SAWSDL-iMatcher: A Customizable and Effective Semantic Web Service Matchmaker

Dengping Wei\textsuperscript{a,b}, Ting Wang\textsuperscript{a}, Ji Wang\textsuperscript{c}, Abraham Bernstein\textsuperscript{b}

\textsuperscript{a} National University of Defense Technology, 410073, Changsha, China
\textsuperscript{b} University of Zürich, Binzmühlestrasse 14, CH-8050, Zürich, Switzerland
\textsuperscript{c} National Laboratory for Parallel and Distributed Processing, 410073, Changsha, China

Abstract

As the number of publicly available services grows, discovering proper services becomes an important issue and has attracted amount of attempts. This paper presents a new customizable and effective matchmaker, called SAWSDL-iMatcher. It supports a matchmaking mechanism, named iXQuery, which extends XQuery with various similarity joins for SAWSDL service discovery. Using SAWSDL-iMatcher, users can flexibly customize their preferred matching strategies according to different application requirements. SAWSDL-iMatcher currently supports several matching strategies, including syntactic and semantic matching strategies as well as several statistical-model-based matching strategies which can effectively aggregate similarity values from matching on various types of service description information such as service name, description text, and semantic annotation. Besides, we propose a semantic matching strategy to measure the similarity among SAWSDL semantic annotations. These matching strategies have been evaluated in SAWSDL-iMatcher on SAWSDL-TC2 and Jena Geography Dataset (JGD). The evaluation shows that different matching strategies are suitable for different tasks and contexts, which implies the necessity of a customizable matchmaker. In addition, it also provides evidence for the claim that the effectiveness of SAWSDL service matching can be significantly improved by statistical-model-based matching strategies. Our matchmaker is competitive with other matchmakers on benchmark tests at S3 contest 2009.

Key words: Semantic Web Service discovery, SAWSDL, matching strategy

1 Introduction

Semantic Web Service (SWS) discovery is the process of locating Web services based on their comprehensive functional and non-functional semantic
representations [1]. As a critical challenge in Semantic Web Service technique, SWS discovery has attracted a significant amount of attention in recent years [1]. Several SWS ontologies have been proposed since the Semantic Web was proposed by Tim Berners-Lee et al. in [2], such as OWL-S [3], WSMO [4], etc. Besides, several semantic-enabled specifications for Web service have also been proposed on top of the industrial standard WSDL, e.g., WSDL-S [5], SAWSDL [6], etc. These various representations lead to appearances of different SWS matchmakers, such as OWLS-iMatcher [7], OWLS-MX [8], WSMO-MX [9], etc.

SAWSDL is a simple extension of WSDL by using three extension attributes: modelReference, liftingSchemaMapping and loweringSchemaMapping, which are used to annotate existing Web services described in WSDL with semantics in an intuitive and low-cost way. SAWSDL has become a W3C recommendation, and the number of services described in SAWSDL is destined to increase rapidly in the future. Hence, there is an urgent need for a SAWSDL service matchmaker that can support SAWSDL service discovery. That is exactly why several SAWSDL matchmakers have been proposed such as URBE [10], SAWSDL-MX [11,12], etc. From the perspective of system architecture, however, most of them were dedicated to providing a specific matching strategy on the pre-fetched description information from SAWSDL documents. Therefore, users can not customize their preferred matching strategies according to different domain applications because of the fixed matching strategy. But in practice users often require this, since the effectiveness of a certain similarity measure depends a lot on the application domain [13] and the characteristics of services within that domain.

Furthermore, the matching strategy is considered as the core of a service matchmaker, which defines how to measure the similarity between the query and the service to return the most similar services to the user. There are many types of functional and nonfunctional service description information, which can be used for matchmaking, and many similarity measures for information retrieval (IR) are also available for each type of the description information. Thus how to measure the similarity between compared contents is very important for the matchmaker. Normally, different kinds of descriptions represent different facets about a Web service. It is considered that the more comprehensive service description information compared, the much fairer matching results obtained. This is exactly why most current matchmakers compare different types of description information at the same time and integrate them into an overall similarity value for ranking. For example, URBE compares the text and structure similarity, and ranks services based on weighted aggregation of structural and text matching scores, while SAWSDL-MX2 exploits support vector machine (SVM) to aggregate the matching of semantic annotations and WSDL structures. However, how to better aggregate these values from different compared information still deserves investigation.
Considering the problems presented above, the purpose of this paper is two-fold. First, this paper proposes a customizable SAWSDL service matchmaker, called SAWSDL-iMatcher, which supports several kinds of matching strategies. In SAWSDL-iMatcher, users can customize their preferred matching strategies for the evaluation of their requests and developers can easily deploy their newly designed matching strategies and compare them with other matching strategies. The previous work OWLS-iMatcher [7] employs iSPARQL for matchmaking OWL-S services which extends SPARQL to support imprecise query strategies. SAWSDL RDF mapping defines mappings between SAWSDL and RDF, and thus provides a possible way to use iSPARQL [14] to query SAWSDL services. However, the mappings are not complete. There is no corresponding mapping defined for type definitions as well as element declarations in WSDL, and thus there is also no corresponding mapping for model references on type definitions and element declarations in SAWSDL. Hence, if we use iSPARQL to query SAWSDL services, the semantic annotations in type definitions and element declarations in SAWSDL services can not be handled. To this end, inspired by iSPARQL, we propose the so-called iXQuery mechanism that extends XQuery with similarity joins for query evaluation, to query SAWSDL services, since WSDL is essentially based on XML.

Second, this paper evaluates various matching strategies in SAWSDL-iMatcher, and tries to find some empirical evidence for customizing matching strategies by analyzing evaluation results. We consider syntactic and semantic matching strategies, as well as statistical-model-based matching strategies that aggregate different matching values from comparing various types of description information by using statistical models. We evaluate these matching strategies on two datasets SAWSDL-TC2\(^1\) and Jena Geography Dataset (JGD)\(^2\). From the evaluation, some observations can be made for Web service discovery. For example, aggregating the results of simplest matching strategies on service name and interface annotations can often get better results than that returned by each single matching strategy. Such evidence would be useful for users to customize their matching strategies when they are confused on selecting suitable matching strategies.

In summary, this paper makes the following contributions:

- It presents a customizable matchmaker for SAWSDL services, called SAWSDL-iMatcher, which is constructed based on a so-called iXQuery mechanism that is an extension of XQuery with similarity joins for query evaluation.
- We have evaluated different matching strategies in SAWSDL-iMatcher and got some empirical evidence by analyzing the experimental results on different datasets. These empirical evidence would be helpful for users to cus-

\(^1\) http://projects.semwebcentral.org/projects/sawsdl-tc
\(^2\) http://fusion.cs.uni-jena.de/professur/jgdeval
tomize their requests.

