SAWSDL-MX2: A Machine-learning Approach for Integrating Semantic Web Service Matchmaking Variants

Matthias Klusch, Patrick Kapahnke and Ingo Zinnikus
German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, Saarbrücken, Germany
E-mail: klusch|patrick.kapahnke|ingo.zinnikus@dfki.de

Abstract

In this paper, we present SAWSDL-MX2, a hybrid semantic Web service matchmaker for SAWSDL services. Building on our initial work in [11], we adopt logic-based as well as text similarity service selection for model references and add a structural approach from [14], which operates on the pure syntactic description of WSDL elements. The integration of these matching variants is accomplished using a Support Vector Machine (SVM) with non-linear kernel, thus automatically adapting an aggregation function based on previously experienced training data. Results of our performance evaluation based on the standard measures recall and precision over the SAWSDL-TC1 test collection as well as an exhaustive example for all basic matching variants are also given.

1. Introduction

As a W3C recommendation dated August 28, 2007, the SAWSDL specification proposes mechanisms to enrich Web services described in WSDL (Web Service Description Language) with semantic annotations. Building on our idea of hybrid semantic service selection presented in [10] and [4], we initially released SAWSDL-MX1 as presented in [11]. This initial version of the matchmaker performs hybrid semantic matching utilizing a logic-based and a standard information retrieval text similarity approach and aggregates the results in a strictly predefined conjunctive way based on the examination of real-world data and the characteristical problems arising from it for semantic service matching. Furthermore, the focus was mainly on the semantic annotations of parameters introduced by the SAWSDL specification and only basic structures of a service interface where considered in the matching process. To remedy these shortcomings, our second version SAWSDL-MX2 presented in this paper additionally makes use of the structural comparison of WSDL descriptions performed by the WSDL Analyzer as presented in [14], a tool to support semi-automated Web service integration. Instead of a fixed integration of all variants, a Support vector Machine (SVM) is used to train a non-linear aggregation function based on training data derived from binary relevance assessments of our test collection SAWSDL-TC1. This approach outperforms all basic variants in terms of recall and precision.

The remainder of this paper is structured as follows. After introducing the matching variants utilized by SAWSDL-MX2 in section 2, an working example is given in section 3. The adaptive integration of these variants using a SVM is described in section 4 followed by the results of our experimental evaluation in section 5. We comment on related work in section 6 and conclude in section 7.

2. Basic Matching Approaches

In the following, we describe the three basic service selection approaches used by SAWSDL-MX2, namely crisp logic-based, text similarity and WSDL structure matching. As service requests, standard SAWSDL Web service description documents are used. This idea is particularly inspired by OWLS-MX [10] and WSMO-MX [4] for OWL-S and WSML. Service offers registered at the matchmaker as well as service requests contain an interface (port type in WSDL 1.1), potentially containing multiple operations defined by an arbitrary number of parameters. Additional definitions concerning the binding to concrete implementations (e.g. via SOAP) are only considered by the WSDL structure comparison provided by WSDL Analyzer.

To combine partial results of logic-based or text similarity matching computed for pairs of service request and service offer operations to a single value, a bipartite graph matching approach is applied on the interface level. A conservative min-max aggregation variant is used to compute

---

1 http://www.w3.org/TR/sawSDL/
2 http://www.w3.org/TR/wsdl/
the overall result. That is, given the global optimal assignment of request operations to service offer operations, the worst result is chosen. This guarantees a fixed lower bound of similarity for every requested operation. If there exists no exhaustive assignment, then the matching process fails, as some requested operations are not offered by the service in question. Please note, that WSDL Analyzer compares the structure of a service as a whole and does not apply this technique.

2.1. Logic-based Operation Matching

The logic-based operation matching part of SAWSDL-MX2 computes the degree of logical match for a given pair of service offer and request operations, $O_O$ and $O_R$, respectively, by successively applying four filters of increasing degree of relaxation: exact, plug-in, subsumes and subsumed-by, which are, in essence, adapted from those of OWLS-MX2 but modified in terms of an additional bipartite concept matching to ensure an injective mapping between offer and request concepts. These mappings are denoted as $M_{in}$ and $M_{out}$ in the following. The reason of this modification is that previous experiments with OWLS-MX showed that many logic-based only failures could have been avoided by this additional constraint. As an overview to description logics and DL reasoning, we refer to [1].

