Hybrid Adaptive Web Service Selection with SAWSDL-MX and WSDL-Analyzer

Matthias Klusch, Patrick Kapahnke, and Ingo Zinnikus

German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, Saarbrücken, Germany
klusch|patrick.kapahnke|ingo.zinnikus@dfki.de

Abstract. In this paper, we present an adaptive, hybrid semantic matchmaker for SAWSDL services, called SAWSDL-MX2. It determines three kinds of semantic service similarity with a given service request, that are logic-based, text-based and structural similarity. In particular, the degree of structural service similarity is computed by the WSDL-Analyzer tool [12] by means of XMLS tree edit distance measurement, string-based and lexical comparison of the respective XML-based WSDL services. SAWSDL-MX2 then learns the optimal aggregation of these different matching degrees over a subset of a test collection SAWSDL-TC1 based on a binary support vector machine-based classifier. Finally, we compare the retrieval performance of SAWSDL-MX2 with a non-adaptive matchmaker variant SAWSDL-MX1 [1] and the straight forward combination of its logic-based only variant SAWSDL-M0 with WSDL-Analyzer.

1 Introduction

As a W3C recommendation dated August 28, 2007, the SAWSDL[1] specification proposes mechanisms to enrich Web services described in WSDL[2] (Web Service Description Language) with semantic annotations. Among others, one goal of these additional descriptions is to support intelligent agents in automated service selection, a task which is hard to accomplish using pure syntactic information of service profiles based mainly on XML-Schema definitions. Typical application scenarios that require or benefit from a service matchmaking component include for example negotiation and coalition forming among agents and automated or assisted service composition. The first hybrid semantic service matchmaker SAWSDL-MX1 for SAWSDL we proposed in [1] adopted the ideas of our hybrid matchmakers OWLS-MX and WSMO-MX (see [5, 3]) for OWL-S, respectively, WSML.
However, SAWSDL-MX1 focused on semantic annotations of the signature but not on the XML structure of the Web service as a whole. This is taken into account by the WSDL-Analyzer tool presented in [12] by means of measuring

[1] http://www.w3.org/TR/sawsdl/
[2] http://www.w3.org/TR/wsdl/,
http://www.w3.org/TR/wsdl20/

the XML tree edit distances between given pair of services through XML type compatibility, token-based text and lexical similarity measurements. Besides, SAWSDL-MX1 combines logic-based and text-similarity based matching in a fixed manner: It applies five logical matching filters and ranks service offers that share the same logical matching degree with respect to a given request according to their text similarity value. The hybrid variant SAWSDL-M0+WA does the same as SAWSDL-MX1 except that its ranking of services with the same logical matching degree is according to their structural similarity value as computed by the WSDL-Analyzer. Finally, the adaptive hybrid variant SAWSDL-MX2 computes three kinds of semantic matching, logical, text and structural similarity-based. In addition, it learns the optimal aggregation of these different types of semantic matching to decide on the semantic relevance of a service to a given request.
One major question concerns the practical applicability of these different matchmakers in general, not restricted to some given domain-specific and/or very small-sized scenario, by means of their retrieval performance over a given initial test collection, SAWSDL-TC1, that consists of more than 900 SAWSDL services from different application domains. The results of our experiments shows that all hybrid matchmaker variants outperform the single matching type variants (logic-based or text or structural only) in terms of precision, while the performance of all three hybrid variants do not significantly differ with respect to SAWSDL-TC1.
The remainder of the paper is structured as follows. After a brief introduction to SAWSDL in section 2, the matching approach of the non-adaptive matchmaker SAWSDL-MX1 is recapitulated in section 3. Section 4 presents the structural matching of WSDL services performed by the WSDL-Analyzer tool. The adaptive aggregation of matching results based on a binary SVM-based classifier by our new adaptive matchmaker SAWSDL-MX2 is described in section 5. Results of our experimental evaluation over the given test collection SAWSDL-TC1 in terms of recall, (macro-)averaged precision and average query response time are shown in section 6. We comment on related work in section 7, and conclude in section 8.

2 Service Descriptions in SAWSDL

SAWSDL is designed as an extension of WSDL enabling service providers to enrich their service descriptions with additional semantic information. For this purpose, the notions of *model reference* and *schema mapping* have been introduced in terms of XML attributes (tags) that can be added to already existing WSDL service description elements as depicted in figure 1.