The remainder of this paper is organized as follows. Section 2 formalizes the service matching and matching strategy used in this paper first, and describes the matching strategies in SAWSDL-iMatcher. Section 3 introduces SAWSDL-iMatcher, including the architecture, the proposed approach to extend XQuery with similarity joins as well as its application scenario (that is, how users use it). Section 4 evaluates the performance of the matching strategies in SAWSDL-iMatcher and also compares them with benchmark matchmakers on two datasets. Section 5 briefly compares the related work from the perspectives of matching strategy as well as system architecture. Finally, the conclusions and future work are summarized in Section 6.

2 SAWSDL Service Matchmaking

2.1 SAWSDL Service Definition

SAWSDL [6] is designed as an extension of WSDL, which enriches the service description with two kinds of attributes: model reference and schema mapping. A model reference can be used with every element within WSDL and XML schema. However, SAWSDL defines its meaning only for WSDL interfaces, operations, faults as well as XML Schema elements, complex types, simple types and attributes. And a schema mapping allows the specification of transformation functions on the WSDL elements to map instance data defined by that XML schema document to the semantic data of the concepts in a semantic model. Usually, the value of model reference is considered to be used in automated service discovery and composition, while the value of schema mapping is used when mediation code is generated to support invocation of a Web service [6].

Although SAWSDL provides only model reference and schema mapping, to distinguish from the standard WSDL services, we call a service described by WSDL together with model reference and schema mapping as SAWSDL service and the corresponding description document as SAWSDL document in the scope of this paper. From the point view of discovery, therefore, each SAWSDL service can be described by the abstract elements in WSDL (like types, message, operation, port type, etc.) and the model reference element. A SAWSDL service in this paper can be described formally as follows.

**Definition 1 (SAWSDL Service)** A SAWSDL service $s$ is described as a tuple $s = (sName, sText, I)$, in which
- **sName**: the name of the service,
- **sText**: the description text for the service,
- **I**: the set of interfaces defined in the service. A SAWSDL service interface $i \in I$ is represented as a 4-tuple $i = \langle i\text{Name}, i\text{Text}, i\text{A}, O \rangle$, where
  - *iName*: the name of the interface,
  - *iText*: the description text for the interface,
  - *iA*: the semantic annotation that is the value of attribute modelReference of Interface component,
- **O**: the set of operations. Each operation $o \in O$ is usually described by a tuple $o = \langle o\text{Name}, o\text{Text}, o\text{A}, iP, oP \rangle$, in which
  - *oName*: the name of the operation,
  - *oText*: the description text of the operation,
  - *oA*: the semantic annotations of the operation,
  - *iP*: the input parameters of the operation,
  - *oP*: the output parameters of the operation.

**Definition 2 (Operation Parameter)** Each Operation parameter $p \in iP \cup oP$ is described by a tuple $p = \langle p\text{Name}, p\text{Text}, p\text{A}, p\text{Type} \rangle$, in which
- *pName*: the name of the parameter.
- *pText*: the description text of the parameter.
- *pA*: the semantic annotations of the parameter.
- *pType*: the type of the parameter, which may be a base XML datatye, simple type or complex type defined in XML Schema XS. Meanwhile, each data type in XS is represented by a tuple $e = \langle e\text{Name}, e\text{Text}, e\text{A}, SE \rangle$, in which
  - *eName*: the name of the element.
  - *eText*: the description text of the element
  - *eA*: the semantic annotations of the element.
  - *SE*: the set of sub-elements of the data type defined in the type system.

It is worth noting that, in this paper, each request is also represented as a SAWSDL service. That is, each request is ultimately a SAWSDL document that contains the fields in the tuple defined in Definition 1. Actually, a user can either provide a SAWSDL document as her request, or specify the content of each field in the tuple based on which the query interface of SAWSDL-iMatcher will create a temporary SAWSDL document as her request. The user’s request does not need to contain all the fields defined in Definition 1 and some of the fields can be empty.

### 2.2 Formalization of Matching Strategy

**Definition 3 (Service Matchmaking)** Service matchmaking is defined as a procedure: given a collection of service advertisements $S = \{s_1, s_2, ..., s_n\}$
and a user’s request \( r \), compute all the similarity values between \( r \) and \( s_i \) according to a specified matching strategy, rank all the services in descending order according to similarity values, and return the ranked list to the user.

The most important factor in service matchmaking is the matching strategy, which is used to compute the similarity value between each service advertisement and the user’s request.

**Definition 4 (Matching Strategies (MS))** A service matching strategy is defined as a function \( ms : S \times S \mapsto [0,1] \), where \( ms(r,s) \) represents the matching value between service \( s \) and request \( r \) according to matching strategy \( ms \), where \( ms(r,s) = 1 \) indicates that service \( s \) can fully satisfy the requirement of \( r \), while \( ms(r,s) = 0 \) means that service \( s \) and query \( r \) are totally different, i.e., service \( s \) can not match query \( r \) at all. The higher the value of \( ms(r,s) \) is, the more that service \( s \) satisfies the requirement of \( r \).

A matching strategy normally involves two aspects: similarity measure and matching content. The former decides how to measure the similarity between two objects, and the latter decides which part of information in Web service description is used for matchmaking. There are several types of description information in SAWSDL service described above, so one can compare the whole description information as a structure matching, while the other can also compare the information in preferred descriptions. Therefore, we divide the matching strategies (MS) into two categories: single matching strategies (SMS) and combined matching strategies (CMS).

**Definition 5 (Similarity Measures (SM))** A similarity measure \( sm \) is a function \( sm : O \times O \mapsto [0,1] \), which associates the similarity between two objects \( o_1 (o_1 \in O) \) and \( o_2 (o_2 \in O) \) to a similarity score \( sm(o_1,o_2) \in [0,1] \). The similarity score of 0 stands for complete inequality and 1 for equality of the compared objects \( o_1 \) and \( o_2 \).

There are plenty of similarity measures available, which are suitable for comparing different types of description information. For example, several string similarity measures, like Levenshtein edit distance [15], Jaro distance [16], etc., can be used to measure the similarity between service name, operation name and so on.

