Adaptive Hybrid Semantic Selection of SAWSDL Services With SAWSDL-MX2

Matthias Klusch, Patrick Kapahnke, Ingo Zinnikus
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, 66123 Saarbruecken, Germany
Email: {klusch, patrick.kapahnke, ingo.zinnikus}@dfki.de

ABSTRACT
We present an adaptive, hybrid semantic matchmaker for SAWSDL services, called SAWSDL-MX2. It determines three kinds of semantic matching of an advertised service with a requested one both of which are described in the standard SAWSDL: Logic-based, text-similarity-based and XML-tree edit-based structural similarity. Before selection, SAWSDL-MX2 learns the optimal aggregation of these different matching degrees off-line over a random subset of a given SAWSDL service retrieval test collection by exploiting a binary support vector machine-based classifier with ranking. Finally, we present a comparative evaluation of the retrieval performance of SAWSDL-MX2. The matchmaker is available at semwebcentral.org.

Keywords: *Semantic services, Matchmaking*

INTRODUCTION
Semantic service selection is commonly considered key to the discovery of relevant services in the semantic Web, and there are already quite a few matchmakers available for this purpose and different formats like OWL-S, WSML and SAWSDL (Klusch 2008). As a W3C recommendation dated August 28, 2007, the SAWSDL[i] (Semantic Annotations for WSDL) specification proposes mechanisms to enrich Web services described in WSDL[ii] (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 semantic services in SAWSDL (Klusch & Kapahnke 2008) adopted the ideas of our hybrid matchmakers OWLS-MX and WSMO-MX (see Klusch et al. 2009a; Kaufer & Klusch 2006) for semantic services in OWL-S, respectively, WSML.

However, SAWSDL-MX1 focuses 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 (Zinnikus et al. 2006) by means of measuring 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 matchmaker variant SAWSDL-MX2 computes three kinds of semantic matching, logical, text and structural similarity-based. In addition, it learns the optimally weighted aggregation of these different types of semantic matching to decide on the semantic relevance of a service to a given request.

One major advantage of this off-line learning is that it renders SAWSDL-MX2, in principle, independent from any given service test collection or future extensions with other matching filters. In fact, the configuration of any non-adaptive matchmaker such as SAWSDL-MX1 would have to be manually retuned by the developer of the matchmaker to reflect such changes.

Whether this adaptation feature may even improve the precision of non-adaptive variants in practice has been checked by us against the only publicly available SAWSDL service retrieval test collection SAWSDL-TC1 consisting of more than 900 SAWSDL services from different application domains. The results of our experiments show that all hybrid semantic service matchmaker variants outperform the single matching type variants (logic-based or text similarity or structural XML similarity only) in terms of precision, while all SAWSDL matchmaker variants available today, whether adaptive or not, do not significantly differ from each other in terms of their precision with respect to this collection SAWSDL-TC1.

The remainder of the paper is structured as follows. After a brief introduction to SAWSDL in the following section, the SAWSDL service matching approach of the non-adaptive matchmaker SAWSDL-MX1 is recapitulated. The subsequent section presents the structural matching of Web services in WSDL performed by the WSDL-Analyzer tool, followed by an illustration of the application of all three matching filters by example. The adaptive aggregation of different matching results based on an off-line learned binary Support Vector Machine (SVM) classifier with ranking by the adaptive matchmaker SAWSDL-MX2 is described thereafter. We briefly present implementation details and then report the results of our experimental evaluation over the public test collection SAWSDL-TC1 in terms of macro-averaged recall/precision, average precision and average query response time. Eventually, we comment on related work on SAWSDL service matchmaking and conclude. This paper is an extended version of (Klusch et al. 2009).

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 including XML Schema definitions for message parameters 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 or more concepts with equally intended meaning expressed in an arbitrary semantic representation 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.

[Insert figure 1 here]

Figure 1. SAWSDL extensions of WSDL service interface components.

However, the current specification of SAWSDL model references poses 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 language like OWL or as defined as part of WSML. 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[iii]? If complementary, how to aggregate them, if equivalent, which one to select best for further processing? This opens up a wide range of differing 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.

[Insert figure 2 here]

Figure 2. Example of pragmatic assumptions for service selection by SAWSDL-MX.

Pragmatic assumptions for SAWSDL service matching by SAWSDLMX. Regarding the above mentioned problems of SAWSDL service matching, the following pragmatic assumptions, illustrated in figure 2 above, were made for using all members of our SAWSDL-MX matchmaker family, that is the non-adaptive matchmakers SAWSDL-MX1, its variant SAWSDL-M0+WA, and the adaptive matchmaker SAWSDL-MX2 each of which we will describe in subsequent sections:

References to formal ontologies in description logics only. The current implementation of SAWSDL-MX performs reasoning on logic-based annotations in OWL-DL[iv] but is not restricted to it: It supports other description logics (DL) if they are translated into the standard DIG 1.1[v] interface representation format.

- Only top-level semantic annotations of service parameters are considered for service matching. Direct top-level annotation of a WSDL message part has priority over the top-level annotation of the respectively referenced (and 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.

SAWSDL-MX1: LOGIC AND TEXT SIMILARITY-BASED SIGNATURE MATCHING

In this section, we describe the hybrid semantic service signature matching performed by the non-adaptive matchmaker SAWSDL-MX1. Its logic-based only variant is called SAWSDL-M0. Since service requests and offers are presumed to be formulated in SAWSDL, each of their interfaces comprising one or multiple operations with semantically annotated signatures, we first present the way in which SAWSDL-MX1 performs hybrid semantic matching on the service interface level based on the results of its hybrid semantic matching of pairs of operations.

Hybrid Service Interface Matching

For each pair of service offer *O* and service request *R*, the matchmaker SAWSDLMX1 first determines their semantic similarity by evaluating every combination of their operations in terms of logic-based only (SAWSDL-M0) and text similarity-based only operation matching. These processes of logic-based and text similarity-based (service) operation matching are described in more detail below.

To determine an injective mapping between service offer and request operations that is optimal regarding their matching degrees, SAWSDL-MX1 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 3. 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 $\{<RO_1,O_1>, <RO_2,O_3>\}$.

[Insert figure 3 here]

Figure 3. Interface level matching of SAWSDL-MX1 for logic-based semantic similarity.