Semantic annotation of WSDL services. More precisely, the following extensions are used for semantic annotations of WSDL services:

– *modelReference*: A *modelReference* points to one ore more concepts with equally intended meaning expressed in an arbitrary semantic representation

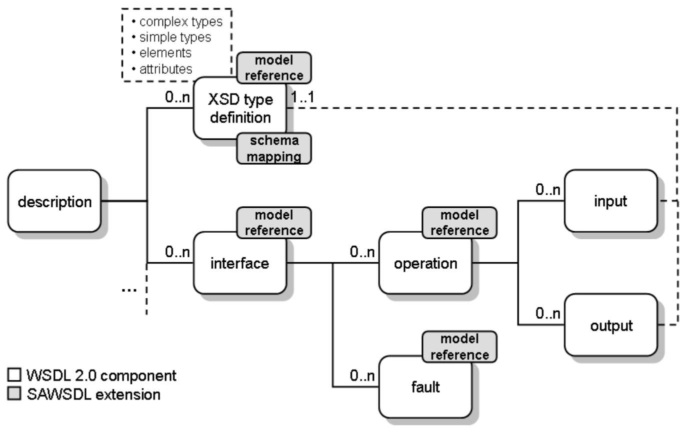


Fig. 1. SAWSDL extensions of WSDL interface components

language. They are allowed to be defined for every WSDL and XML Schema element, though the SAWSDL specification defines their occurrence only in WSDL interfaces, operations, faults as well as XML Schema elements, complex types, simple types and attributes. The purpose of a model reference is mainly to support automated service discovery.

– *liftingSchemaMapping*: Schema mappings are intended to support automated service execution by providing rules specifying the correspondences between semantic annotation concepts defined in a given ontology (the "upper" level) to the XML Schema representation of data actually required to invoke the Web service (the "lower" level), and vice versa. A *liftingSchemaMapping* describes the transformation from the "lower" level in XML Schema up to the ontology language used for semantic annotation.
– *loweringSchemaMapping*: The attribute *loweringSchemaMapping* describes the transformation from the "upper" level of a given ontology to the "lower" level in XML Schema.

However, the current specification of SAWSDL model references imposes quite some problems for semantic service matchmaking as follows.

No uniform, formal ontology language. Unlike OWL-S or WSML, the specification of SAWSDL does not restrict the developer to any uniform, formal ontology langage like OWL or WSMO. As a result, any mean of automated semantic service selection has to cope with the semantic interoperability problems of heterogeneous domain ontologies and ontology languages. While this problem could be resolved in some cases by means of syntactic and semantic transformations - such as for OWL-DL and WSML-DL - it remains hard in general.

Multiple references to different ontologies. The same holds especially for references to different kinds of ontologies like plain or structured text files, annotated image archive, or logic theories. In fact, SAWSDL allows multiple references to different kinds of ontologies for annotating even the same service description element. How shall any semantic service matchmaker know how to process them to understand the semantics of that single element? Are its annotations meant to be complementary or equivalent? If complementary, how to aggregate them, if equivalent, which one to select best for further processing? This opens up a wide range of pragmatic solutions for SAWSDL service matching.

Top-level vs bottom-level annotations. According to the SAWSDL specification, semantic annotation by means of so-called *top-level annotation* and *bottom-level annotation* shall be considered both independent from each other and applicable at the same time. While *top-level annotation* refers to the annotation of a complex type or element definition of a message parameter by means of a model reference as a whole, any *bottom-level annotation* focuses only on a single (atomic) XML element. Unfortunately, it remains unclear how to evaluate a matching *between* top-level and low-level annotated parameters, or which one to prefer if both levels of annotation are available for a complex service description element. In addition, element and type definition specifying a message component can be annotated at the same time.

Pragmatic assumptions for SAWSDL service matching. Regarding the above mentioned problems of SAWSDL service matching, the following pragmatic assumptions were made for using our SAWSDL-MX matchmaker variants SAWSDL-M0+WA, SAWSDL-MX1, and SAWSDL-MX2:

– References to formal ontologies in description logics only. The current implementation of SAWSDL-MX performs reasoning on logic-based annotations in OWL-DL[3] but is not restricted to it: It supports other description logics (DL) if they are translated into the standard DIG 1.1[4] interface representation format.
– Only top-level semantic annotations of service parameters are considered for service matching. However, any direct top-level annotation of a WSDL message part has priority over the top-level annotation of the respectively referenced (and likewisely annotated) XML-Schema element or type.
– In case of multiple annotations of a single element at the same level, one of them is selected uniformly at random. Only semantic annotations of service (IO) parameters are considered, but not annotations of entire operations or interfaces. However, the proposed matching variants could easily be adopted for this purpose.

In the following sections, we describe each of these different SAWSDL matchmaker variants and results of their comparative experimental evaluation of performance.

[3] http://www.w3.org/2004/OWL/
[4] http://dig.sourceforge.net/

3 SAWSDL-MX: Logic and Text Similarity-Based Signature Matching

In this section, we describe the hybrid service signature matching performed by both SAWSDL-MX matchmaker variants, the non-adaptive SAWSDL-MX1 and the adaptive SAWSDL-MX2. The logic-based only variant of SAWSDL-MX is called SAWSDL-M0. Service requests and offers are presumed to be formulated in SAWSDL, each comprising one or multiple operations with semantically annotated signatures.

