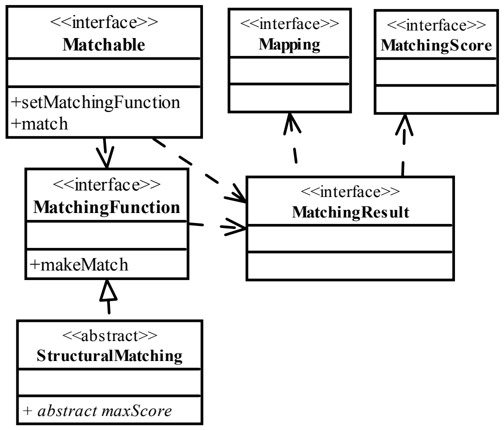

Figure 4. Matchmaking framework adaptation

The matching process is performed with the invocation, on behalf of framework control logic, of the method *match*. The hot spot to be implemented is the interface MatchingFunction that defines a type with a single operation, *makeMatch*, which performs the matching of a template with a target description, according to the Strategy design pattern.

The Strategy implementation is driven via the abstract class StructuralMatching, which defines the algorithm for the *makeMatch* that is a template method for the structural matching algorithm, according to the Template Method design pattern. To this intent, the abstract class defines also an abstract hook method *maxScore*, which is used to define the maximum similarity score for the elements. The matching method returns a matching result that is composed of the matching score assigned to the couple of descriptions examined and a mapping among the elements in the descriptions, defined as we have described in the second section related to the matching problem.

The use of a hook method allows for different implementations of the mapping problem. We have developed three different solutions for this hook method: *total enumeration* of all the possible association, a *greedy algorithm* and a solution based on the *Hungarian algorithm*, a technique for solving resource allocation problems.

6. Performance evaluation

To evaluate the different implementations of the Wang-Stroulia algorithm, we conducted three tests. First, we have tested the correctness of the implementation, performing a matching with services with different levels of similarity. Second, we conducted a quantitative test about the performance of different implementations of the algorithm, which differ in the functionality that computes the maximum score of similarity. The last test was about the quality of the matching.

In the first test, we used the same Web Service as described in [1] (to obtain comparable results) which contains only one operation: int GetData(POType p). It is a complex data type and is sketched in Figure 5. We conducted three tests, performing the match with other two simple services containing only one operation and in the third case with an identical service. The data types used in the first two examples, Type1 and Type2, are also shown in Figure 5, the third data type, Type3, is identical to the template type POType.

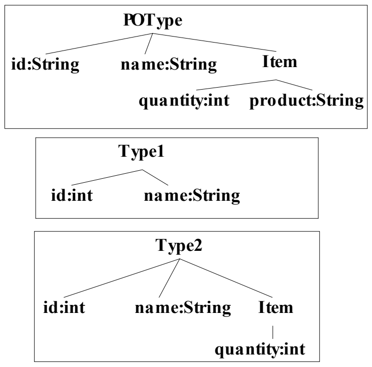

Figure 5. Data types used in correctness test

The maximum score that could be realized in this test is 70, that is the matching score obtained from the matching of the template description with itself. The following table shows the mapping between POType and Type 2 in the second test, giving a maximum score of 40. Symbol ?→ indicates recursive invocation of the matching function, to compare complex elements. The resulting mapping is denoted by the numbers in bold style.

Table 1. Second test data types matching

		Type2		
		id:int	item:Item	code:String
POType	id:String	5	?→10	**10**
	item:Item	?→10	**?→10+10**	?→5
	name:String	**10**	?→5	10

The test results are presented in Table 2. We can observe that the algorithm implementation is able to give different results for the description that presents major structural difference (the first one); the result is less different for a description that is more similar (the second one). In third case, the matching of the most similar service returns a value equals to the maximum score. The quantitative testing was conducted to evaluate the different implementations for solving the maximum score problem. We used the three different algorithms with different computational complexities, shown in Table 3.

The first algorithm is applicable in every situation and gives always the optimum solution.

Table 2. Correctness testing

Test	Score	(Score/Max Score)*100
1st Test	34	49%
2nd Test	60	86%
3rd Test	**70**	**100%**

The second method is always applicable but give a sub-optimal solution. The third method is applicable only to match description of the same dimension, i.e. it works only on square matrices, but gives the optimal solution.