One conservative (min-max) option of determining the matching degree between service offer and request based on their pairwise operations matching is to assume the worst result of the best operation matching results. In other words, we guarantee a fixed *lower* bound of similarity for *every* requested operation - which is what SAWSDL-MX1 is doing. In the example shown in Figure 3, 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.

Logic-based Operation Matching

The logic-based operation matching by SAWSDL-MX1 bases on the successive application of the following four filters of increasing degree of relaxation to a 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 (Baader et al. 2003). 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 exists 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}: $\forall m \in M_{out}$: $m_1 \in out(O_R) \wedge m_2 \in out(O_O) \in m_2 \in lsc(m_1)$). 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}: $\forall m \in M_{out}$: $m_1 \in out(O_R) \wedge m_2 \in out(O_O) \in m_1 \sqsupseteq m_2$). 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}: $\forall m \in M_{out}$: $m_1 \in out(O_R) \wedge m_2 \in out(O_O) \in m_2 \in lgc(m_1)$). 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*.

Text Similarity-Based Operation Matching

The hybrid variants of SAWSDL-MX1 also perform a complementary text similarity-based matching by means of classical token-based text similarity measures *Loss-of-Information*, *Extended Jaccard (Tanimoto)*, *Cosine* and *Jensen-Shannon* as implemented, for example, in SimPack[vi]. For this purpose, the semantic signatures (i.e., the semantic annotations of the I/O message parameters) of both service request and offer are considered as text such that the degree of semantic similarity is measured in terms of their averaged text similarity.

Concretely, 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 a mere text string which is 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.

The motivation for applying the above mentioned text similarity measures, originally intended for natural language document comparison, is the experimentally evidenced observation that the primitive (concept and role) components or terms that are used to describe more complex concepts logically keep to *Zipf 's Law*. That is, taking into account all *documents* derived from the logical description of the semantic service signatures provided by services taken from the

public SAWSDL service retrieval test collection SAWSDL-TC1 (this also holds e.g. for the public OWL-S service test collection OWLS-TC2), the observed distribution of terms is as follows: $f(k,s,N) = \frac{1}{k^s} \Big/ \sum_{n=1}^{N} \frac{1}{n^s}$ with k being the observed rank, N the number of ranks and s a given parameter to characterize the distribution. For natural language text, $s \approx 1$ is usually assumed, reducing the formula to a normalized variant of $1/k$, i.e. the term at rank 1 occurs two times as often as the term at rank 2, three times as often as rank 3 and so on.

[Insert figure 4 here]

Figure 4. Term distribution for service signature text documents (SAWSDL-TC1).

Figure 4 scatterplots (logarithmically scaled) one of the term indices. In fact, the term distributions for both service signature (service input, respectively, output concepts) indices support the proposition that the data generated for text similarity matching as described above resemble natural language text.

Non-Adaptive SAWSDL-MX1 Algorithm

To summarize, the hybrid semantic service matchmaker SAWSDL-MX1 first applies both logical matching and text matching to each pair of operations of each pair of advertised and requested SAWSDL service interface. It then determines the degrees of both logical and text matching for the given service pair on the interface level by means of two respective bipartite graph matchings based on these results which eventually leads to the hybrid semantic service matching degree denoted as a 2-tuple (logic-based semantic service matching degree, text similarity-based service matching value). Finally, it ranks those service offers with the same logical matching degree (i.e., the first part of the hybrid matching degree) with respect to the given request according to their text similarity values (i.e., the second part of their hybrid matching degree) in decreasing order.

SAWSDL-M0. The logic-based only matchmaker variant SAWSDL-M0 of SAWSDLMX1 only performs the logic-based semantic service matching as described above, and ranks service offers that share the same logical matching degree with respect to the given request uniformly at random.

WSDL-ANALYZER: STRUCTURAL SERVICE MATCHING

The WSDL-Analyzer (WA) tool introduced in (Zinnikus et al. 2006) performs a structural only matching of WSDL 1.1 services. That is, it ignores the semantic annotations of SAWSDL service descriptions, hence treats any SAWSDL service as a mere WSDL service. The WA tool detects similarities and differences between WSDL files to find a list of non-logic-based semantically relevant Web services. Since its similarity algorithm inherently 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 XML schema information of WSDL operation signatures 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 recursively calculates the similarity between the XML structures of a requested and a candidate service, respects the structural information of complex datatypes involved, and is flexible enough to allow for a relaxed matching as well as matching between parameters that come in different orders in service parameter lists.

WSDL-Analyzer Algorithm

The structural matching of two WSDL services is a multi-step process. It starts off with (1) comparing the operation sets of both services on interface level, which is based on (2) the comparison of the structures of the operation signatures, that is their input and output messages, which, in turn, is based on (3) the comparison of the XML Schema data types of the objects communicated by these messages. This recursive structural matching of two XML-based WSDL service descriptions is performed as follows.

A WSDL description is represented as a labeled tree where leaf nodes are the basic built-in data types provided by the XML schema specification[vii]. Let $L = \{l_1, l_2, \ldots, l_n\}$ be a set of labels. A *labelled tree* $T = (N, E, root(T), \varphi)$ is an acyclic, connected graph with:

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

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 $a \in N_{T1}$ and $b \in N_{T2}$ do compute

$$sim(a,b) = \begin{cases} \omega_n \cdot sim_n(\varphi(a),\varphi(b)) + \omega_s \cdot \max\left\{\oplus_{i,j}\ (sim(n_i, m_j))\right\} & ,\varphi(a),\varphi(b) \notin D \\ sim_t(\varphi(a),\varphi(b)) & ,\varphi(a),\varphi(b) \in D \end{cases}.$$

where $(a, n_i) \in E_{T1}$, $(b, m_j) \in E_{T2}$ 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_n and ω_s are used to either increase or decrease the effect of element (label) name or structural similarity.

The computation of type similarity sim_t bases on a given type compatibility table, that assigns a similarity value to each combination of the considered basic data types. The similarity of names (labels) sim_n can be calculated with different measures such as string edit distance, substring containment or WordNet[viii] 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.

Hybrid Semantic Matchmaker SAWSDL-M0+WA

The hybrid semantic service matchmaker SAWSDL-M0+WA performs (a) logic-based semantic matching and (b) 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.