**Definition 6 (Matching Content Extractor)** A matching content extractor \( mc \) is a function \( mc : S \times T \mapsto O \), which extract the description of type \( t \) \((t \in T)\) from service \( s \) \((s \in S)\), and output the content as an object \( o \) \((o \in O)\).

**Definition 7 (Single Matching Strategies (SMS))** A single matching strategy \( sms_t^{sm} \) is using one similarity measure \( sm \) to measure the similarity between service \( s \) and request \( r \) on specific type \( t \) \((t \in T)\) of description information, and is defined as a function \( sms_t^{sm} : S \times S \mapsto [0,1] \), where \( sms_t^{sm}(r,s) = \).
For example, levenshtein-edit-distance-based strategy (led) on service name \((sms_{s,Name}^{led})\) measures the similarity between service \(s\) and request \(r\) by using levenshtein edit distance to compute the similarity value between the request term and the service name.

Each type of the Web service description corresponds to one of the Web service facets. To discover Web services effectively, it is better if all these facets are considered. When the user’s request is comprehensive (providing all kinds of description information about the desired service), aggregating all the matching values on different parts of description is an intuitively good way to rank the services.

**Definition 8 (Similarity Aggregation Schemes (AS))** A similarity aggregation scheme \(as\) is a function \(as: [0,1] \times [0,1] \times \cdots \times [0,1] \mapsto [0,1]\).

There are various methods to aggregate several similarity scores into an overall similarity score, which are usually classified as linear and non-linear aggregation. The simplest example of linear aggregation is averaging, that means, every similarity score has the same weight in computing the overall similarity value. Non-linear aggregation is used when the overall similarity value cannot be written as a linear combination of all the similarity values.

**Definition 9 (Combined Matching Strategies (CMS))** A combined matching strategy \(cms^{as}_{MS}\) is using one similarity aggregation scheme \(as\) \((as \in AS)\) to aggregate the similarity scores computed from several matching strategies \(MS\) \((MS = \{ms_1, ms_2, ..., ms_n\})\), and is defined as a function \(cms^{as}_{MS}: S \times S \mapsto [0,1]\), where \(cms^{as}_{MS}(r, s) = as(ms_1(r, s), ms_2(r, s), ..., ms_n(r, s))\), \(ms_i \in MS\) can be either a single matching strategy or a combined matching strategy.

For example, \(cms^{average}_{\{sms_{s,Name}^{led}, sms_{s,Text}^{TF,IDF}\}}\) represents average matching strategy on service name and description text, which averages the similarity values from the matching of service name and service description text by using similarity measures Levenshtein edit distance and TF-IDF \([17]\) respectively.

From the definition above, each similarity measure on a certain type of description information specifies a single matching strategy. Therefore, different similarity measures performing on the same type of service description are treated as different matching strategies, and different types of matching content compared by the same similarity measure are also looked as different matching strategies. For example, we can establish a matching strategy on service name using Levenshtein edit distance, and another matching strategy also on service name but using Jaro distance. We can also establish a matching strategy by using cosine similarity measure to compare the set of parameters of an operation, together with another matching strategy which
compares the set of semantic annotations of an operation by using the same similarity measure.

There are many popular similarity measures from different applications, which have been implemented in generic Java library, such as SimPack \cite{18}, SimMetrics\cite{3}. Currently, SAWSDL-iMatcher proposed in this paper supports all the main measures in SimPack as well as some newly designed similarity measures, and many matching strategies have been built in.

2.3 Matching Strategies in SAWSDL-iMatcher

This section describes the matching strategies in SAWSDL-iMatcher which are evaluated in this paper. We propose a semantic matching strategy for semantic annotations of service operations and build several syntactic matching strategies for each type of service description and statistical-model-based matching strategies.

2.3.1 Syntactic Matching Strategies

- **Name-based Matching Strategies.** The services’ names are usually given by programmers who write the codes of services or generate WSDL documents. The programmer usually follows some coding conventions, and the names of variables as well as methods often have some meaning. In the same sense, the names of services are also meaningful. A good service name can briefly summarizes the capability of the service, like the title in a news report. Therefore, the matching strategies that compare service names would be useful if the service names contain either the function that the service (e.g., BookingFlightService) provides or the input/output parameters that the service involves (e.g., BookPriceService). However, the effectiveness of service discovery based on service names depends on the quality of services’ names, i.e., how meaningful the services’ names are.

To demonstrate the characteristics of the names of real world services, we have investigated the data set of real world services QWS-wsdl\cite{4}. After eliminating the services that are repetitive or no longer exist in the Internet, there are 1598 services left. The conventions for naming services can be categorized into six classes. Table 1 shows the percentage of each category and the corresponding examples with initial letter ‘a’. These services’ names that follow the first three conventions occupy about 94% and are meaningful (e.g., “AddressDistanceCalculator”), while the last three conventions do not seem adequate for deriving anything (e.g., “ABA”). It is quite easy to

\cite{3} http://www.dcs.shef.ac.uk/~sam/simmetrics.html
\cite{4} http://www.uoguelph.ca/~qmahmoud/qws/index.html
Table 1
Conventions for naming services in QWS-wsdl

<table>
<thead>
<tr>
<th>Conventions</th>
<th>Examples with initial letter ‘a’</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenated string with initial capital letters</td>
<td>AccountingService,</td>
<td>52%</td>
</tr>
<tr>
<td></td>
<td>AdaptiveInterfaceService,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AddressDistanceCalculator,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AddNumbersService,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AddressImageWSService,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AddressLookup,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AddressManager</td>
<td></td>
</tr>
<tr>
<td>Concatenated string not necessarily with initial capital letters</td>
<td>acitableService,</td>
<td>24.7%</td>
</tr>
<tr>
<td></td>
<td>acitraceService,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>acdvalidService</td>
<td></td>
</tr>
<tr>
<td>String concatenated by underline</td>
<td>alignment_wu_blastn_rawService,</td>
<td>17%</td>
</tr>
<tr>
<td></td>
<td>alignment_wu_blastn_emuService,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>alignment_consensus_consService,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>alignment_consensus_megamergerService,</td>
<td></td>
</tr>
<tr>
<td>Acronym in capitals</td>
<td>ABA, ARSA, ATTSMS</td>
<td>2.4%</td>
</tr>
<tr>
<td>Numbers or letters with numbers</td>
<td>2004, A7Postal, acq2xx</td>
<td>3.6%</td>
</tr>
<tr>
<td>Company name</td>
<td>AmazonBox, AmazonEC2</td>
<td>0.2%</td>
</tr>
</tbody>
</table>

decompose the service names with initial capital letters or underlines into a set of words, then some text-based similarity measures can be used to compare them. For the comparison of service names following the second convention, however, some tokenizers are needed to decompose them into a set of words first.