**Exact match:** $O_O$ exactly matches $O_R \iff (\exists m \in M_{in} : \forall m \in M_{in} : m_1 \in \text{in}(O_O) \wedge m_2 \in \text{in}(O_R) \wedge m_1 \equiv m_2) \land (\exists m \in M_{out} : \forall m \in M_{out} : m_1 \in \text{out}(O_R) \land m_2 \equiv m_1 \wedge m_2 \in \text{out}(O_O) \wedge m_1 \equiv m_2)$. There exists a mapping of perfectly matching inputs as well as perfectly matching outputs.

**Plug-in match:** $O_O$ plugs into $O_R \iff (\exists m \in M_{in} : \forall m \in M_{in} : m_1 \in \text{in}(O_O) \wedge m_2 \in \text{in}(O_R) \wedge m_1 \equiv m_2) \land (\exists m \in M_{out} : \forall m \in M_{out} : m_1 \in \text{out}(O_R) \wedge m_2 \equiv m_1 \wedge m_2 \in \text{out}(O_O) \wedge m_1 \equiv m_2)$. The plug-in 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:** $O_O$ subsumes $O_R \iff (\exists m \in M_{in} : \forall m \in M_{in} : m_1 \in \text{in}(O_O) \wedge m_2 \in \text{in}(O_R) \wedge m_1 \equiv m_2) \land (\exists m \in M_{out} : \forall m \in M_{out} : m_1 \in \text{out}(O_R) \wedge m_2 \equiv m_1 \wedge m_2 \in \text{out}(O_O) \wedge m_1 \equiv m_2)$. This filter further relaxes constraints by allowing service offer outputs to be arbitrarily more specific than the request outputs.

**Subsumed-by match:** $O_O$ is subsumed by $O_R \iff (\exists m \in M_{in} : \forall m \in M_{in} : m_1 \in \text{in}(O_O) \wedge m_2 \in \text{in}(O_R) \wedge m_1 \equiv m_2) \land (\exists m \in M_{out} : \forall m \in M_{out} : m_1 \in \text{out}(O_R) \wedge m_2 \equiv m_1 \wedge m_2 \in \text{out}(O_O) \wedge m_2 \equiv \text{lsc}(m_1))$. 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 referred concept classification.

The overall algorithm for logic-based matching of operations considers the filters in the following order based on the degree of relaxation: exact $>$ plug-in $>$ subsumes $>$ subsumed-by $>$ fail. The notion of fail applies to cases where none of the filtering tests succeeded.

2.2. Text Similarity Operation Matching

In addition, SAWSDL-MX can perform syntactic matching based on selected token-based text similarity measures. That is, a syntactic similarity value is computed for every pair of service offer and request operation. The implemented similarity measures for SAWSDL-MX are the Loss-of-Information, the Extended Jaccard, the Cosine and the Jensen-Shannon similarity measures.

The motivation for applying these measures, originally intended for natural language document comparison, is the observation, that the concept and role names used to self-contained 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 SAWSDL-TC1 (this also holds e.g. for OWLS-TC), the observed distribution of terms is as follows:

$$f(k, s, N) = \frac{1}{k^s} \sum_{n=1}^{N} \frac{1}{n^s},$$

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 $\frac{1}{k}$, 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. Figure 1 scatterplots (logarithmically scaled) one of the term indices, whose creation is described in the following, and the assumed distribution to support the proposition that the data generated for text similarity matching as proposed in this paper resembles natural language text.
The weighted keyword vectors of inputs and outputs for every operation are generated by first unfolding the referenced concepts in the ontologies (as defined for standard tableaux reasoning algorithms). The resulting set of primitive concepts of all input concepts of a service operation is then processed to a weighted keyword vector based on TFIDF weighting scheme, the same is done with its output concepts. To compute the TFIDF values, two distinct indices are used depending on whether inputs or outputs are compared. For this reason, the IR-based matching used in SAWSDL-MX can be considered as structured. To produce a single result similarity, the I/O text similarities of a service offer operation and a request operation are averaged.