SERVICE MATCHING EXAMPLE

In the following, we demonstrate the application of the three different semantic service matching approaches, that is logic-based, text similarity-based and structural service matching by example. Suppose a user is searching for a Web service that returns a map for a given location as input. The main parts of the description of the desired service in SAWSDL are as follows:

```
<types><schema ...>
  <complexType name="GPSType" modelReference="http://...#GPSPos">
    <sequence>
      <element name="longitude" type="float"/>
      <element name="latitude" type="float"/>
    </sequence>
  </complexType>
  <simpleType name="MapType" modelReference="http://...#Map">
    <restriction base="anyURI"/>
  </simpleType>
</schema></types>
<message name="getMapResponse">
  <part name="_Map" type="MapType"/> </message>
<message name="getMapRequest">
  <part name="Location" type="GPSType"/> </message>
<portType name="PositionMapInterface">
  <operation name="getMAP">
    <input message="tns:getMapRequest"/>
    <output message="tns:getMapResponse"/>
  </operation>
</portType>
```

Listing 1 ***Service Request***

The requested service offers exactly one operation which returns the reference to an appropriate map given a GPS position as input. As usual, the message data types of the operation signature are semantically annotated using SAWSDL model references that point to appropriate ontologies in which the semantics of the used annotation concepts are defined. In this example, the operation (service) input is claimed to conform to the concept *GPSPosition* while the output

is a *Map*, both concepts described in OWL. Besides, there is a service offer (registered at the matchmaker) which provides similar functionality:

```
<types><schema ...>
  <simpleType name="LocationType" modelReference="http://...#Location">
    <restriction base="string"/>
  </simpleType>
  <simpleType name="MapType" modelReference="http://...#RoadMap">
    <restriction base="anyURI"/>
  </simpleType>...
</schema></types>
<message name="getMAPResponse">
  <part name="MapPart" type="MapType"/> </message>
<message name="getMAPRequest">
  <part name="LocationPart" type="LocationType"/> </message>
<portType name="LocationMapInterface">
  <operation name="createMAP">
    <input message="tns:getMAPRequest"/>
    <output message="tns:getMAPResponse"/>
  </operation>
  <operation ...>...</operation>
</portType>
```

Listing 2 ***Service Offer***

During the matching process, this offer is compared to the request using one or more variants as proposed in this paper. For the logic-based and text similarity matching, the interface level matching is examined at first. Since the request only consists of a single operation, the bipartite graph matching problem is trivial and omitted here. We just assume that the second operation of the service offer provides different functionality and thus is not part of the final assignment.

Logical Service Matching

For the logic-based semantic service operation matching, the following concept definitions are assumed (written in the more compact standard DL notation instead of the normative RDF syntax of OWL-DL):

$$Map \sqsubseteq Image \sqcap \exists hasScale.Scale \sqcap \exists contains.GeographicalEntity$$

$$RoadMap \sqsubseteq Map \sqcap \exists contains.Road$$

$$Location \sqsubseteq GeographicalEntity \sqcap \exists hasParameter.GPSPos$$

$$GPSPos \sqsubseteq \exists hasLongitude.FPNum \sqcap \exists hasLatitude.FPNum$$

Sequential application of the different logic-based matching filters for this example results in a *fail*, which can be considered as *false negative*. Although the fact $RoadMap \sqsubseteq Map$ holds for the outputs, there does not exist an inclusion for the concepts *Location* and *GPSPos*. Thus, the example exposes one of the main problems with this standard matching technique. Both concepts are strongly related to each other, but logic-based concept subsumption reasoning fails to recognize that. One possibility to deal with such cases is the usage of text similarity approaches as described previously in this paper.

Text Similarity-Based Matching

The unfolded concept definitions are used as input for a text similarity measure based on the vector space model utilizing TFIDF weights over a given (service) document corpus. In this example, the matchmaker applies the common *Loss-of-Information* similarity measure which just takes the two service documents into account:

$$LOI(R,S) = 1 - \frac{\left|R \cup S\right| - \left|R \cap S\right|}{\left|R\right| + \left|S\right|},$$

where R is the set of terms of the request document and S the set of terms of the service offer. Since the proposed text similarity approach is *structured*, distinct values are computed for inputs and outputs respectively.

For example, the set of terms of inputs of the request after unfolding and stopword elimination is R_{in} = {*hasLongitude, FPNum, hasLatitude*} while the corresponding set for the service offer is S_{in} = {*Location, GeographicalEntity, hasParameter, hasLongitude, FPNum, hasLatitude*}. The partial similarity result restricted to these service inputs then is $LOI(R_{in}, S_{in}) \approx 0.66$, which indicates a moderate textual similarity (in contrast to the logic-based *fail*). For the outputs, the similarity value is computed as $LOI(R_{out}, S_{out}) \approx 0.92$, which indicates high similarity (only the single term *Road* distinguishes both sets). The overall structured textual similarity of both services R and S is the average of both values, that is $LOI(R, S) = 0.75$, which is clearly in the upper part of the similarity function range, thus compensates the logic-based *false negative*.

Structural Service Matching

To illustrate the functionality of the structural WSDL matching described in Section 4, the similarity computation of the two operations in question is exemplified here. Please note, that the actual similarity computation of the WSDL Analyzer (WA) involves the whole document structure, but this is an analogous process omitted here for reasons of simplicity.

Basically, the WA algorithm recurs into the lower level WSDL tree representation nodes of both service descriptions until a leaf node containing a basic data type occurs. If this is also the case for the other service at the same time, the data type similarity is looked up in a predefined table. If a basic data type occurs on one hand but some complex structure at the other hand, the system tries to find the best matching simple type contained in the complex type. Recursion takes place at every pair of nodes to compare structures that are not leaf nodes in the corresponding WSDL trees. At this step, the label similarity is computed for these nodes and the *maximum* of possible similarities for subnode recursion is added to the overall similarity.

[Insert figure 5 here]

Figure 5. Structural WSDL service operation matching example.

For our example of service request and offer (cf. Listings 1 and 2, Figure 5), the operation similarity (sub-) computation involves comparing *GPSType* to *LocationType*. Since *GPSType* is a complex type and *LocationType* a leaf node, simple types contained in *GPSType* are checked for type similarity resulting in 0.8 as stated in the lookup table (both simple subtypes in question are *float* and for the offer *string* was found).