Table 3. Computational complexity

Algorithm	Complexity
Total Enumeration (optimal)	$O((\max\{m,n\})!)$
Greedy (sub optimal)	$O((\max\{m,n\})^2)$
Kuhn-Munkres (Hungarian method – optimal with constraints)	$O(n^3)$

To produce a performance comparison, we conducted the matching between two descriptions with the same number of elements, so it is possible to use also the third algorithm. The performance was measured with a growing dimension of the description, i.e. the number of composing elements, starting with 6 and scaling up until 10. All the operations were conducted on a: PC – Pentium IV 2.4 GHz with 512 MB of RAM. The values measured were the total execution times of the matching function. The results obtained are shown in Table 4, where time is measured in milliseconds.

To conduct a deep analysis of performance, we have plotted the execution times in Figure 6 with the dimension of descriptions on the horizontal axis and execution time on the vertical axis, by using a logarithmic scale.

Table 4. Performance test results

Matrix dimension	Total Enumeration	Greedy	Kuhn-Munkres
6	26 ms	0,073 ms	2,5 ms
7	107 ms	0,08 ms	2,8 ms
8	944 ms	0,1 ms	3,01 ms
9	11751 ms	0,151 ms	3,31 ms
10	148362 ms	0,198 ms	3,52 ms

The figure shows that the total enumeration solution is computational heavy even with the smallest value, and performs the matching with time that increases exponentially. The greedy algorithm returns a sub-optimal solution, which could be a non-optimal mapping, but the execution times are really very good.

The Hungarian method returns an optimal solution and presents the best compromise between correctness and execution time. The major problem is its applicability to only the special cases where the template description and the target description are composed by the same number of elements.

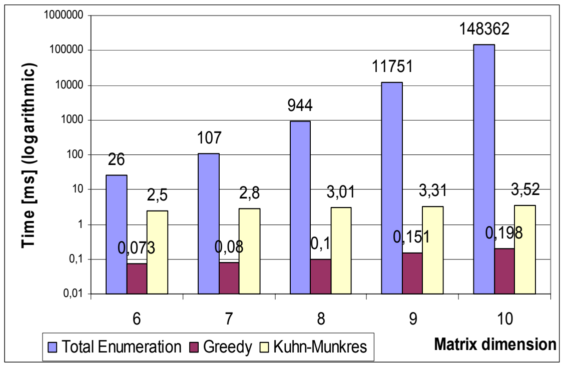

Figure 6. Performance results

The last test is about the qualitative feature of the algorithm. To this end, we uses precision and recall metrics, both well known in the information retrieval research [19]. Precision measures the fraction of really similar descriptions returned on the total of the returned descriptions. It represents the ability of the strategy to present correct results. Recall measures the fraction of really similar descriptions on the total of all the similar descriptions in the service space. It represents the ability of the strategy to individuate as many as correct results. Both values range from 0 to 1, with 1 as the best result and 0 the worst.

To carry out a qualitative test, it is necessary to manually perform a pre-match, that is defining in advance which is the service space, which results are correct and which are not correct. To perform this pre-match we have used the website XMethods [20], that presents publicly available Web Services to the users, classified in categories. We selected 20 services in five different categories, used also for the evaluation in [1] in order to perform a similar testing.

DNA info searcher	5 services
FAX	4 services
Email Address Verifier	5 services
Stock Quote Finder	3 services
Weather Info Finder	3 services

The services in each category were considered to be similar and compatible among them. The testing was conducted using each service as template and comparing it with any other service. This means that for the first category, we used the five services as template and all 20 services as target list. In this test, we used the algorithm implementation that employs the greedy techniques to resolve the mapping sub-problem. The results obtained are shown in Table 5.

Table 5. Qualitative test results

Category	p[%]	r[%]
DNA info searcher	64,0	80,0
Fax	32,5	43,8
Email Address Verifier	38,6	28,0
Stock Quote Finder	8,3	33,3
Wheater Info Finder	26,4	44,4
Average	**34,0**	**45,9**

The overall average precision obtained is about 34,0% and the overall average recall is about 45.9%, which show that the algorithm could make errors when there are services that, although different, could have a quite similar syntactic structure. The results obtained are compatible to those presented in [1]. Wang and Stroulia have proposed the use of supporting techniques to improve precision and recall. Also in our opinion, algorithm should not be used alone but supported by other techniques or in a system that uses also semantic strategies to improve precision and recall, while structure matching contribute to global performance improvements.

7. Conclusions

In this paper, we presented the implementation of a structure-matching algorithm that aims at improving UDDI registry querying. The algorithm was realized as part of a larger project under development: a framework for Matchmaking of Web Services. The algorithm was realized using a Template Method pattern for having different resolving strategies.