The performance of service-name-based matching strategies is irregular depending on the qualities of service names. Some service names can describe the capabilities of services well, while some can not. The reason for this may be that, unfortunately, there is no unified convention for naming Web services. Some standards specify “verb noun” phrases to name services, e.g., “FindBookPriceService”, and others may specify the composition of the interface elements to name services, e.g. “BookPriceService”. Therefore, the names of web services lack consistency, and seem to rely on the whim of the creator. As the simplest way for retrieving web services, service-name-based matching strategies would be much more effective in service matching if the names of services follow a convention and describe the capabilities of the services as much as possible.

Several similarity measures available can be used to measure the similarity value between a service name and a query term, such as levensthein edit distance [13], average string [19], Dice’s coefficient [20], Jaro coefficient [21], TF · IDF [17]. Several name-based matching strategies are built in SAWSDL-iMatcher by exploiting the string similarity measures implemented in Simpack [18].

• **Description-Tex-based Matching Strategies.** Description texts mostly consist of comments written in natural language by service developers. Generally, these description texts can make the code more understandable. There are two main advantages of these description texts [22]. First, description texts usually use simple sentences instead of using complicated
phrases and thus are easily processed. Second, these texts use natural lan-
guage in a specific way, which is called sublanguage [23]. A sublanguage is
characterized by a specialized vocabulary, semantic relations and syntax.
Thus the description texts of services in a specific domain may have similar
characteristics such as using domain specific terminology, abbreviations and
phrases, which may make it possible to find similar services based on the
matchmaking of description texts. Therefore, description texts have been
considered as important description information for discovery by several
matchmakers [7,24]. The description text in this paper is represented by
the classic vector space model: term frequency-inverse document frequency
(tf-idf) model [17]. The description texts were preprocessed, using a tok-
enizer, converting to lower case, stemming using the Porter stemmer [25],
and filtering with a list of stop-words firstly, then the description text of
SAWSDL document is represented as a vector, in which each dimension
corresponds to a separate term and the term weights are products of the
term frequency (local) and the inverse document frequency (global) para-
eters. SAWSDL-iMatcher currently supports seven vector-based similarity
measures (including cosine, dice, euclidean, jaccard, manhattan, overlap,
and pearson’s correlation coefficient) from Simpack [18] for comparing the
similarity of description text.

- **Semantic-Annotations-based Matching Strategies.** Semantic anno-
tations are usually considered as the most important information for auto-
mated discovery due to the formal semantic representation. Most match-
makers support the matching of semantic annotations for discovery in dif-
ferent ways [26,11,7]. In this paper, the similarity between semantic an-
notations can be measured by syntactically comparing the sets of semantic
concepts, which are called syntactic matching strategies on semantic annota-
tions. Generally, the degrees of semantic matching on semantic annotations
are determined by the subsumption relationships in domain ontologies and
often categorized into several grades such as exact, plugin, etc. [27]. However,
the semantic matching can not distinguish between two pairs of semantic
annotations if they belong to the same degree of semantic matching. For
example, there are one concept A in request and two concepts B and C
in services, and A is a super-class of both B and C. Then the matching
degrees of B and C are plugin, and thus it is difficult to judge whether B is
more similar to A than C. In this case, syntactic methods can be used in
the quantification of similarity, i.e., generating a concrete similarity value,
which would be useful for ranking services. Supposing the syntactic similarity
between A and B is 0.9, while the syntactic similarity between A and C
is 0.8, then we can say B is more similar to A than C.

Each service is represented by the unfolded concept expression (as in
OWLS-MX [8]) through reasoning the domain ontology with reasoner Pel-
let.⁵ The unfolded concept expression of a concept C includes all the ance-

---

⁵ [http://clarkparsia.com/pellet/](http://clarkparsia.com/pellet/)
tor concepts of C (including C) except the root concept Thing. For example, there are two services \( a = \text{BookPrice.wsdl} \) and \( b = \text{novel_price_service.wsdl} \) from SAWSDL-TC2. The unfolded concept expression of input concepts are \( \text{Unfold}^a_\text{Book} = \{ \text{Book}, \text{Monograph}, \text{Publication}, \text{PrintedMaterial}, \text{Object} \} \) for service \( a \) and \( \text{Unfold}^b_\text{Novel} = \{ \text{Novel}, \text{Book}, \text{Monograph}, \text{Publication}, \text{PrintedMaterial}, \text{Object} \} \) for service \( b \). Each unfolded concept expression is represented as a vector, and the term weight is set to 1 if the term appears, to 0 otherwise. The vector representations are \( V^a = [0, 1, 1, 1, 1, 1]^T \) and \( V^b = [1, 1, 1, 1, 1, 1]^T \) respectively, which have cosine similarity value of 0.91. Likewise, their unfolded concept expression of output concepts are the same \( \{ \text{Price}, \text{UntangibleObjects} \} \), which results in a similarity score of 1. The overall similarity of input and output semantic annotations is 0.955 by averaging. All the vector similarity measures implemented in Simpack have been implemented in SAWSDL-iMatcher for measuring the similarity between two unfolded concept expressions.

2.3.2 Semantic Matchmaking Strategy

As described above, semantic annotations are considered as the most important information for representing the semantics of service, almost every SWS matchmaker supports a certain kind of semantic matchmaking strategy \cite{27,28,29}. For example, SAWSDL-MX \cite{11} computes the degree of logic-based match for a given pair of service offer operation and service request by successively applying four filters of increasing degree of relaxation: Exact, Plugin, Subsumes and Subsumed-by.

SAWSDL-iMatcher supports a relaxed semantic matching strategy, which considers the semantic satisfaction from two directions, i.e., whether the output parameters of request are satisfied by the output parameters of service and whether the input parameters of service are satisfied by the input parameters provided by the request. The similarity between a service \( ws \) and a request \( r \) is computed as the following formula:

\[
\text{Similarity}(r, ws) = \alpha \cdot \text{Sim}_s(I_{ws}, I_r) + \beta \cdot \text{Sim}_s(O_r, O_{ws})
\]

where \( \alpha + \beta = 1 \), \( 0 \leq \alpha \leq 1 \), and the values of \( \alpha \) or \( \beta \) can be customized according to users’ preferences. For example, if users want a certain outcome and do not care about the inputs, the value of \( \beta \) can be set greater than \( \alpha \). If users want something that can process their data, \( \beta \) can be set less than \( \alpha \). \( I_x \) and \( O_x \) represent all the semantic annotations of input parameters and output parameters of service \( x \) respectively, and \( \text{Sim}_s(X, Y) \) means the degree that
set $Y$ satisfies $X$ and is defined as

$$Sim_x(X, Y) = \begin{cases} 
1, & |X| = 0, \ (1) \\
\sum_{r' \in X} \max_{s' \in Y} \text{SemanticMatching}(r', s'), & |X| \neq 0. \ (1')
\end{cases}$$

where $\text{SemanticMatching}(r', s')$ is defined in Algorithm 1, in which the semantic similarity between $r'$ and $s'$ is the degree that concept $s'$ satisfies the concept $r'$.