By comparing their labels using WordNet, a relationship is discovered. Also, upper level nodes have a high textual similarity in most cases for the example, e.g. *getMapRequest* and *getMAPRequest*. For the output part of the operation, the algorithm ends up in comparing the two leaf nodes labeled *MapType* in both cases. Both also share the same simple datatype *anyURI*, thus these branches add a high similarity value to the overall structural similarity value.

Similarity	$w_n \cdot sim_n$	$w_s \cdot \max\{\oplus_{i,j}\}$
sim(GPSType, LocationType)	1.5	sim_t(float, string) = 0.8 (leaf/non-leaf)
sim(_Location, LocationPart)	2	2.3
sim(getMapRequest, getMAPRequest)	4	4.3
sim(<input>, <input>)	0 (no names)	8.3
sim(MapType, MapType)	3	sim_t(anyURI, anyURI) = 1
sim(Map, MapPart)	2	4
Sim(getMapResponse, getMAPResponse)	4	6
sim(<output>, <output>)	0 (no names)	10
sim(getMAP, createMAP)	**2**	**18.3**

Table 1 ***Structural similarity values computed by the WSDL-Analyzer (WA)***

Table 1 summarizes all similarity computations of the WA assuming that both weighting parameter values (ω_n, ω_s) are equal to 1. For computing the name similarity, WA applies a simple token-based string comparison in combination with WordNet matching. The token matching adds a value of 1 for each matched token to the overall similarity, WordNet matching ranges to an interval [0, 1], and both results are added for the overall sim_n value. The given lookup table for data type matching (sim_t) assigns similarity values in [0, 1] where equal types have a similarity value of 1.

As can be seen from the bottom line of Table 1, the overall result for the example request and service offer pair is 20.3, which is normalized by the maximum obtainable structural matching value of 28 to 0.73. Obviously, each of the different semantic matching approaches above has its pros and cons. While logic-based semantic matching takes advantage of formal semantic definitions of concepts and roles, it may be mislead in some cases as for the concepts *GPSPos* and *Location*. The additional use of text IR techniques and structural matching may overcome this problem in this example: Text similarity of 0.75 and structural similarity of 0.73 are both sufficiently high enough to indicate semantic relevance. However, as noted for example in (Klusch et al. 2009), each of these non-logic-based semantic matching approaches can, in principle, introduce misclassifications on their own.

SAWSDL-MX2: ADAPTIVE MATCHING AGGREGATION

As mentioned above, the logic-based filters of SAWSDL-MX1 have been complemented by text similarity matching in a fixed, that is, in a non-adaptive way. In particular, services were ranked according to their logical matching degree first and then (within the same logical matching class) according to their signature text similarity. Obviously, there are some problems with this approach. The first problem is the tedious search for a text similarity threshold parameter value for arbitrary pairs of service offers and requests. Second, more general, how to best combine different semantic service matching filters such as those described in previous sections to obtain a reasonable retrieval performance in terms of precision and recall? In particular, how to achieve this in a way that renders the matchmaker independent from any service collection in principle?

One option to resolve these issues is to let the matchmaker learn how to do it rather than finding a solution by hand. Inspired by the work of (Joachims & Radlinski 2007) and (Kiefer & Bernstein 2008) on off-line adaptive search, we developed an off-line adaptive hybrid semantic matchmaker for SAWSDL services, called SAWSDL-MX2, that simply learns how to optimally solve the just mentioned problems with fixed hybrid matchmaking by means of a support vector machine-based classifier that is trained and cross-validated over a given test collection and then applied to the selection process for a service request at hand.

SAWSDL-MX2 Overview

In short, the SAWSDL-MX2 matchmaker returns a ranked list of relevant services *S* for a given request *R* in SAWSDL based on the aggregated results of separately performed logical, text and structural similarity-based matching. Each of these different matching filters has been described above for SAWSDL-MX1 and the WSDL-Analyzer tool. Their aggregation by SAWSDL-MX2 is optimal with respect to average classification accuracy according to its binary SVM-classifier that has been learned over a given training set previously. In the following, we describe the

training (off-line learning) phase and the subsequent use of the learned SVM-classifier for hybrid semantic service selection by SAWSDL-MX2 in more detail.

Off-Line Learning of SVM Classifier

The problem of classifying a given service *S* with respect to its semantic relevance to a given request *R* can be re-formulated as the problem of learning a binary support vector machine-based classifier. That is to find a separating hyperplane in a given feature space *X* such that for all positive and negative training samples with minimal distances (these particular samples are also called support vectors) to it, these distances are maximal.

In case of SAWSDL-MX2, we consider a 7-dimensional feature space $X = \{0, 1\}^5 \times [0, 1] \times [0, 1]$, where each of the first five binary dimensions corresponds to the occurrence of one out of five different logical matching degrees (*exact*, *plug-in*, *subsumes*, *subsumed-by*, *fail*) followed by the two real-valued dimensions for text, respectively, structural similarity-based matching degrees. For example, the feature vector $x_i = (0, 0, 1, 0, 0, 0.6, 0.7)$ ($i \leq N$, N is the size of the training set of positive and negative samples) indicates that the matching results for the service offer/request pair (S, R) that corresponds to the training sample (x_i, y_i) (with $y_i = 1$ if S is relevant to R according to the binary relevance sets defined in the test collection SAWSDL-TC1, else $y_i = -1$) yields a logical *subsumes* match, a text similarity of 0.6, and structural similarity of 0.7.

For the training set $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, we randomly selected 2325 samples in total derived from SAWSDL-TC1 with equal quantities of positive and negative samples. This amounts to around 10% of the complete search space of samples (which size is the number of requests times the number of services in the used collection) over which the binary SVM classifier for service relevance is learned.

The SVM classification problem is defined as the following optimization problem:

$$\text{minimize in } w,b,\xi : \frac{1}{2} w^T w + C\sum_{i=1}^{N} \xi_i \text{ subject to } \forall 1 \leq i \leq N : y_i\left(w^T \phi(x_i) + b\right) \geq 1 - \xi_i, \xi_i \geq 0,$$

where w and b define the optimally separating hyperplane as the set of points satisfying $w^T\phi(x) + b = 0$. Furthermore, w is the *normal vector* which specifies the orientation of the plane, b is called *bias* and indicates the offset of the hyperplane from the origin of the feature space X. The error term $C\sum_{i=1}^{N} \xi_i$ is introduced to allow for outliers in a non-linear separable training set, where the error penalty parameter C must be specified beforehand. The predefined function ϕ maps features into a higher, possibly infinitely dimensional space in which the SVM finds an optimal hyperplane that allows the classification of non-linear separable data (more precise with respect to the original dimension of X)[ix].