3.1 Hybrid Service Interface Matching

For each pair of service offer O and service request R, the matchmaker determines their semantic similarity by evaluating every combination of their operations in terms of either logic-based only (SAWSDL-M0) or text similarity-based only operation matching, or both (SAWSDL-MX1, SAWSDL-MX2). The process of logic-based and text similarity-based (service) operation matching is described in more detail below.
To determine an injective mapping between service offer and request operations that is optimal regarding their matching degrees, SAWSDL-MX applies bipartite operation graph matching. Nodes in the graph represent the operations and the weighted edges are built from possible one-to-one assignments with their weights derived from the computed degree of (logical/text/hybrid) operation match. If there exists such a mapping, then it is guaranteed that there exists an operation of the service offer for every requested operation, disregarding the quality of their matching at this point.
For example, consider the service request and service offer given in figure 2. Every request operation RO_i (with $i \in \{1, 2\}$) is compared to every advertisement operation O_j (with $j \in \{1, 2, 3\}$) with respect to logic-based filters defined in the next section. In this example, RO_1 exactly matches with O_1, but fails for O_2 and O_3. O_3 is a weaker plug-in match for RO_2 (the subsumed-by match of RO_2 with O_2 is even weaker than a plug-in match). The best (max) assignment of matching operations is $\{\langle RO_1, O_1\rangle, \langle RO_2, O_3\rangle\}$.
One conservative (min-max) option of determining the matching degree between service offer and request based on their pairwise operation matchings is to assume the worst result of the best operation matchings. In other words, we guarantee a fixed *lower* bound of similarity for *every* requested operation - which is what SAWSDL-MX is doing. In the example shown in figure 2, the service offer is considered a *plug-in* match for the request. Other not yet implemented possibilities would be to merge the operation matching results based on their average syntactic similarity values and to provide more detailed feedback to the user on the operation matchings involved.

3.2 Logic-based Operation Matching

The logic-based operation matching by SAWSDL-MX bases on the successive application of the following four filters of increasing degree of relaxation to a

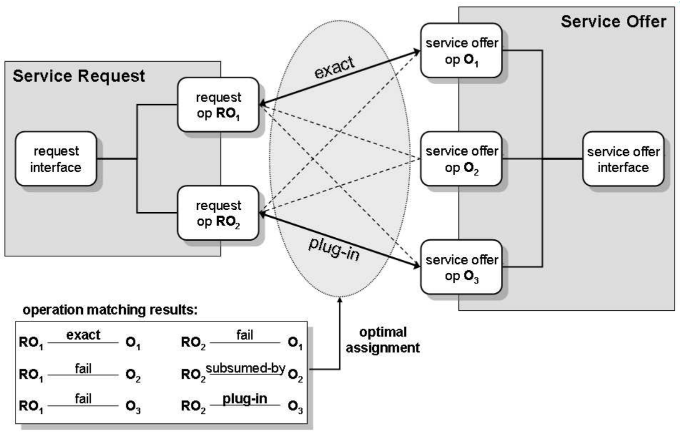


Fig. 2. Interface level matching of SAWSDL-MX

given pair service offer operation O_O and service request O_R: *Exact*, *Plug-in*, *Subsumes* and *Subsumed-by*. These filters have been originally developed for the matchmaker OWLS-MX but extended with bipartite concept graph matching to ensure an injective mapping between I/O concepts of service offer and request, whenever possible. As an overview to description logic and DL reasoning, we refer to [9]. The sets $lgc(c)$ and $lsc(c)$ contain the least generic concepts of c (direct parent) and the least specific concepts of c (direct child), respectively.

Exact match: Service operation O_O *exactly* matches service operation $O_R \Leftrightarrow (\exists$ injective assignment $M_{in} : \forall m \in M_{in} : m_1 \in in(O_O) \wedge m_2 \in in(O_R) \wedge m_1 \equiv m_2) \wedge (\exists$ injective assignment $M_{out} : \forall m \in M_{out} : m_1 \in out(O_R) \wedge m_2 \in out(O_O) \wedge m_1 \equiv m_2)$. There exist a one-to-one mapping of perfectly matching inputs as well as perfectly matching outputs. Assuming that an operation fullfills a requesters need if every input can be satisfied and every requested output is provided, the assignments only require to be injective (but not bijective), thus additional available information not required for service invocation and additional provided outputs not explicitly requested are tolerated.

Plug-in match: Service operation O_O *plugs into* service operation $O_R \Leftrightarrow (\exists$ injective assignment $M_{in} : \forall m \in M_{in} : m_1 \in in(O_O) \wedge m_2 \in in(O_R) \wedge m_1 \sqsupseteq m_2) \wedge (\exists$ injective assignment $M_{out} : \forall m \in M_{out} : m_1 \in out(O_R) \wedge m_2 \in out(O_O) \wedge m_2 \in lsc(m_1))$. The filter relaxes the constraints of the *exact* matching filter by additionally allowing input concepts of the service offer to be arbitrarily more general than those of the service request, and advertisement output concepts to be direct child concepts of the queried ones.