Algorithm 1 $\text{SemanticMatching}(r', s')$: returns the degree that concept $s'$ satisfies the concept $r'$.

**Input:** $r'$ is the request concept.
$s'$ is the serve concept.
$AV(r', s')$ represents the alignment value between $r'$ and $s'$.

**Output:** $\text{sim} \in [0, 1]$
$\text{sim}_{\text{semantic}} = 0, \text{sim}_{\text{alignment}} = 0, \text{similarity} = 0$
if $r' \in \text{Ancestor}(s')$ then
\[\text{sim}_{\text{semantic}} = 1\]
else
\[\text{sim}_{\text{semantic}} = \text{sim}_{\text{syntactic}}(r', s')\]
end if
if alignment and $r', s'$ belong to two different ontologies then
\[\text{Sim}_{\text{alignment}}(r', s') = AV(r', s')\]
end if
\[\text{similarity} = \text{Max} (\text{Sim}_{\text{alignment}}, \text{sim}_{\text{semantic}})\]

In Algorithm 1 $\text{Ancestor}(x)$ represents all the ancestor classes of class $x$ (including $x$), $\text{Sim}(r', s')$ represents a certain syntactic similarity measure which can estimate the semantic similarity between $r'$ and $s'$. The value of $AV(r', s')$ is computed by the ontology alignment toll Lily [30]. In this paper, we investigate several set similarity measures to measure the syntactic similarity between two concepts.

As we can see from the above definition, this semantic-satisfaction-based matching strategy is a relaxed semantic matching strategy. When the concept $s'$ is a super-concept or sibling concept of the concept $r'$, then concept $r'$ is possibly satisfied by concept $s'$, and an estimating value of this possibility is used as the semantic similarity.

2.3.3 Statistical-Model-based Matching Strategies

Different components in SAWSDL describe different facets of Web services. To discover Web services effectively, all these facets should be considered. When the user’s query is comprehensive, aggregating all the matching values on different parts of description is an intuitively way to get the overall similarity
score between the query and the service. Many matchmakers use empirical values as the weights of different types of description [10]. However, it is difficult to set the weights in practice, that is, it is difficult to say one type of information is much more important than another type of information in service matchmaking.

We decided to learn these weights from the known pairs of query and service, which we know if they are relevant or not. Each pair of query and service is represented by a vector space model, in which all the selected matching strategies and the relevance information are the dimensions of the vector space. Especially, each matching strategy represents one dimension of the feature vector, and the value in each vector is computed by the corresponding matching strategy. A pair of query and service is represented as a vector: 

\[
\text{Pair}(r_i, s_j) = \langle ms_1(r_i, s_j), ms_2(r_i, s_j), ..., ms_N(r_i, s_j), \text{relevant}(r_i, s_j) \rangle
\]

where \( r_i \) is the query and \( s_j \) is the service, \( ms_k(r_i, s_j) \) represents the similarity value between \( r_i \) and \( s_j \) according to the matching strategy \( ms_k \), \( N \) is the number of matching strategies used and \( \text{relevant}(r_i, s_j) \) specifies whether \( s_j \) is a relevant service to the request \( r_i \). All the vectors \( \{\langle r_i, s_j \rangle\} \) are considered as the training set. The statistical model learned is used to predict whether the new pair of query and service is relevant or not, specially, the probability that they are relevant. The matchmaker ranks the services for a request according to these probabilities.

Generally, the relevant services of a query are much less than the matched services, thus the number of irrelevant pairs of service and query is much larger than the number of relevant pairs of service and query. Therefore, there are much more negative instances than positive instances in the training dataset which in this case is actually an unbalanced training dataset. The unbalanced training dataset will lead to the effect that conventional machine learning methods are biased toward the larger class. To overcome this problem, cost sensitive model is developed by defining the penalty of each kind of samples. Our goal is to use the learned model to predict the probability of that a service is relevant to the query, then the matchmaker ranks services according to these probabilities. Normally, users want to find their desired services at the top of the ranking list, and do not care whether all the relevant services are returned. From this point of view, a false negative prediction is, therefore, considered to have more serious consequences than a false positive prediction in this work.

Currently, SAWSDL-iMatcher supports several statistical-model-based matching strategies by using several different algorithms from Weka [31], such as simple linear regression, J48 decision tree [32,33], logistic regression [34], support vector regression (\( \epsilon \)-SVR) [35,36], etc., to induce the statistical models.
3 SAWSDL-iMatcher

This section describes our proposed matchmaking system SAWSDL-iMatcher, from its architecture, core iXQuery framework to user interface.

3.1 Overview

Generally, whether a user can find the suitable services heavily depends on the matching strategy, since the effectiveness of matching strategies is data- and domain-dependent. It is time-consuming and inefficient for users to try several different matchmakers to get their desired services. Therefore, it is an urgent requirement for a SAWSDL matcher to support user-customizable matching strategies.

Our proposed matchmaking system, called SAWSDL-iMatcher, can satisfy this requirement. The “i” stands for “me” emphasizing its capability for supporting user-customizable matching strategies. SAWSDL-iMatcher supports numerous single matching strategies for matchmaking Web services with respect to each type of description information. For each kind of service description, SAWSDL-iMatcher uses different logic- or IR-based techniques to search for suitable services. SAWSDL-iMatcher also supports several good-performing aggregation schemata like weighting schemes, statistical-model-based schemes, etc., which can integrate the matching results that come from different matching strategies into an overall similarity value. When the user’s request involves several kinds of description information of Web service, a combined matching strategy can be customized. All the supported matching strategies are described in Section 2.3. Besides the customization, developers also can easily deploy their own matching strategies into SAWSDL-iMatcher.

Besides the above functionalities, SAWSDL-iMatcher also provides a simple statistics handler, which can generate evaluation results when the set of relevant services for a given query service is known.