Since $w = \sum_{i=1}^{N} y_i \alpha_i \phi(x_i)$ is a linear combination of training sample feature vectors the dual formulation of the SVM classification problem that is actually solved by SAWSDL-MX2 is as follows:

$$\text{maximize in } \alpha : \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j K(x_i, x_j) - \sum_{i=1}^{N} \alpha_i$$

$$\text{subject to } \sum_{i=1}^{N} y_i \alpha_i = 0, \forall 1 \leq i \leq N : 0 \leq \alpha_i \leq C$$

.

The kernel function $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ implicitly defines ϕ in the scalar product. The problem is solved by finding a set of Lagrange multipliers ϕ_i representing the hyperplane for which training samples x_i with $\phi_i \neq 0$ are called support vectors (of the hyperplane). For SAWSDL-MX2, we choose the RBF Kernel (Radial Basis Function) as suggested in (Hsu et al. 2007):

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}.$$

Unlike polynomial kernels, it only introduces a single parameter γ which keeps the complexity of model selection low. Besides, for specific parameter settings it can behave like a linear or sigmoid kernel.

The searching of a SVM parameter setting (C, γ) that is optimal with respect to its average classification accuracy has been done through means of grid search and 6-folded cross-validation. Binary classification of samples $x \in X$ for service pair (S, R) with the above parameters is defined as follows: $d(x) = \sum_{i=1}^{N} y_i \alpha_i K(x_i, x) + b$ with bias b satisfying the Karush-Kuhn-Tucker condition (Chang & Lin 2001), such that S is classified as relevant if and only if $d(x) > 0$. Please note, that w is not a direct output of the dual optimization but computed using the objective value o of the dual optimization and the coefficients α_i based on the relation between the primary and dual problem:

$$\|w\|^2 = w^T w = \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j K(x_i, x_j) = 2 \cdot \left(o + \sum_{i=1}^{N} \alpha_i \right).$$

For more details on SVM in general and on the dual problem solving in particular, we refer the interested reader to, for example, (Hsu et al. 2007).

[Insert figure 6 here]

Figure 6. Illustration of SVM classification for training set of linear separable 2D samples.

Figure 6 illustrates the functioning of a linear SVM classifier (i.e. $K(x_i, x_j) = x_i^T x_j$) in a two-dimensional example for adapting the aggregation of two service matching variants (structural + text) forming a linearly separable 2D feature space $[0,1] \times [0,1]$. The parameters of both problem formulation variants for this case are shown: the normal vector w specifies the orientation of the resulting hyperplane and b the offset from the origin for the primal formulation. The dual parameters α_i characterize the support vectors (labeled SV in the figure), which lie exactly on the planes with maximum margin computed by the SVM optimization problem solver.

Using Learned SVM Classifier with Ranking for Service Selection

Once the matchmaker SAWSDL-MX2 has finished its training phase over the given test collection, it can apply the learned SVM classifier to any new service pair (S, R) with a new request R that is not known in the training set to compute the hybrid semantic matching degree. More concrete, it just applies its learned binary classifier d to the corresponding feature vector x of (S, R) as described above. That is, service S is relevant to R, if and only if $d(x) > 0$, otherwise it is classified as irrelevant. Finally, the matchmaker SAWSDL-MX2 then ranks the service S

according to the distance $dist(x) = d(x)/|w|$ of its feature vector x to the learned hyperplane. Eventually, it returns the computed hybrid semantic matching degree for the pair (S, R) as the tuple ($d(x)$, $dist(x)$).

The off-line learning of SAWSDL-MX2 renders it, in principle, independent from any given test collection, in particular any set of services registered at the matchmaker (it just has to learn over the respectively modified test collection in case of changes) as well as any set of different matching filters that the developer would like to use in combination in future versions of SAWSDL-MX2. In this case it just automatically re-learns off-line how to best aggregate them for the actual service collection at hand. That is particularly in line with the off-line adaptive search engine Striver (Joachims & Radlinski 2007) and the adaptive OWL-S matchmakers OWLS-MX3 (Klusch & Kapahnke 2009) and OWLS- iMatcher2 (Kiefer & Bernstein 2008).

IMPLEMENTATION

We implemented the SAWSDL matchmaker variants described in previous sections in Java. In particular, we used the sawsdl4j[x] API (handling SAWSDL for WSDL 1.1) to parse SAWSDL documents, the OWL API[xi] for accessing OWL ontologies used for annotation, the DIG 1.1 standard interface to handle OWL-DL (SHOIQ) knowledge base queries, and the OWL-API/Pellet[xii] reasoner as OWL-DL inference engine for logic-based semantic matchmaking. The binary SVM-classifier with ranking as used by the SAWSDL-MX2 has been implemented with the public libSVM[xiii] software.

Figure 7 provides an overview of the components of the SAWSDL-MX2 matchmaker architecture. The implementation of both the non-adaptive SAWSDL-MX1 and the adaptive SAWSDL-MX2 is publicly available at the portal semwebcentral.org.

[Insert figure 7 here]

Figure 7. Architecture of the adaptive hybrid semantic matchmaker SAWSDL-MX2.

The semantic service selection tool called SAWSDL-MX offers both matchmakers with an integrated graphical user interface (see Figure 8).

[Insert figure 8 here]

Figure 8. Graphical user interface of the semantic service selection tool SAWSDL-MX.

The functionality of this tool covers all steps of semantic service selection ranging from the selection of given test collection, configuration of the selected matchmaker variant, and the display of not only its answer set for a particular request but also the results of applying different selected retrieval performance evaluation measures which can be conveniently saved in form of an evaluation summary report in PDF.

PERFORMANCE EVALUATION

For evaluating the retrieval performance of the different non-adaptive and adaptive SAWSDL service matchmakers described in previous sections, we conducted a comparative evaluation experiment based on the only publicly available SAWSDL service retrieval test collection

SAWSDL-TC1 and the classical retrieval performance measures macro-averaged recall/precision, average precision and average query response time. For checking the statistical significance of the evaluation results for different matching variants, we used the standard Friedman test.