The performance evaluation showed that the algorithm is useful to discover services similarity but when used alone it does not give satisfactory results, as shown by the values of precision and recall indexes. The precision and recall could be improved when the algorithm is used with other discovery and matching strategies. To this end, we have some ideas for possible further improvements.

We planned to use the algorithm together with a semantic matching strategy, and we are conducting these tests in our framework. In one scenario, the structure matching is used to complement semantic matching, when semantic information is not sufficient. In this scenario, the structure-matching algorithm is not used completely, but only its first step, the data similarity, could be used to improve the performance of semantic matching. The matching could be adopted to compare the types of the parameters when data semantic information is not available.

In another case, the first step of the matching process could be the sorting, in descending similarity order, of the services, performing a filtering operation on the services space and so reducing the quantity of semantic matching to perform.

An additional enhancement could be achieved by increasing the modularity of the structure matching. In the third test, in fact, we noted that the algorithm performs quite well in matching some types of services in certain categories, for example DNA search, and gives worst result in others, for example Stock Quote. Then it could be possible to use different heuristics in the assignment of similarity score for taking into account this problem. The heuristic could be more specialized and fit well in a single taxonomic category. Having a heuristic for each category could be a noteworthy improvement in qualitative performance.

Finally, another additional improvement could be a heuristic that adapts itself at runtime. It could be adjusted considering runtime performances, that is, collecting results from matching and obtaining feedbacks from users to adjust heuristic scoring. In such a way the structure matchmaking could be adaptively fitted to the client behaviour and category of services used.

Acknowledgements

The work described in this paper is framed within the LOCOSP project funded by Italian Ministry of University and Scientific Research, ex MIUR.

We also thank Ausilio Di Prizito for his support during the implementation phase of the algorithm.

8. References

[1] E. Stroulia, Y. Wang, “Flexible Interface Matching for Web-Service Discovery”, 4th IEEE International Conference on Web Information Systems Engineering (WISE 03), 10-12 December 2003, Rome Italy.

[2] M. Garofalakis, and A. Kumar. "Correlating XML Data Streams Using Tree-Edit Distance Embeddings". In Proceedings of ACM PODS’2003.

[3] S. Matthew, J. McGovern, M. Stevens, S. Tyagi, “Java Web Services Architecture”, Morgan Kauffman Publishers, 2003.

[4] World Wide Web Consortium, W3C, http://www.w3c.org.

[5] WSDL, Web Services Description Language, http://www.w3c.org/TR/wsdl.

[6] SOAP, Simple Object Access Protocol 1.1, 2002, http://www.w3.org/TR/SOAP.

[7] UDDI, Universal Description, Discovery, and Integration, http://www.uddi.org/.

[8] UDDI technical paper, http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf.

[9] OASIS, http://www.oasis-open.org/.

[10] J. Cardoso, A. Sheth, “Semantic e-Workflow Composition”, Journal of Intelligent Information Systems (JIIS), 2003.

[11] P. Shvaiko, “Iterative Schema-based Semantic Matching”, Technical Report DIT-04-020, University of Trento, 2004.

[12] J. Euzenat, P. Shvaiko, “A Survey of Schema-based Matching Approaches”, Journal of Data Semantics (JoDS), IV, LNCS 3730, 2005.

[13] T. Kawamura, T. Payne, M. Paolucci, K. Sycara, “Semantic matching of web services capabilities”, in Proc. of the 1st International Semantic Web Conference (ISWC), 2002.

[14] T. Kawamura, T. Payne, M. Paolucci, K. Sycara, “Importing the Semantic Web in UDDI”, in Proceedings Web Services, E-Business and Semantic Web Workshop, CAiSE 2002. Toronto, Canada, 2002.

[15] P. A. Bernstein, E. Rahm, “A Survey of Approaches to Automatic Schema Matching”, VLDB Journal, 2001.

[16] UDDI4J, http://www.uddi4j.org.

[17] WSDL4J, http://sourceforge.net/projects/wsdl4j.

[18] WSIF, http://ws.apache.org/wsif/.

[19] I. Cho, L. Krause, J. McGregor, “A protocol-based approach to specifying interoperability between objects”, In Proceedings of TOOLS’26 Santa Barbara, California, 1998.

[20] XMethods, http://www.xmethods.com/.

[21] H. Garcia-Molina, S. Melnik, E. Rahm, ”Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching”, ICDE, 2002.

[22] P. A. Bernstein, S. Melnik, E. Rahm, “Rondo: A Programming Platform for Generic Model Management”, SIGMOD Conference, 2003.

[23] P. A. Bernstein, J. Madhavan , E. Rahm, ”Generic Schema Matching with Cupid”, VLDB, 2001.