3.2 Architecture of SAWSDL-iMatcher

The three-level architecture of SAWSDL-iMatcher is sketched in Figure 1. It contains a user interface, iXQuery framework and data model. In Figure 1 rectangles represent components in SAWSDL-iMatcher and the arcs represent data flow. The user interface level consists of one main component called query generator which helps users to generate requests in terms of iXQuery expression. The data model level contains several knowledge bases that
SAWSDL-iMatcher will use. The collection of SAWSDL/WSDL documents is the source for retrieval; The document alignment stores all the alignment results of heterogeneous domain ontologies and will be used in similarity measures which consider the alignment value as the similarity score between two heterogeneous concepts; The ontology database manages all the domain ontologies related to Web services; The classification cache is used to store the ancestor classes of the annotated concepts related to Web services. The cache is updated automatically when there is a new ontology added into the ontologies database or a new Web service added to the data model.

iXQuery framework is the core of SAWSDL-iMatcher, which has three main components:

- **XQuery Engine.** SAWSDL-iMatcher exploits Saxon\(^6\) as its XQuery engine. It passes the parameters to the similarity engine, receives the results from similarity engine and binds the similarity scores to variables in XQuery expressions. The number of services that can be retrieved by SAWSDL-iMatcher is theoretically decided by the capability of how many xml files the XQuery engine can handle.
- **Similarity Engine.** It embeds the similarity strategies by configuring the

parameters received from XQuery engine and returning similarity scores to XQuery engine.

- Description Extractor. It is used to extract different types of Web service description, like service name, description text, input parameters, and output parameters, etc. When a query expression is evaluated, XQuery engine will invoke the specified description extractor to get the corresponding Web service description.

An ontology reasoner is used to classify ontologies, and the resulting subsumption hierarchy is used by the semantic matchmaking strategies to define the semantic relationships between concepts. In this paper, we assume that all the SAWSDL descriptions in SAWSDL-iMatcher exploit OWL to represent the semantic models of SAWSDL, although SAWSDL specification does not specify a language for representing the semantic models. Therefore, SAWSDL-iMatcher exploits Pellet as its semantic reasoner and OWL API\(^7\) as the interface for accessing OWL-based domain-specific ontologies. We have not yet evaluated the impact of the ontologies size on SAWSDL-iMatcher, since the maximum number of ontologies that can be supported in SAWSDL-iMatcher probably depends on the scalability of the state-of-the-art ontology reasoners (i.e., the Pellet reasoner used in SAWSDL-iMatcher) and the characteristics of ontologies such as the ontology description language, the size of the ontology (including the depth of the hierarchy and the number of concepts). Currently, the number of ontologies that SAWSDL-iMatcher has tested is 43.

SAWSDL-iMatcher can perform matchmaking without any separated pre-processing of service descriptions (such as information extraction) or post-processing (such as ranking the results), since XQuery makes it possible to easily and efficiently extract information from native XML databases. Therefore, there is no need of parsing new SAWSDL documents collected by the system. Besides, there is no need of changing or extending original data representation, when new matching strategies consider different aspects of the SAWSDL description. SAWSDL-iMatcher provides a flexible interface for users to customize matchmaking strategies according to different requirements and application domains. For example, two users A and B want to find a weather forecast service. A has no accurate details about the input and output parameters that the required operation should have, then she can provide only a natural language description text as her request and then exploit the name or text matchmaking strategies which perform the syntactic matching between the text of request and the description text of candidate services. Another user B has necessary background knowledge to construct a comprehensive request, which contain several fields in the tuple of the service definition. Thus she can adopt one aggregated matchmaking strategy, in which each matching strategy is used to measure a certain kind of description and then the matching results

\(^7\) http://owlapi.sourceforge.net/
for several components are aggregated into an overall result according to an aggregation schema. It is also convenient for developers in the field of Web service to build and evaluate their own matchmaking strategies in an unified framework. SAWSDL-iMatcher is also possible to be extended to support other XML-based Semantic Web service languages, such as WSDL-S, OWL-S, etc. (shown in the dotted line in Figure [1]). Currently, SAWSDL-iMatcher only supports the matching of SAWSDL services.

3.3 Extension of XQuery with Similarity Measure

In this section, we present the iXQuery approach to extend XQuery with similarity joins for SAWSDL service matchmaking. XQuery supports two kinds of functions: built-in functions and user-defined functions. XQuery includes over 100 built-in functions for string values, numeric values, data and time comparison, node and QName manipulation, sequence manipulation, Boolean values and so on. Unfortunately, these built-in functions can not satisfy the need of similarity comparison. In addition to the built-in functions, XQuery allows users to declare functions of their own, which may provide a possible way to make XQuery supporting similarity joins. There are two ways to define such functions: user-defined function and external function in external environment.

SAWSDL-iMatcher chooses Saxon as a Java implementation of XQuery to support similarity joins, since Saxon can perform queries on individual files or on collections of files without having to install a XML database.

iXQuery makes use of static methods mechanism to support similarity joins during the evaluation of iXQuery expression. Multiple existing similarity measures library like SimPack, are easily employed to compose sophisticated user- and data-specific similarity joins. The procedure of invoking external functions is simple: a call to an external similarity function is made and arguments are passed to the function if the namespace of the function is prefixed with the XQuery expression. The similarity between the arguments is computed and bound to the returned variable in the query.

Considering the simple iXQuery example in Listing [1] it returns the services list whose names are similar to the name of the request y according to the similarity score of edit-distance-based similarity measure. In this case, the namespace points to the class “editdistance”, with the Java protocol prefix indicating that this should be interpreted as external function extension in Java (line 1). The for clause iterates over an input sequence (lines 2-4) and calcu-

---

8 http://www.w3.org/2005/02/xpath-functions/
9 http://www.w3.org/TR/xquery/
lates the similarity score between the passed arguments by a static method "similarity" which defined in class "editdistance" (line 5). Finally, it returns a ranked list of Web services together with their corresponding similarity scores.

```
1 declare namespace ed="java:editdistance";
2 for
3   $x in collection(<servicesURL>),
4   $y in doc(<queryURL>)
5 let $s:=ed:similarity($x/wsdl:definitions/wsdl:service/@name,
6                    $y/wsdl:definitions/wsdl:service/@name)
7 order by $s descending
8 return <pair>{document-uri($x),$s}</pair>
```

Listing 1. Example of iXQuery matchmaking strategy

### 3.4 Description Extractor Library

When a user of SAWSDL-iMatcher wants to customize her own matching strategies, the first thing she needs to do is to extract the matching content from SAWSDL document by utilizing XPath expression. However, it is not a good idea to ask users to write the matching contents in XPath expressions each time they formalize their requests, especially for people with no or little XQuery and SAWSDL background.