In the following, we focus on the comparative retrieval performance evaluation of the non-adaptive matchmakers SAWSDL-M0+WA and SAWSDL-MX1, and the adaptive matchmaker SAWSDL-MX2. For more detailed results on SAWSDL-MX1 alone, we refer to (Klusch & Kapahnke 2008).

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[xiv] using the OWLS2WSDL[xv] 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 SAWSDL-TC1 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 are pointing 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 XP 32b, Java 6, 2.8 GHz CPU and 4 GB RAM. We used the publicly available semantic service matchmaker evaluation tool SME2[xvi] developed at DFKI for comparative performance evaluation of semantic service matchmakers.

Retrieval Performance Evaluation Measures

For retrieval performance evaluation, we measured the classical precision and recall:

$$Prec = \frac{|A \cap B|}{|B|}, Rec = \frac{|A \cap B|}{|A|},$$

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 at standard recall levels*:

$$Prec_i = \frac{1}{|Q|} \cdot \sum_{q \in Q} \max\{P_o | R_o \geq Rec_i \wedge (R_o, P_o) \in O_q\},$$

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 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 n/λ, $n = 1, \ldots, \lambda$) 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. $count(r) = \sum_{i=1}^{r} isrel(i)$. Please note that the AP measure is independent from the way and size of ranking.

Statistical Significance Test

In general, differences in 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 (Hull 1993) to be a vital explanatory component of a comparative retrieval performance evaluation.

We are using the Friedman Test variant proposed in (Iman & Davenport 1980) as $F_N = MSR/MSE$, 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 and m the number of tested matching variants. The resulting p-value indicates, if there is a significant difference between the variants which one cannot 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.

Performance Evaluation Experiments

As a first experiment, we compared the retrieval performance of SAWSDLM0+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.

[Insert figure 9 (a) and (b) side by side here]

(a) Logical (SAWSDL-M0) vs. structural (WSDL-Analyzer/WA) vs. hybrid (SAWSDL-M0+WA) semantic service selection.
(b) Non-adaptive hybrid (SAWSDL-M0+WA & SAWSDL-MX1) vs. adaptive hybrid (SAWSDL-MX2) semantic service selection.

Figure 9. Macro-averaged precision/recall of SAWSDL service matchmaking variants.

As shown in Figure 9(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 (Klusch & Kapahnke 2008). 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 SAWSDLMX2 (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 in Figure (b), the adaptive SAWSDL-MX2 performs better than SAWSDL-M0 (logic-based only) and at least as good as the non-adaptive variant SAWSDL-MX1 utilizing logic-based matching and extended Jaccard (Tanimoto) text similarity-based matching. This is mainly due to the fact, that text similarity computation as described introductorily 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 in the future.

	SAWSDL-MX1	**SAWSDL-MX2**	**COM4SWS**	**URBE**	**SAWSDL-iMatcher3**
AP	0.7	0.68	0.68	0.73	0.63
AQRT	3.1s	7.9s	6.14s	20s	.75s

Table 2 Average precision and query response time (in seconds) of SAWSDL-MX matchmaker variants compared to other SAWSDL matchmakers URBE, COM4SWS and SAWSDL-iMatcher3.

Table 2 summarizes the average precision (AP) and query response time (AQRT) of SAWSDL-MX1 and SAWSDL-MX2 as well as other available SAWSDL matchmakers developed elsewhere, that are non-adaptive COM4SWS, URBE and adaptive SAWSDL-iMatcher3; we will comment on these relevant matchmakers in the next section. The results of their performance evaluation over the same test collection SAWSDL-TC1 (and same publicly available evaluation tool SME2[xvii]) are taken from the summary report of the 2009 edition of the international semantic service selection contest (S3 2009).

The average precision values for the individual service matching variants, that are logic-based matching (SAWSDL-M0), structural matching by WSDL-Analyzer (WA) and text matching with TFIDF-Cosine (IR) are 0.53, respectively, 0.44, respectively, 0.46. Their average query response times are 1.7, respectively, 3, respectively 1.4 seconds. The differences between their precision values are not statistically significant. It is noteworthy that the text matching did outperform neither pure logic-based nor structural matching. The latter is partly due to redundant text similarity measurement of unfolded I/O concepts by the WA and IR since the XML schema parameter definitions of SAWSDL services in SAWSDL-TC were automatically derived from

respective I/O concept definitions for corresponding OWL-S services in the collection OWLS-TC2 by the publicly available conversion tool OWLS2WSDL [xviii].

The hybrid semantic SAWSDL matchmaker variants SAWSDL-M0+WA (logic + structural) and SAWSDL-MX1 (logic + text) outperformed the individual filters (SAWSDL-M0, WA, IR) in terms of AP 0.61, respectively, AP 0.68. For example, the statistical significance test for macro-averaged recall/precision results for SAWSDL-M0+WA (logic + structural matching) compared to the structural only matching by the WSDL-Analyzer (WA) and the logic-based only matchmaker variant SAWSDL-M0 yielded $p = 0.026$, respectively, $p = 0.0028$ both of which below threshold α. That is, the hybrid combination significantly outperformed the individual ones at 5% level. This is in perfect line with experimental evaluation results for hybrid semantic service matchmakers reported elsewhere (Klusch et al. 2006, 2009, 2009a).

RELATED WORK

All SAWSDL matchmakers listed in Table 2 did perform equally well, that is with reasonable and statistically insignificant differences between their average precision ($p = 0.331$ at 5% level), but with unacceptably high query response times for many applications. For the given test collection SAWSDL-TC1, no performance gain resulted from using adaptive SAWSDL matchmakers like SAWSDL-MX2 (SVM classifier) or SAWSDL-iMatcher3 (linear regression) instead of non-adaptive ones like SAWSDL-MX1, COM4SWS and URBE manually optimized with respect to the collection.