Subsumes match: Service operation O_O *subsumes* service operation $O_R \Leftrightarrow (\exists$ injective assignment $M_{in} : \forall m \in M_{in} : m_1 \in in(O_O) \wedge m_2 \in in(O_R) \wedge m_1 \sqsupseteq$

$m_2) \wedge (\exists$ injective assignment $M_{out} : \forall m \in M_{out} : m_1 \in out(O_R) \wedge m_2 \in out(O_O) \wedge m_1 \sqsupseteq m_2)$. This filter further relaxes constraints by allowing service offer outputs to be arbitrarily more specific than the request outputs (as opposed to the *plug-in* filter, where they have to be direct children). Thus, a *plug-in* can be seen as special case of a *subsumes* match resulting in a more fine-grained view at the overall service ranking.

Subsumed-by match: Service operation O_O is *subsumed by* service operation $O_R \Leftrightarrow (\exists$ injective assignment $M_{in} : \forall m \in M_{in} : m_1 \in in(O_O) \wedge m_2 \in in(O_R) \wedge m_1 \sqsupseteq m_2) \wedge (\exists$ injective assignment $M_{out} : \forall m \in M_{out} : m_1 \in out(O_R) \wedge m_2 \in out(O_O) \wedge m_2 \in lgc(m_1))$. The idea of the *subsumed-by* matching filter is to determine the service offers that the requester is able to provide with all required inputs and at the same time deliver outputs that are at least closely related to the requested outputs in terms of the inferred concept classification.

The matching degree of *fail* is true if and only if none of the matching filters defined above succeed. As a result, services are ranked according to their matching degree in the following decreasing order: *exact* > *plug-in* > *subsumes* > *subsumed-by* > *fail*.

SAWSDL-M0. The logic-based only matchmaker variant SAWSDL-M0 applies the above logical matching filters only, and ranks service offers that share the same logical matching degree with respect to a given request uniformly at random.

3.3 Text Similarity-Based Operation Matching

The hybrid variants of SAWSDL-MX also perform complementary text similarity-based matching by means of classical token-based text similarity measures *Loss-of-Information*, *Extended Jaccard*, *Cosine* and *Jensen-Shannon* as implemented, for example, in SimPack[5]. For this purpose, the signatures of both request and offer are considered as text such that the degree of semantic similarity is measured in terms of their averaged text similarity.
More concrete, each semantic service signature is transformed into a pair of weighted keyword vectors for input, respectively, output - according to the classical vector space model of information retrieval. For this purpose, each input concept is logically unfolded in the shared ontology (as defined for standard tableaux reasoning algorithms) and concatenated with all others to a complex logical expression containing only primitive components and logical operators. This expression is treated as mere text string being processed to a TFIDF weighted keyword vector; the same is done with service output concepts. The TFIDF term weighting values are computed over two distinct text indices depending on whether service inputs or outputs are compared.

[5] http://www.ifi.uzh.ch/ddis/research/semweb/simpack/

SAWSDL-MX1. The hybrid semantic service matchmaker SAWSDL-MX1 applies the logical matching filters mentioned above and ranks service offers that share the same logical matching degree with respect to a given request according to their text similarity value.

4 WSDL-Analyzer: Structural WSDL Matching

The WSDL-Analyzer (WA) tool introduced in [12] performs a strucrual only matching of WSDL 1.1 services. In fact, it ignores the semantic annotations of SAWSDL descriptions and treats a SAWSDL description as a mere WSDL description. The WA tool detects similarities and differences between WSDL files and can be used to find a list of non-logic-based semantically relevant Web services. Since its similarity algorithm produces a mapping between WSDL service descriptions, the tool can also be used for supporting mediation between services. More concrete, the WA tool exploits various types of schema information such as element names, datatypes and structural properties, and characteristics of data instances, as well as background knowledge from dictionaries and thesauri. The similarity algorithm calculates the similarity between the XML structures of a requested and a candidate services, respects the structural information of complex datatypes and is flexible enough to allow for relaxed matching as well as matching between parameters that come in different orders in service parameter lists.

The comparison of two WSDL services is a multi-step process. It starts off with (1) the comparison of the operation sets offered by the services, which is based on (2) the comparison of the structures of the operations input and output messages, which, in turn, is based on (3) the comparison of the datatypes of the objects communicated by these messages. The recursive structural matching of two XML-based WSDL service descriptions is performed as follows.

A WSDL description is represented as a labelled tree where leaf nodes are the basic built-in datatypes provided by the XML schema specification[6]. Let $L = \{l_1, l_2, ..., l_n\}$ be a set of labels.

A **labelled tree** $T = (N, E, root(T), \varphi)$ is an acyclic, connected graph with:

- $N = \{n_1, n_2, ..., n_n\}$ is a set of nodes.
- $E \subset N \times N$ is a set of edges.
- $root(T)$ the root of the tree.
- $\varphi : N \to L$ is a function which assigns a label to each node with basic datatypes $D \subset L$.