2.3. Structural WSDL Matching

For a pure syntactic/structural approach to service matchmaking, we use the WSDL Analyzer as introduced in [14]. It ignores the semantic annotations of SAWSDL descriptions and treats a SAWSDL description as WSDL file. The WSDL Analyzer is a tool for detecting similarities and differences between WSDL files. Since the similarity algorithm produces a mapping between WSDL files, the tool can also be used for supporting mediation between services.

The underlying approach exploits various types of schema information (e.g. element names, datatypes and structural properties), characteristics of data instances, as well as background knowledge from dictionaries and thesauri. The algorithm respects the structural information of complex datatypes and is flexible enough to allow relaxed matching and matching between parameters that come in different orders in parameter lists.

The comparison of two WSDL files is a multi-step process: it involves the comparison of the operation sets offered by the services, which is based on the comparison of the structures of the operations input and output messages, which, in turn, is based on the comparison of the datatypes of the objects communicated by these messages.

A WSDL description can be represented as labelled tree where leaf nodes are the basic built-in datatypes provided by the XML schema specification3. 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 \( N = \{n_1, n_2, ..., n_n\} \) a set of nodes, \( E \subseteq N \times N \) a set of edges, \( root(T) \) the root of the tree and \( \varphi : N \rightarrow L \) a function which assigns a label to each node with basic datatypes \( D \subseteq L \). 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 the trees recursively.

For \( a \in N_{T_1} \) and \( b \in N_{T_2} \),

\[
\text{sim}(a, b) = \begin{cases} 
\omega_n \cdot \text{sim}_n(\varphi(a), \varphi(b)) + \omega_s \cdot \max \{\oplus_{i,j}\}, & \varphi(a), \varphi(b) \notin D \\
\text{sim}_E(\varphi(a), \varphi(b)), & \varphi(a), \varphi(b) \in D
\end{cases}
\]

Here, \((a, n_i) \) and \((b, m_j) \) \( E \) and \( D \) \( \oplus_{i,j} \) is the sum of pairs \( \text{sim}_i(n_i, m_j) \) for \( 1 \leq i \leq \text{card}(n) \) and \( 1 \leq j \leq \text{card}(m) \) such that each \( n_i \) and \( m_j \) occur at most once in the sum. If \( \text{card}(n) \neq \text{card}(m) \), some of the nodes cannot be matched. Weights \( \omega_n \) and \( \omega_s \) are used to increase or decrease the effect of name or structural similarity. Similarity of types \( \text{sim}_i \) is based on a compatibility table which assigns a value to each combination of basic datatypes. The similarity of labels \( \text{sim}_n \) can be calculated with different measures such as string edit distance, substring containment or WordNet 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.

3. Service Matching Example

In the following, an exhaustive example demonstrating all matching algorithms presented in this paper is given. The scenario is as follows: a user orders his service selection agent to find a Web service, which offers a map given a location as input. To do this, a valid service request document is generated (this could be done by the user directly or by the agent, however, it is out of scope for this paper). It is structured as follows:

```xml
<types>
  <complexType name="GPSType" modelReference="http://...#GPSPos"/>
  <complexType name="MapType" modelReference="http://...#Map"/>
  <complexType name="Location" type="GPSType"/>
</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>
```

3http://www.w3c.org/TR/xmlschema-2/

4http://wordnet.princeton.edu/
Please note, that not every detail of the service description is provided here for reasons of space-saving. Basically, a service is requested, which offers an operation returning the reference to an appropriate map given a GPS position as input. The data types used to describe the operation syntactically are annotated using SAWSDL model references. The input is claimed to conform to the concept GPSPosition while the output is a Map, both described in OWL.