However, as mentioned above, the automated optimal aggregation of different kinds of selection results through machine learning renders the respectively adaptive matchmakers independent from changes of considered service collections as well as newly added or modified matching filters. In case of non-adaptive matchmakers, this would require a tedious manual (re-) tuning of their individual and aggregated matching processes to (again) keep up with the precision of adaptive ones. The off-line learning time of SAWSDL-MX2 over its training subset of SAWSDL-TC1 was 31 minutes of which 5 minutes were devoted to feature space building for 2325 samples (cf. page 15). In addition, most non-adaptive matchmakers use linearly weighted aggregation functions while SAWSDL-MX2 automatically learns such aggregation even for non-linear separable feature spaces.

It is noteworthy that the only other adaptive SAWSDL service matchmaker SAWSDL-iMatcher3 trained over the full test collection of the contest by their developers using a linear regression model was slightly outperformed by SAWSDL-MX2 which was only trained over a random 10% subset using SVM-based classification. The extremely fast query response time of SAWSDL-iMatcher3 is due to its essentially pre-computed answer sets for this collection.

In any case, we emphasize that the above results of our experimental evaluation strongly depend on the only publicly available SAWSDL service retrieval test collection SAWSDL-TC1. In this regard, the reported results are preliminary until a more comprehensive standard SAWSDL test collection (like the TREC collections in the information retrieval domain) becomes available in the future.

The first search engine for WSDL-S services has been Lumina (Li et al. 2006) developed in the METEOR-S project[xix]. It follows an approach similar to FUSION based on mapping WSDL-S (the predecessor of SAWSDL) to UDDI but performs non-logic-based semantic matching in terms of structural ontology-based (path lengths between terms in shared ontology) and simple keyword-based service matching scores only.

The URBE matchmaker (developed by Pierluigi Plebani in 2008 at the Politecnico di Milano, Italy) performs non-logic-based semantic SAWSDL service matching (Plebani & Pernici 2009).

In particular, it performs text similarity matching of property-class and XSD I/O message data types as well as structural ontology-based I/O concept similarity in terms of worst-case path length between concepts in a shared ontology. There is no logical reasoning involved. Like SAWSDL-MX1 and SAWSDL-MX2, URBE performs bipartite graph-matching of service operations and ranks services based on weighted aggregation of structural and text matching scores. Remarkably, URBE won the special track for SAWSDL service selection of the international S3 (semantic service selection) contest editions 2008 and 2009 in terms of average precision (0.72). Though, the differences to the adaptive SAWSDL matchmakers SAWSDL-MX2 and SAWSDL-iMatcher3 were statistically not significant, and its average query response time was worst (20 sec) compared to that of its next ranked competitors.

The COM4SWS matchmaker (developed by Stefan Schulte and his colleagues at the Technische Universität Darmstadt, Germany) is also a hybrid semantic SAWSDL service matchmaker: It performs non-logic-based clustering (FarthestFirst, syntactic distance) of service descriptions in the vector space model and checks for logic-based mutual coverage of (subclasses of) I/O concepts. Matching of service components from a service offer and a service request is based on the Hungarian algorithm (i.e., bipartite service signature graph matching) while ranking values are computed as weighted similarity values from all service abstraction levels. No further information is available on COM4SWS yet. It is noteworthy, that it performed close to the next ranked adaptive SAWSDL-MX2 in terms of average precision (0.681) but slightly faster in average (6.14 sec) at the 2009 edition of the S3 contest.

The SAWSDL-iMatcher3 (developed by Dengping Wei and Avi Bernstein in 2009 at the University of Zurich, Switzerland) is most relevant to SAWSDL-MX2 with respect to adaptive selection of SAWSDL services. It partially applies the selection process performed by the OWL-S matchmaker iMatcher developed by the same group (Kiefer & Bernstein 2008) to the case of SAWSDL service selection. In particular, SAWSDL-iMatcher3 is hybrid in the sense that it performs logical matching based on output concept subsumption and text similarity-based matching of service names, if they are provided in the respective SAWSDL descriptions. For the latter purpose, it applies different token- and edit-based text similarity measures. Like SAWSDL-MX2 it learns how to best select services off-line but applies a linear regression model with ranking and does perform neither text matching of logically unfolded I/O concept definitions nor WSDL structure matching.

(Syeda-Mahmood et al. 2005) present a hybrid semantic service matchmaking approach for WSDL-S with annotations in OWL based on the combination domain-independent knowledge aquired from the WordNet thesaurus and domain-dependent semantic scoring. In contrast to SAWSDL-MX2, it relies on a fixed scoring schema that is adapted manually to valuate different types of concept relations. Hybrid result aggregation is also fixed by considering the minimum result as overall ranking score, whereas our approach automatically adapts the overall aggregation strategy including different degrees of logical match using SVM training. Apart from the matching procedure itself, a redundant storage strategy named attribute hashing is presented to improve retrieval performance in large service repositories.

The adaptive and hybrid semantic service matchmaker OWLS-MX3 (Klusch & Kapahnke 2009) is similar to SAWSDL-MX2 in that it performs the same SVM-based off-line learning and is restricted to semantic annotations in OWL-DL. However, both matchmakers significantly differ in various aspects of their selection process such as the way of how they perform structural matching. While SAWSDL-MX2 performs structural matching on the WSDL files of SAWSDL service descriptions ignoring the semantic annotations, OWLS-MX3 ignores the grounding of

semantic services in WSDL completely. Another differing aspect is the definition of the logic-based filters. While SAWSDL-MX2 applies bipartite graph matching at operation level to compute the optimal one-to-one parameter assignment for request and service offer, OWLS-MX3 allows multiple assignments for a parameter mapping, thus producing false positives caused by parameters that subsume more than one parameter of its counterpart.

For a comprehensive survey of semantic service matchmakers in general, we refer the interested reader to (Klusch 2008). A survey of approaches to WSDL service signature or behavior (message-based conversation) matching such as WXplorer (Stroulia & Wang 2004) or Woogle (Dong et al. 2004) is out of the scope of this paper but provided, for example, in (Plebani & Pernici 2009). Taking a broader view, the task of semantic service retrieval can be interpreted as finding compositions of services that fulfill given requirement specifications, as for example in SEMAPLAN (Akkiraju et al. 2006), which applies semantic service matching in a cost function to guide an AI planner in generating favorable service compositions, or Opossum (Toch et al. 2007), which introduces the notion of service networks to represent service dependencies as basis for approximated retrieval of service composition. However, integration of SAWSDL-MX2 in such a system is out of the scope of this paper.