The process of calculating the similarity of two trees T_1 and T_2 starts with the roots $root(T_1)$ and $root(T_2)$, and traverses these trees recursively:

For $a \in N_{T_1}$ and $b \in N_{T_2}$ do compute,

$$sim(a,b) = \begin{cases} \omega_{name} * sim_{name}(\varphi(a), \varphi(b)) & \\ +\omega_{struct} * max(\oplus_{i,j}(sim(n_i, m_j))), & \varphi(a), \varphi(b) \notin D \\ sim_{type}(\varphi(a), \varphi(b)), & \varphi(a), \varphi(b) \in D \end{cases} .$$

[6] http://www.w3c.org/TR/xmlschema-2/

where $(a, n_i) \in E_{T_1}$, $(b, m_j) \in E_{T_2}$ and $\oplus_{i,j}(n_i, m_j)$ denotes the sum of pairs $sim(n_i, m_j)$ for $1 \leq i \leq card(n)$ and $1 \leq j \leq card(m)$ such that each n_i and m_j occur at most once in the sum. If $card(n) \neq card(m)$, some of the nodes cannot be matched. Weights ω_{name} and ω_{struct} are used to either increase or decrease the effect of element (label) name or structural similarity. Computation of type similarity sim_{type} bases on a given type compatibility table which assigns a value to each combination of basic data types. The similarity of labels sim_{name} can be calculated with different measures such as string edit distance, substring containment or WordNet[7] similarity (semantic proximity). In order to improve the mapping results, we used substring matching and WordNet. Experiments showed that especially in rather standardized areas the results are better than with pure data type mapping.

SAWSDL-M0+WA. The hybrid semantic service matchmaker SAWSDL-M0+WA applies the logical matching filters only, and ranks service offers that share the same logical matching degree with respect to a given request according to their degree of structural similarity as computed by the WSDL-Analyzer.

5 SAWSDL-MX2: Adaptive Matching Aggregation

Inspired by [2, 4], we developed an adaptive, hybrid semantic service matchmaker SAWSDL-MX2. This matchmaker (a) separately computes three different kinds of semantic service matching degrees, that are logical, text and structural matching (each of them as described in previous sections), and then (b) learns over a given traing set how to best aggregate them to decide on the semantic relevance (ranking) of a service to a given request. For the latter purpose, SAWSDL-MX2 exploits a Support Vector Machine (SVM) to learn a binary classification function, which is characterized by a hyperplane in a given feature space. This classification function evaluates for any given pair of service offer and request, their binary relevance class membership, i.e. *relevant* or *irrelevant*, for the matching problem at hand. For result ranking, the distances of training samples to this plane are computed, and then taken as reference similarity values for deciding on the relevance of unknown services.
More concrete, let $X = \{0, 1\}^5 \times [0, 1] \times [0, 1]$ feature space with feature vectors $x_i = (f_1, f_2, f_3, f_4, f_5, f_6, f_7)$ feature vectors where each feature f_1 to f_5 represents the logic-based matching result for a service query/offer pair including *fail*, feature f_6 stands for the text-based similarity value, and f_7 for the structural similarity value computed by the WSDL-Analyzer. For example, the feature vector $(0, 0, 1, 0, 0, 0.6, 0.7)$ represents the hybrid semantic matching result computed by SAWSDL-MX as follows: A logical *subsumes* match with text similarity of 0.6 and structural similarity of 0.7. The y_i values of the training samples equal -1 for an irrelevant service offer given a query, and 1 for the relevant case. As usual, relevance sets in the test collection are subjectively defined

[7] http://wordnet.princeton.edu/

by domain experts. Eventually, the input to the SVM of SAWSDL-MX2 is a set of training examples $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ with $x_i \in X$ and $y_i \in \{-1, 1\}$. The result of running a SVM on such input is a hyperplane, possibly in a higher dimensional space, which separates training examples in the feature space as good as possible while the distance of the nearest points of each category is maximized to avoid biased categorization. This is expressed in the following optimization problem:

$$\text{minimize } w,b,\zeta\text{: } \frac{1}{2} w^T w + C \sum_{i=1}^{N} \zeta_i$$

$$\text{subject to } \forall 1 \leq i \leq N : y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \zeta_i \geq 0,$$

where w and b define the optimal hyperplane according to the previously mentioned characteristics. The error term $C \sum_{i=1}^{N} \zeta_i$ is introduced to allow for outliers in a non-linear separable training set, where the error penalty parameter C must be specified beforehand. ϕ is a predefined function which maps features into a higher dimensional space. There exists also a dual problem description, which utilizes *Lagrange multipliers* to express the hyperplane as linear combination of support vectors. This form allows for the introduction of a *kernel* function K, which implicitly defines the original function ϕ of the primal problem. For our experiments, the *radial basis function* (RBF) has been used as kernel. It is controlled by a second parameter γ and is defined as follows: $K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$. To find a good parameter setting (C, γ), the n-fold cross validation and grid-search approach proposed in [13] has been conducted. For more details on SVM's in general and on the dual problem solving, we refer the interested reader for example to [14].