Among others, there is a service offer registered at SAWSDL-MX2, which provides similar functionality and which is described as follows:

```
Listing 1. Service Request
```

```
<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.

For the logic-based operation matching, the following concept definitions are assumed (considering the more compact standard DL notation instead of OWL-DL):

```
   Map ≡ Image ∩ ∃hasScale.Scale
       ∩ ∃contains.GeographicalEntity
   RoadMap ≡ Map ∩ ∃contains.Road
   Location ≡ GeographicalEntity
```

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 ⊆ Map holds for the outputs, there does not exists 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 subsumption reasoning does not discover it.

One possibility to catch such cases is the usage of text similarity approaches as described in section 2.2. There, the unfolded concept definitions are used as input for a text similarity measure. Since the vector space model utilizing TFIDF weights relies on a whole document corpus, we use the Loss-of-Information model for our example, which just takes the two documents to compare into account. It is defined as follows: $LOI(R, S) = 1 - \frac{|R \backslash S| + |S \backslash R|}{|R| + |S|}$, 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 the inputs of the example, which are the problematic case for the standard logic-based matching, the set of terms for the request after unfolding and stopword elimination is $R_{in} = \{\text{hasLongitude}, \text{FPNum}, \text{hasLatitude}\}$. For the service offer, the set is $S_{in} = \{\text{Location}, \text{GeographicalEntity}, \text{hasParameter}, \text{hasLongitude}, \text{FPNum}, \text{hasLatitude}\}$. The partial result for the inputs then is $LOI(R_{in}, S_{in}) = \frac{3}{8}$, which indicates a moderate textual similarity (in contrast to logic-based fail). For the outputs, the similarity value is $LOI(R_{out}, S_{out}) = \frac{13}{16}$, which indicates high similarity (only the single term Road distinguishes both sets). The overall structured textual similarity then is the average of both values, i.e. approx. 0.75, which is clearly in the upper part of the similarity function range and thus compensates the logic-based false negative.

To illustrate the functionality of the structural WSDL matching described in 2.3, the similarity computation of the two operations in question is exemplified here. Please note, that the actual similarity computation of the WSDL Analyzer involves the whole document structure, but this is an analogous process omitted here for reasons of simplicity. Basically, the algorithm recursively enters the lower level WSDL tree representation nodes of both service descriptions until a leaf node containing a basic datatype occurs. If this is also the case for the other service at the same time, the datatype similarity is looked up in a predefined table. If a basic datatype 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. For our example, 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 this branches add a high similarity to the overall sum. Table 1 summarizes all similarity computations of the WSDL Analyzer assuming that the weighting parameters $\omega_n$ and $\omega_s$ both are equal to 1. For the name similarity, simple string similarity obtained in combination with WordNet matching is used. 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 $\text{sim}_n$ value. The lookup table for datatype matching ($\text{sim}_i$) assigns similarity values in [0, 1], where equal types result in exactly 1. As can be seen from the bottom line of the table, the overall result for the example request and service offer pair is 20.3, which is normalized by the maximum obtainable value 28 to approx. 0.73.

From the given example, it can be easily seen, that each of the proposed approaches has it’s advantages and shortcomings. While logic-based matching takes advantage of formal semantic definitions of concepts and roles, it may be misleading in some cases as for the concepts GPSPos and Location. The additional use of IR techniques and structural comparison may overcome this problem sometimes (and vice versa, e.g. text similarity neglects logical operators like and). For the given problem, text similarity was about 0.75 and WSDL Analyzer resulted in 0.73, which supports the idea of a hybrid combination of all variants.