For this reason, iXQuery specifies a preliminary classification of matching content according to the types of SAWSDL description in Section 2 and also provides a user defined function library “part”, which includes all the description extractors users can use to construct their query expressions.

The listing 2 shows the same example of iXQuery matching strategy as that described in Listing 1. In this example, description extractor `Name` is used to reduce the burden of writing content extractors in XPath expression.

```
1 import module namespace part="http://127.0.0.1/sawsdl/library/" at "function.xq";
2 declare namespace ed="java:editdistance";
3 for
4   $x in collection(<servicesURL>),
5   $y in doc(<queryURL>)
6 let $s:=ed:similarity(part:Name($x),part:Name($y))
7 order by $s descending
8 return <pair>{document-uri($x),$s}</pair>
```

Listing 2. Example of iXQuery matchmaking strategy with description extractors
3.5 Customizing Query Behavior

In this section, we will illustrate how users can customize their query behavior. If one wants to use SAWSDL-iMatcher to discover the services, the only thing she needs to do is to specify an iXQuery matching strategy like List 2 for her request. Then, XQuery engine evaluates the query expression and returns the ranked list of Web services to her. In SAWSDL-iMatcher, there are two ways to generate the iXQuery matching strategy such that users could make a choice according to their preference.

- For non-expert user who is not familiar with SAWSDL-iMatcher or XQuery, she can specify her request content and the preferred matching strategies through the query interface that SAWSDL-iMatcher provides. SAWSDL-iMatcher first creates a temporary SAWSDL document according to the request content, and then generates the iXQuery expression via the query generator according to the specified matching strategies. Finally, SAWSDL-iMatcher sends the iXQuery expression to XQuery engine for evaluation.

- For expert user who is familiar with SAWSDL-iMatcher and XQuery, she can customize a new iXQuery expression manually, and then invoke the XQuery engine directly.

The results of [13] show that the best-performing similarity measure seems to be domain dependent. In SWS matchmaking, there are several matching strategies available, some of which are shown in Section 2.3. Each matching strategy has its own use cases due to its strengths and weaknesses in matchmaking services. Hence, the selection of proper matching strategy for specific application context would be crucial and challenging. In most matchmakers, users can not specify their preferred matching strategies to complete their tasks.

SAWSDL-iMatcher supports a customizable mechanism, in which users can deploy their preferred matching contents and similarity measure together into a matching strategy according to the characteristics of user’s query and the collection of Web services. The significance of customization in SAWSDL-iMatcher lies in the fact that different user queries may need different matching strategies. Users can either specify the matching strategies directly according to their prior knowledge from using various service matchmakers, or get some recommendations from SAWSDL-iMatcher who has learned some empirical evidence that may be helpful for the selection of matching strategies.

In SAWSDL-iMatcher, users can customize the query behavior by setting the properties of iXQuery expressions from the following aspects:

- Customizing matching contents. A SAWSDL service may contain several kinds of description information. For a certain request, one may wonder
which kinds of descriptions need to be compared to get better results, since different ranked lists will be returned when different descriptions are compared. Users can specify their preferred matching contents, like the “part:Name($x)” in Listing 2.

- Customizing similarity measures. For a certain description, there may be several similarity measures available for use. For example, there are several string similarity measures that can be used to compare service names. Users can specify a certain similarity measure to compare a certain kind of description. For example, users can specify “edit distance” to measure the similarity between the query’s name and the service’s name.

- Customizing aggregation schema. When users specify several matching strategies, they need to specify a way to integrate the matching values obtained from those matching strategies into an overall matching value for ranking. For example, users can specify the weights for the matching of input/output parameters in the semantic matching strategy.

- Customizing filtering conditions. Users can specify a condition to filter the Web services that they are interested in by using the XQuery `where` clause.

4 Evaluation

This section first describes the goal of this evaluation, that is, evaluating the performance of the built-in matching strategies in SAWSDL-iMatcher. Then, several matching strategies are evaluated and the comparison with other benchmark matchmakers from S3 contest 2009 is given. Finally, some concluding remarks are presented.

4.1 Evaluation Goal

The matching strategy is a very important component in a matchmaker. It decides which kinds of description information of Web service is used for matching and which similarity measure is good at computing the similarity value between each pair of service and query. As discussed in Section 2, there are many matching strategies available. Various matching strategies exploit different similarity measures to compare different types of description information, such as service description text [24], service interface [8,37]. Comparison of various matching strategies is helpful to improve the effectiveness of service matchmaking.

Therefore, the goal of this evaluation is to evaluate the built-in matching strategies in SAWSDL-iMatcher from single matching strategies to statistical-model-based matching strategies, and try to point out the strengths and weak-
nesses of each kind of matching strategy. Meanwhile, some lessons and empirical evidence are learned through the comparison with existing matchmakers. In the end, users can be inspired by these results in customizing their queries when they are confused on selecting or designing good-performing matching strategies for an application.

4.2 Performance Measures

In this section, we briefly review the performance measures used in this evaluation, which are commonly used in information retrieval [38]. Precision and Recall Precision and recall are two widely used performance measures for measurement of search effectiveness. Given a query \( q \), precision \( P \) is the proportion of the relevant documents retrieved by the matchmaker to all the retrieved documents, and is described as

\[
P = \frac{|\text{relevant documents} \cap \text{retrieved documents}|}{|\text{retrieved documents}|}
\]

Recall \( R \) is the proportion of relevant documents which have been retrieved to all the relevant documents, and is described as

\[
R = \frac{|\text{relevant documents} \cap \text{retrieved documents}|}{|\text{relevant documents}|}
\]

Mean Average Precision Average precision \( AP \) is the average of precisions computed at each point of the relevant documents in the returned ranked list [39].

\[
AP = \frac{\sum_{r=1}^{N} (P@r \times \text{rel}(D_r))}{\sum_{r=1}^{N} \text{rel}(D_r)}
\]

where \( N \) is the number of retrieved documents, \( P@r \) is the precision when considering only the top \( r \) results in the ranked list, \( D_r \) is the \( r \)th document in the ranked list, \( \text{rel}() \) is a two-valued function on the relevance. This metric is referred to geometrically as the area under the precision-recall curve. Mean average precision (MAP) is the average precision of multiple queries.

\[
\text{MAP} = \frac{\sum_{i=1}^{M} AP_i}{M}
\]

where \( M \) is the number of evaluated queries.