Implementation of SAWSDL-MX2. SAWSDL-MX2 has been fully implemented in Java using the sawsdl4j[8] API (handling SAWSDL for WSDL 1.1) and the OWL API[9] for access to SAWSDL and OWL files, the DIG 1.1 as standard interface to handle SHOIQ knowledge base queries, and the Pellet[10] reasoner as inference engine for logic-based matchmaking. As SVM implementation, we used libSVM[11].

6 Evaluation of Performance

For the adaptive integration of SAWSDL-MX and WSDL Analyzer by SAWSDL-MX2, a retrieval performance evaluation based on the well known measures recall and precision has been conducted. To prove the statistical significance of different matching variants, we applied the *Friedman Test*. In the following, we focus

[8] http://knoesis.wright.edu/opensource/sawsdl4j/
[9] http://owlapi.sourceforge.net/
[10] http://pellet.owldl.com/
[11] http://www.csie.ntu.edu.tw/ cjlin/libsvm/

on the comparative evaluation of the three SAWSDL-MX variants described in previous sections: the non-adaptive SAWSDL-M0+WA, SAWSDL-MX1 and the adaptive SAWSDL-MX2. For more detailed results on SAWSDL-MX1 alone, we refer to [1].

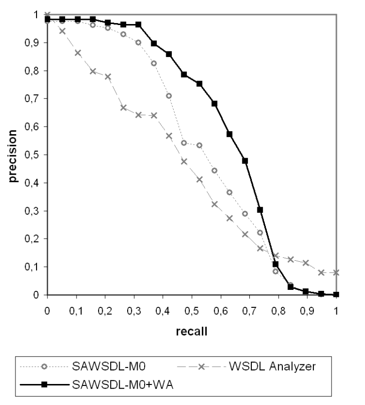
6.1 Evaluation Setup

The experimental evaluation of service retrieval performance is based on the first SAWSDL test collection SAWSDL-TC1. It is semi-automatically derived from OWLS-TC 2.2[12] using the OWLS2WSDL[13] tool, as there is currently no other standard test collection for SAWSDL available. OWLS2WSDL transforms OWL-S service descriptions (and concept definitions relevant for parameter description) to WSDL through syntactic transformation. Top-level annotations taken from the original OWL-S descriptions have been added for XML Schema type definitions used to describe message inputs and output. The collection consists of around 900 Web services covering different application domains: education, medical care, food, travel, communication, economy and weaponry. It also includes a set of queries and binary relevance sets subjectively specified by domain experts. As one result, each service in SAWSDL-TC1 contains only a single interface with one operation. All automatically derived model references point to OWL ontologies. Therefore, this test collection can only be seen as a first attempt towards a commonly agreed testing environment for SAWSDL service discovery and our evaluation has to be considered as preliminary. The performance tests have been conducted on a machine with Windows 2000, Java 6, 1,7 GHz CPU and 2 GB RAM using the SME^2 tool[14] for automated evaluation.

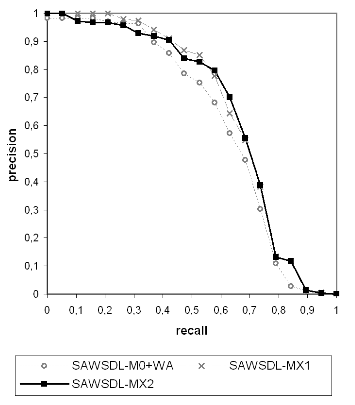
6.2 Performance Tests

For retrieval performance evaluation, we measured precision and recall: $Prec = \frac{|A \cap B|}{|B|}$ and $Rec = \frac{|A \cap B|}{|A|}$, where A is the set of all relevant documents, and B the set of all retrieved documents for a request. Further, we measured the macro-averaged precision: $Prec_i = \frac{1}{|Q|} \cdot \sum_{q \in Q} \max\{P_o | R_o \geq Rec_i \wedge (R_o, P_o) \in O_q\}$, where O_q denotes the set of observed pairs of recall/precision values for query q when scanning the ranked services in the answer set for q stepwise for true positives in the relevance sets of the test collection. For evaluation, the answer sets are the sets of all services registered at the matchmaker which are ranked with respect to their (totally ordered) matching degree. In other words, we computed the mean of precision values for answer sets returned by the matchmaker for all queries in the test collection at standard recall levels Rec_i $(0 \leq i < \lambda)$. Ceiling interpolation is used to estimate precision values that are not observed in the answer sets for some queries at these levels; that is, if for some query there is no precision value at some recall level (due to the ranking of

[12] http://projects.semwebcentral.org/projects/owls-tc/
[13] http://projects.semwebcentral.org/projects/owls2wsdl/
[14] http://projects.semwebcentral.org/projects/sme2/