### 4. SVM Result Aggregation

Inspired by the work done in [6] and [8], we decided to implement a supervised machine learning approach, that trains a classifier based on the basic matching results produced as described in the previous section, namely a Support Vector Machine (SVM). A SVM aims at finding an hyperplane, which separates training examples of two different classes expressed in terms of feature vectors $x_i$ in an n-dimensional feature space $X$, such that all members of one class are located above and all members of the other class are located below this plane. To counteract overfitting, this disjunctive plane is chosen such that it maximizes the distance to the nearest elements of both classes. To classify an unknown element also expressed in terms of its features, it’s position in $X$ is checked against the trained separation.

The application of the SVM algorithm to the service matching problem is as follows: A training example $x_i$ ($i \leq N$, $N$ is the size of the training set) is a vector of features regarding a pair of service request $R$ and service offer $S$. The features are defined as the basic matching result values of this pair, i.e. the logic-based, text similarity and structure matching. Since the result of logic-based filtering is not real-valued but categorical, we assume a separate binary feature for each possible outcome (Exact, Plug-in, Subsumes, Subsumed-by, Fail), thus the feature space $X$ is defined in seven dimensions as $X = \{0, 1\}^7 \times [0, 1] \times [0, 1]$. Each training example $x_i$ also has it’s known class membership $y_i \in \{-1, 1\}$, where a value of $-1$ stands for a negative (i.e. irrelevant) and 1 for a positive (i.e. relevant) example. Thus, the training set is a set of tuples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$. Relevance sets are subjectively defined by domain experts as part of the test collection.

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

$$
\text{minimize } w, b, \xi; \quad \frac{1}{2} w^T w + C \cdot \sum_{i=1}^{N} \xi_i
$$
subject to $\forall i \leq i \leq N : y_i (w^T \phi(x_i) + b) \geq 1 - \zeta_i, \zeta_i \geq 0$, where $w$ and $b$ define the optimal hyperplane according to the previously mentioned characteristics as the set of points fulfilling the equation $w^T \phi(x) + b = 0$. $w$ 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. The error term $C \cdot \sum_{i=1}^{N} \zeta_i$ is introduced to allow for outliers in a non-linear separable training set, where the error penalty parameter $C$ must be specified beforehand. $\phi$ is a predefined function which maps features into a higher and maybe infinite dimensional space. Thus, a SVM finds a hyperplane in this higher dimensional space, allowing a more precise classification of non-linear separable data (with respect to the original dimension of $X$).

Since $w$ can be described as a linear combination of training example feature vectors $w = \sum_{i=1}^{N} y_i \alpha_i \phi(x_i)$, there exists a dual formulation of the SVM classification problem:

$$\begin{align*}
\text{minimize} & \quad \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j K(x_i, x_j) - \sum_{i=1}^{N} \alpha_i \\
\text{subject to} & \quad \sum_{i=1}^{N} y_i \alpha_i = 0, \forall 1 \leq i \leq N : 0 \leq \alpha_i \leq C,
\end{align*}$$

which is usually solved instead. It allows to introduce a kernel function $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$, which implicitly defines $\phi$ in the scalar product. The dual problem is solved by finding a set of Lagrange multipliers $\alpha_i$ (also called Kuhn-Tucker coefficients), which represent the hyperplane together with the training examples. The $\alpha_i$ are called dual parameters in context of SVM, while $w$ is called direct parameter. Training examples $x_i$ for which $\alpha_i \neq 0$ are called Support Vectors. For $K$, the prominent Radial Basis Function (RBF) $K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$ was chosen.

Using the trained parameters, a classification function $d$ can be formulated as follows: $d(x) = \sum_{i=1}^{N} y_i \alpha_i K(x_i, x) + b$. The bias $b$ not explicitly computed during dual problem solving, can be derived using the Karush-Kuhn-Tucker condition (KKT). Details on the computation of $b$ can be found in [2]. A new object $x$ can be classified as relevant or irrelevant by calculating its similarity value. It is obtained using $dist(x) = d(x)$. Please note, that $w$ is not a direct output of the dual optimization. It is computed using the objective value of the dual optimization and the coefficients $\alpha_i$ using the relationship of primal and dual problem formulations $||w||^2 = w^T w = \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j K(x_i, x_j) = 2 \cdot (\alpha + \sum_{i=1}^{N} \alpha_i)$.