CONCLUSION

We discussed different hybrid SAWSDL service matchmakers each of which outperform the individual types of semantic service signature matching that they combine. Among others, the comparative experimental performance evaluation showed that the combined use of logical and non-logic-based structural matching may indeed outperform logic-based only matching but not the combination of logical with text similarity-based matching by SAWSDL-MX1.

Further, the adaptive combination of all three types of matching by SAWSDL-MX2 performed as well as the non-adaptive hybrid variant in terms of average precision. This experimental result appears somewhat disappointing at first glance but, apart from that it depended on the only available test collection SAWSDL-TC1, the major benefit of the offline learning capability is that it renders SAWSDL-MX2 independent from any manual adjustment to other or updated test collections and matching filters in the future. In fact, if services or ontologies change or new filters shall be used or integrated into a given non-adaptive matchmaker, the optimal configuration of its filters with respect to precision would have to be sought by the developer in time-consuming experiments and analysis otherwise.

REFERENCES

Akkiraju, R., Srivastava, B., Ivan, A.-A., Goodwin, R. & Syeda-Mahmood, T. (2006). *SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition*. Proceedings of 6th International Conference on Web Services (ICWS), IEEE Computer Society, Washington, DC, USA.

Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D. & Patel-Schneider P.F. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Chang, C.-C. & Lin, C.-J. (2001). *Libsvm: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Dong, X., Madhavan, J. & Halevy, A. (2004). *Mining Structures for Semantics*. ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations Newsletter, 6(2).

Hsu C.-W., Chang C.-C. & Lin C.-J. (2007). *A Practical Guide to Support Vector Classification*.
Hull, D. (1993). *Using statistical testing in the evaluation of retrieval experiments.* Proceedings of 16th ACM SIGIR conference on research and development in information retrieval.
Iman, R.L. & Davenport, J.M. (1980). *Approximations of the critical region of the Friedman statistic*. Communications in Statistics, A9(6).
Joachims, T. & Radlinski, F. (2007). *Search Engines that Learn from Implicit Feedback*. IEEE Computer, 40(8).
Kaufer, F. & Klusch, M. (2006). *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.
Kiefer, C. & Bernstein, A. (2008). *The Creation and Evaluation of iSPARQL Strategies for Matchmaking*. Proceedings of 5th European Semantic Web Conference (ESWC), LNCS, 5021, Springer.
Klusch, M. (2008): *Semantic Service Coordination*. In: (Schumacher et al. 2008), Chapter 4.
Klusch, M., Fries, B. & Sycara, K. (2006). *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.
Klusch, M. & Kapahnke, P. (2008). *Semantic Web Service Selection with SAWSDL-MX*. CEUR Proceedings of 2nd Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2), Karlsruhe, Germany, CEUR 416.
Klusch, M. & Kapahnke, P. (2009): *OWLS-MX3: An Adaptive Hybrid Semantic Service Matchmaker for OWL-S*. Proceedings of 3rd International Workshop on Semantic Matchmaking and Resource Retrieval (SMR2) at ISWC, Washington, USA; CEUR 525
Klusch, M., Kapahnke, P. & Zinnikus, I. (2009): *SAWSDL-MX2: A Machine-Learning Approach for Integrating Semantic Web Service Matchmaking Variants*. Proceedings of IEEE 7th International Conference on Web Services (ICWS), Los Angeles, USA, IEEE CS Press.
Klusch, M., Fries, B. & Sycara, K. (2009a). *OWLS-MX: A Hybrid Semantic Web Service Matchmaker for OWL-S Services*. Web Semantics, 7(2), Elsevier.
Kourtesis, D. & Paraskakis, I. (2008). *Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery*. Proceedings of 5th European Semantic Web Conference (ESWC), LNCS, 5021, Springer.
Li, K., Verma, K., Mulye, R., Rabbani, R., Miller, J. A. & Sheth, A. P. (2006). *Designing Semantic Web Processes: The WSDL-S Approach*. Semantic Web Services, Processes and Applications. J. Cardoso, A. Sheth (eds.), Springer.
Plebani, P. & Pernici, B. (2009). *URBE: Web Service Retrieval Based on Similarity Evaluation*. IEEE Transactions on Knowledge and Data Engineering, 21(11).
S3 - International Semantic Service Selection Contest. Website: http://www.dfki.de/~klusch/s3.
Schumacher, M., Helin, H. & Schuldt, H. (2008) Eds. *CASCOM – Intelligent Service Coordination in the Semantic Web*. Birkhäuser Verlag, Springer.
Stroulia, E. & Wang, Y. (2004). *Structural and Semantic Matching for Assessing Web-Service Similarity*. Cooperative Information Systems, 14(4), World Scientific.
Toch, E., Gal, A., Reinhartz-Berger, I. & Dori, D. (2007). *A semantic approach to approximate service retrieval*. ACM Transactions on Internet Technology 8(2), ACM.
Zinnikus, I., Rupp H.-J. & Fischer, K. (2006). *Detecting Similarities between Web Service Interfaces: The WSDL Analyzer*. Proceedings of 2nd International Workshop on Web Services and Interoperability (WSI 2006).

[i] http://www.w3.org/TR/sawsdl/

[ii] http://www.w3.org/TR/wsdl/ and http://www.w3.org/TR/wsdl20/

[iii] The W3C recommendation for SAWSDL does not define a logical relationship among multiple model references for a single element.

[iv] http://www.w3.org/2004/OWL/

[v] http://dig.sourceforge.net/

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

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

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

[ix] The fraction ½ is introduced for computational reasons only, and does not affect the classification result.

[x] http://knoesis.wright.edu/opensource/sawsdl4j/

[xi] http://owlapi.sourceforge.net/

[xii] http://pellet.owldl.com/

[xiii] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[xiv] http://projects.semwebcentral.org/projects/owls-tc/

[xv] http://projects.semwebcentral.org/projects/owls2wsdl/

[xvi] http://projects.semwebcentral.org/projects/sme2/

[xvii] SME2: Semantic Service Matchmaker Evaluation Tool. Available at projects.semwebcentral.org/projects/sme2/

[xviii] OWLS2WSDL converter. Available at projects.semwebcentral.org/projects/owls2wsdl/

[xix] http://lsdis.cs.uga.edu/projects/meteor-s/