(a) integration vs. basic strategies



(b) different integration variants

Fig. 3. Performance of matching variants

services in the returned answer set by the matchmaker) the maximum precision of the following recall levels is assumed for this value. The number of recall levels from 0 to 1 (in equidistant steps $\frac{n}{\lambda}, n = 1 \dots \lambda$) we used for our experiments is $\lambda = 20$. The *Average Precision* (AP) measure produces a single-valued rating of a matchmaker for a single query result: $AP = \frac{1}{|R|} \sum_{r=1}^{|L|} isrel(r) \frac{count(r)}{r}$, where R is the set of relevant items previously defined by domain experts for the examined query, L the ranking of returned items for that query, $isrel(r) = 1$ if the item at rank r is relevant and 0 otherwise and $count(r)$ the number of relevant items found in the ranking when scanning top-down, i.e. $count(r) = \sum_{i=1}^{r} isrel(i)$. The AP measure is independent from the way and size of ranking.

As a first experiment, we compared the retrieval performance of SAWSDL-M0+WA to that of both approaches applied solely. This experiment was conducted mainly to check, whether even such a simple hybrid combination of logic-based and non-logic-based semantic matching as in SAWSDL-M0+WA can improve upon the performance of each of both (SAWSDL-M0 and Text-IR) individually. As shown in figure 3(a), the combination of both performs best at almost every recall level except towards full recall. This is in perfect line with our experimental results on SAWSDL-MX1 reported in [1]. As we already pointed out there, ontologies currently found in the Web are merely inclusion hierarchies or taxonomies rarely making use of elaborated logical concept definitions for service annotation, which still dampens the benefit of any logic-based semantic matching approach.

To compare the performance of the adaptive hybrid matchmaker SAWSDL-MX2 (logic, text, structural similarity) with that of the non-adaptive variants SAWSDL-M0+WA (logic and structural similarity) and SAWSDL-MX1 (logic and text similarity), we conducted a second evaluation experiment. As shown figure in 3(b), the adaptive SAWSDL-MX2 performs better than SAWSDL-M0

	SAWSDL-M0+WA	SAWSDL-MX1	SAWSDL-MX2
AP	0.61	0.66	0.65
AQRT	15.48s	8.17s	18.8s

Table 1. Average precision and query response time (in seconds) of SAWSDL-MX matchmaker variants

(logic-based only) but is as good as the non-adaptive variant SAWSDL-MX1 utilizing logic-based matching and extended Jaccard text similarity-based matching.
This is mainly to the fact, that text similarity computation as described in section 3.3 is closely related to structural matching when applied to mere is-a ontologies (inclusion hierarchies, taxonomies). In fact, for the given test collection, where SAWSDL files have been semi-automatically derived from OWL-S and the XML Schema parameters origin from OWL concept definitions, the WSDL-Analyzer (WA) indirectly performs both structural and text similarity-based concept matching which makes it partly redundant to SAWSDL-MX2 in such cases. Nevertheless, for the general case, the adaptive approach of SAWSDL-MX2 enables an easy and well-defined integration of arbitrary matching mechanisms to improve result rankings.
Table 1 summarizes the averaged precision (AP) and query response time of the discussed SAWSDL matchmakers. We emphasize that these results strongly depend on the test collection used. In summary, the non-adaptive SAWSDL-MX1 (logic + text) still performs best even over the adaptive variant SAWSDL-MX2 which exhibits a longer response time in average due to its comparatively most complex matching (logic + text + structural).

6.3 Statistical Significance Tests

The differences in the performance evaluation results can be shown to be statistically significant or insignificant by means of the so-called Friedman test. This is a non-parametric test for simultaneously analyzing ranked result sets of at least two different (service matching) methods and has been shown in [11] to be a vital explanatory component of a comparative retrieval performance evaluation. We are using the Friedman Test variant proposed in [10] as $F_N = \frac{MSR}{MSE}$, where MSR is the mean-squared difference between the different matching variants and MSE the mean-squared error. The resulting value can be compared to the F-distribution with $m-1$ and $(n-1)(m-1)$ degrees of freedom, where n is the number of queries anf m the number of tested matching variants. The resulting *p-value* indicates, if there is a significant difference between the variants which one can not interpret as being an implication of the *null hypothesis*, i.e. that variations of the matchmaker rankings per query are insignificant. As a threshold value for p, we rely on $\alpha = 0.05$, which is frequently used for tests like this. To produce the rankings for the test, averaged AP values have been used. The test resulted in a value of $p = 0.026$ for SAWSDL-M0+WA compared to the WSDL-Analyzer, and $p = 0.0028$ compared to SAWSDL-M0. Both values are below the threshold α which means that the recall/precision results are a significant improvement at 5% level. However, the test returned a value of $p = 0.331$ for the second evaluation experiment which means that none of the evaluated matchmaker variants performs significantly better than any of the others regarding the used test collection. As already meantioned before, this is mainly due to the fact, that the additional structural comparison implicitly performs partly redundant matching, as the semi-automatically derived test collection services mainly differ with respect to the XML Schema parameter definitions derived from the original OWL concepts used in the original test collection OWLS-TC 2.1.