The training set used as input for the SVM is obtained from the SAWSDL-TC1 test collection, which already contains relevance assessments made by human judges. Since the evaluation is also performed using this test collection, we decided to train the aggregation function on a small subset drawn at random. Our training set consisted of 5% of all available judgements, which are about 600 samples. To find a good parameter setting $(C, \gamma)$, the n-fold cross validation and grid-search approach proposed in [5] has been conducted. As SVM implementation, we used libSVM$^5$.

## 5. Evaluation of Performance

For SAWSDL-MX2, a retrieval performance evaluation based on the well-known measures recall and precision has been conducted. It is based on the SAWSDL-TC1 test collection, which was semi-automatically derived from OWLS-TC 2.2$^6$ using the OWLS2WSDL$^7$ tool, as there is currently no standard test collection for SAWSDL matchmaking available. OWLS2WSDL transforms OWL-S service descriptions (and concept definitions relevant for parameter description) to WSDL through syntactic transformation. The collection consists of around 900 Web services covering different application domains. For this service offers, a set of queries has been selected and relevance sets have been subjectively specified by domain experts. As the creation of this test collection has been done by transforming OWL-S services contained in OWLS-TC 2.2, which provides services containing only one atomic process per description, every SAWSDL advertisement only contains a single interface with a single operation (but possibly multiple I/O’s). The performance tests have been conducted on a machine with Windows 2000, Java 6, 1.7 GHz CPU and 2 GB RAM using SME$^2$ as evaluation environment.

The performance measures used for our experimental evaluation are defined as follows: $Rec = \frac{|A \cap B|}{|A|}$, $Prec = \frac{|A \cap B|}{|B|}$, where $A$ is the set of all relevant documents for a request and $B$ the set of all retrieved documents for a request. We adopt the prominent macro-averaging of precision. That is, we compute 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. The macro-averaged precision is defined as follows: $Prec_i = \frac{1}{\lambda} \sum_{q \in Q} \max \{ P_o R_o \geq Rec_i \land (R_o, P_o) \in O_q \}$, where $O_q$ denotes the set of observed pairs of recall/precision values for query $q$ when scanning the ranked ser-

---

$^5$http://www.csie.ntu.edu.tw/~cjlin/libsvm/

$^6$http://projects.semwebcentral.org/projects/owls-tc/

$^7$http://projects.semwebcentral.org/projects/owls2wsl/

$^8$http://projects.semwebcentral.org/projects/sme2/
vices in the answer set for \( q \) stepwise for true positives. The well-known Average Precision measure, which produces a single-valued rating of a matchmaking for a single query result has also been used for evaluation. It is defined as follows:

\[
AP = \frac{1}{|R|} \sum_{r=1}^{L} \text{isrel}(r) \frac{\text{count}(r)}{r}
\]

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, \( \text{isrel}(r) = 1 \) if the item at rank \( r \) is relevant and 0 otherwise and \( \text{count}(r) \) the number of relevant items found in the ranking when scanning top-down, i.e. \( \text{count}(r) = \sum_{i=1}^{r} \text{isrel}(i) \). The Average Precision measure enables performance evaluation invulnerable to varying sizes of returned rankings.

5.1. Performance Tests

At first, we compared the retrieval performance of the learned aggregation of SAWSDL-MX2 compared to its basic approaches. As can be seen in figure 2, SAWSDL-MX2 clearly outperforms all other approaches, especially considering the top part of the ranked results (low recall area). This is in line with our experiments conducted in [11], where the hybrid matching variant of SAWSDL-MX1 outperformed the logic-based as well as the IR-based matching. As we already pointed out there, ontologies currently found on the Web are mostly simple taxonomies and rarely use advanced features of e.g. description logics, which damps the benefit of a logic-based approach. Thus, additional consideration of other service features improves the result rankings most of the time. Additional Friedman test series confirmed the fact, that the results were significantly better at 5% level (all \( p \) values below 0.0027).

To see how the aggregation learned from the SVM performs compared to the original fixed hybrid filter definitions of SAWSDL-MX1, we also added it for a second experiment. The results also plotted in figure 2 show, that it is only approximately as good as SAWSDL-MX1 utilizing extended Jaccard text similarity in the hybrid filters. This is mainly due to the fact, that text similarity computation is based on terms derived from unfolded concept definitions as described in section 2.2, which is closely related to structural features in a taxonomy. 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, WSDL Analyzer indirectly performs some structural concept comparison too, which makes it partly redundant and thus does not improve the overall result. So for this relatively special case of SAWSDL-TC1, there is unfortunately no positive impact on the result. Nevertheless, for the general case, the SVM approach enables the integration of arbitrary matching mechanisms in a simple and well-defined way to comprise many aspects of service descriptions to improve result rankings. Also, similar experiments with OWLS-MX and a structural similarity function based on the distance over a direct common subsumer and depth of concepts in an implied is-a hierarchy gave better results, where the usage of a SVM improves overall precision. The adaptive aggregation introduced for OWLS-MX3 and an extensive evaluation of it is still work in progress.

Table 2 summarizes the averaged AP values for all evaluated matching variants. These can be considered for comparing different approaches using single values instead of recall-precision curves. The average query response times are also shown.

### Table 2. Averaged AP values and query response times

<table>
<thead>
<tr>
<th></th>
<th>LB</th>
<th>textIR</th>
<th>WA</th>
<th>MX1</th>
<th>MX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg. AP</td>
<td>0.53</td>
<td>0.46</td>
<td>0.44</td>
<td>0.66</td>
<td>0.65</td>
</tr>
<tr>
<td>avg. QRT</td>
<td>6.5s</td>
<td>1.4s</td>
<td>8.0s</td>
<td>8.2s</td>
<td>15.9s</td>
</tr>
</tbody>
</table>

6. Related Work

To the best of our knowledge, there exist only very few implemented semantic service discovery systems for SAWSDL. [12] presents a solution to SAWSDL Web service discovery using UDDI registries called FUSION. In FUSION, any 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-MX2, each service offer has only to satisfy one matching condition based on subsumption relationships in-
ferred by a reasoner, thus the ranking is not affected by different degrees of logic-based match, neither does FUSION perform a syntactic or hybrid semantic match, as SAWSDL-MX2 and WSDL Analyzer do. Like SAWSDL-MX2, FUSION is strictly bound to OWL-DL, since for each service, a semantic representation in terms of an individual of a predefinedOWL concept is constructed. Lumina [9] developed in the METEOR-S project9 follows a similar approach based on a mapping of WSDL-S (and later on SAWSDL respectively) to UDDI but performs syntactic service matching only. Another SAWSDL matchmaker recently appeared as participant of the S310 (Semantic Service Selection) contest, namely the URBE matcher by Plebani et al., which performs non-logic-based matching in terms of text similarity and structural comparisons. However, there is no additional information available on the used algorithms yet. For a survey of semantic service matchmakers in general, we refer the interested reader to [13].

Apart from SAWSDL service discovery, SVM’s have been used in the context of information retrieval as described in [3]. Cohen et. al combined different text similarity comparison variants using a SVM for traditional document retrieval. As the presented evaluation showed, the results have been improved with respect to precision. In [7], Striver, an integrated Web search engine combining various prominent single approaches such as Google or MSN Search is presented. Additionally, many basic features extracted directly from the search results have been used as training features. In the context of semantic Web service discovery, [8] introduced the iMatcher, which integrates various text similarity measures, applied to OWL-S service descriptions, using different machine-learning algorithms. The performance evaluation also showed that the aggregation outperforms single text matching variants.

7. Conclusion

SAWSDL-MX2 performs hybrid semantic Web service matching for SAWSDL operations based on both logic-based reasoning and IR-based syntactic similarity measurement, and combines the results to provide a matching result for service interfaces with multiple operations. WSDL Analyzer additionally performs structural comparison originally intended to support Web service integration. For aggregating the results obtained from application of these basic variants, a machine-learning strategy using a SVM was chosen. Performance evaluation has been conducted to show that the combination of the presented basic variants improves service selection.

SAWSDL-MX1 and SAWSDL-TC1 are both publicly available at semwebcentral.org.

References