7 Related Work

To the best of our knowledge, there exist only very few implemented semantic service discovery systems for SAWSDL. In FUSION [7], any SAWSDL service description is classified at the time of its publishing and then mapped to UDDI to allow for fast lookups. In case of unknown semantic service requests reasoning has to be done at query time. In contrast to SAWSDL-MX, the FUSION discovery relies on one infered logical matching degree only, Like SAWSDL-MX, FUSION is strictly bound to OWL-DL. Lumina [8] developed in the METEOR-S project[15] follows a similar approach based on mapping of WSDL-S (and later on SAWSDL respectively) to UDDI but performs syntactic (structural and text similarity-based) service matching only. Finally, the URBE matchmaker by Plebani[16] performs non-logic-based matching in terms of text similarity and structural comparisons. Unfortunately, no public information is available on this matchmaker yet. For a survey of semantic service matchmakers in general, we refer the interested reader to [6].

8 Conclusion

We discussed three different hybrid SAWSDL service matchmakers all of which outperforming the individual types of matching they combine for detecting the semantic relevance of a service with interfaces with multiple operations to a given request. The combination with structural service matching by the WSDL-Analyzer tool turned out to be of benefit compared to logic-based only matching but not with respect to logical and text similarity-based matching. In addition, applied to the given test collection SAWSDL-TC1, the adaptive hybrid approach combining all three types of matching did not outperform the non-adaptive variant in [1] yet. We are currently working on a new hybrid matchmaker variant SAWSDL-MX3 that also supports different knowledge representation formalisms. SAWSDL-MX1 and SAWSDL-TC1 are publicly available at *semwebcentral.org*.

[15] http://lsdis.cs.uga.edu/projects/meteor-s/
[16] http://www-ags.dfki.uni-sb.de/ klusch/s3/html/2008.html

References

1. Klusch, M., Kapahnke, P.: Semantic Web Service Selection with SAWSDL-MX. Proceedings of the 2nd Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2), 2008
2. Joachims, T., Radlinski, F.: Search Engines that Learn from Implicit Feedback. Computer Volume 40, Issue 8, Aug. 2007 Page(s):34 - 40, 2007
3. Kaufer, F., Klusch, M.: WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker. Proceedings of the 4th IEEE European Conference on Web Services (ECOWS 2006), IEEE CS Press, Zurich, Switzerland, 2006
4. Kiefer, C., Bernstein, A.: The Creation and Evaluation of iSPARQL Strategies for Matchmaking. Proceedings of the 5th European Semantic Web Conference (ESWC), Lecture Notes in Computer Science, Vol. 5021, pages 463–477, Springer-Verlag Berlin Heidelberg, 2008
5. Klusch, M., Fries, B., Sycara, K.: Automated Semantic Web Service Discovery with OWLS-MX. Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Hakodate, Japan, ACM Press, 2006
6. Klusch, M.: Semantic Web Service Coordination. In: M. Schumacher, H. Helin, H. Schuldt (Eds.) CASCOM - Intelligent Service Coordination in the Semantic Web. Chapter 4. Birkhuser Verlag, Springer, 2008
7. Kourtesis, D., Paraskakis I.: Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery. Proceedings of the 5th European Semantic Web Conference (ESWC 2008), Lecture Notes in Computer Science (LNCS), vol. 5021, Springer-Verlag Berlin Heidelberg, pp. 614628, 2008
8. Li, K., Verma, K., Mulye, R., Rabbani, R., Miller, J. A., Sheth, A. P.: Designing Semantic Web Processes: The WSDL-S Approach. Chapter submitted to Semantic Web Processes and Their Applications. J. Cardoso, A. Sheth, Editors. Springer
9. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider P.F.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press 2003, ISBN 0-521-78176-0
10. Iman, R.L., Davenport, J.M.: Approximations of the critical region of the friedman statistic. Communications in Statistics, A9:571–xxx, 1980
11. Hull, D.: Using statistical testing in the evaluation of retrieval experiments. Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval, pages 329–338, 1993
12. Zinnikus I., Rupp H.J., Fischer K.: Detecting Similarities between Web Service Interfaces: The WSDL Analyzer. Second International Workshop on Web Services and Interoperability (WSI'2006), Pre-conference Workshop of Conference on Interoperability for Enterprise Software and Applications, I-ESA'06, March 20th - 21st, Bordeaux, 2006.
13. Hsu C.-W., Chang C.-C., Lin C.-J.: A Practical Guide to Support Vector Classification. 2007
14. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. Proceedings of the fifth annual workshop on Computational learning theory Pittsburgh, Pennsylvania, United States, pages 144–152, 1992