Macro-Averaged Precision and Recall As the evaluation of a single query is oftentimes not sufficient to make a statistically significant statement, many queries are involved and the macro average precision over all queries should be
computed as it gives equal weight to each user query. Ceiling interpolation is used to estimate precision values at each standardized recall level, since each query likely has a different number of relevant services.

4.3 Test Collections

In this evaluation, we use two SAWSDL datasets from Semantic Service Selection (S3) contest 2009\(^{10}\): SAWSDL-TC2 and Jena Geography Dataset (JGD).

SAWSDL-TC2 is semi-automatically derived from OWLS-TC 2.2\(^{11}\) using the tool OWLS2WSDL\(^{12}\). This collection consists of 894 Semantic Web services from 7 domains (education, medical care, food, travel, communication, economy, weapon) and 26 requests written in SAWSDL (for WSDL 1.1).

The original dataset JGD is a collection of about 200 geography services that have been gathered from web sites like seekda.com, xmethods.com, webserviceclist.com, programmableweb.com, and geonames.org. This evaluation uses one of its sub-collection, which is created and annotated for SAWSDL-MX2 by Klusch and Kapahnke \(^{12}\). It consists of 50 services and 10 requests, and is used in JGD cross evaluation track in the S3 contest 2009.

4.4 Benchmark Matchmakers

To compare the matching strategies in SAWSDL-iMatcher, the matchmakers that participated in S3 contest 2009 are considered as the benchmark matchmakers. They are URBE \(^{10}\), SAWSDL-MX2 \(^{12}\), COM4-SWS , iMatcher 3/1, WSColab\(^{13}\), Themis-S \(^{24}\), IRS-III \(^{26}\), which are described in details in Section 5.

All the results of performance tests are from the automated evaluation in the Semantic Web Service Matchmaker Evaluation Environment (SME2)\(^{14}\). All the SME2 plugin matchmakers evaluated in this paper are available online at: http://www.ifi.uzh.ch/ddis/people/wei.


\(^{11}\)http://semwebcentral.org/projects/owls-tc/

\(^{12}\)http://projects.semwebcentral.org/projects/owls2wsdl/

\(^{13}\)http://www.ibspan.waw.pl/~gawinec/wss/wscolab.html

\(^{14}\)http://projects.semwebcentral.org/projects/sme2/
4.5 Experimental Results

This section describes the results of this experiment. First, we present the performance of single matching strategies on different types of description information. Then, we take a look at the statistical-model-based matching strategies. Finally, the comparison with the benchmark matchmakers is given.

4.5.1 Single Matching strategies

Service-Name-based Strategies. We start our evaluation with the comparison of simple iXQuery matchmaking strategies on service name. Figure 2(a) shows the performance comparison on SAWSDL-TC2. It depicts that Dice’s coefficient-based [20] matching strategy clearly outperforms all other matching strategies in terms of precision and recall, such as Levenshtein edit distance [15], Jaro coefficient [21], TF-IDF [38] and average string [19]. Figure 2(b) shows the performance comparison of the above matchmaking strategies on Jena Geography Dataset (JGD). Here, Dice’s coefficient-based strategy also clearly outperforms all the other matchmaking strategies until about one-quarter of the relevant services have been retrieved. At the same time, it has the highest mean average precision (MAP) of 0.4469 in this comparison. On these two datasets, Dice’s coefficient-based strategy is very well suited for matching service name, therefore, we will use it to compare service names.

The results of Figure 2 show that the name-based matching strategies perform better on SAWSDL-TC2 than on JDG, since the naming convention for services and queries in SAWSDL-TC2 is much more consistent than that in JGD. The names in SAWSDL-TC2 are concatenated strings, in which each word corresponds to one parameter name, while names in JGD do not have
We can also analyze the average precision of each query for each service-name-based matching strategy. On SAWSDL-TC2, there are over 70.4% queries with average precision (AP) over 0.5 and 14.8% queries with AP less than 0.3. On JGD, there are 50% queries with AP over 0.5 and 20% queries with AP below 0.1. To illustrate this, consider the worst query “Altitude Request” as an example, it has two relevant services whose names are “GeoNames_SRTM3” and “EarthTools_Elevation_Height_Above_Sea_Level”, which are totally different syntactic descriptions. These results show that the performance of service-name-based matching strategies is irregular depending on the characteristics of names in different datasets.

The reason for this may be that, unfortunately, there is no standard convention for naming Web services. Some suggest use “verb noun” phrases to name services, e.g., “FindBookPriceService”, while others may specify the composition of the interface elements to name services, e.g., “BookPriceService”. Therefore, the names of web services lack consistency, and seem to rely on the whim of the creator. As a simplest way for retrieving web services, service-name-based matching strategies would have much more sense in service matching if the names of services follow a standard convention and can describe the capabilities of the services as much as possible. In this paper, we suggest that the services be named by the following conventions:

- Use “verb noun” phrases to name services. The verb specifies the utility of the service, i.e., what kind of operation this service can provide. The noun specifies the entities that the service handles and returns. For example, in a service name “BookingFlightService”, the word “Booking” specifies the utility of the service and “Flight” specifies the entities that the service mainly handles.
- Use CAMEL case: initially capitalize each word in service name. This would make the tokenization easier and more precise.
- Avoid implementation and protocol information, like words “soap”, “http”, “java”, etc. This kind of information is not important for describing the capability of services.
- Avoid the word “service” in service names. Right now, many service names contain the word “service”, which is not important for describing the capability of services. If two names both have the word “service” at the end, it is possible to bring inappropriate results when they are compared by a string similarity measure. For example, two services named “BookService” and “CarService” are actually totally different. If we use a string similarity measure, e.g., Dice’s coefficient to compute the similarity between them, we will get a similarity value 0.62 because of the common substring “Service”.

Service-Description-Text-based Strategies. Description text is the value
Table 2
Percentages of services without description text

<table>
<thead>
<tr>
<th>Datasets</th>
<th>definition</th>
<th>operation</th>
<th>service</th>
<th>message</th>
<th>types</th>
</tr>
</thead>
<tbody>
<tr>
<td>JGD-50(50)</td>
<td>19 (38%)</td>
<td>7 (14%)</td>
<td>33 (66%)</td>
<td>28 (56%)</td>
<td>29 (58%)</td>
</tr>
<tr>
<td>JGDFull(203)</td>
<td>91 (44.8%)</td>
<td>28 (13.8%)</td>
<td>144 (70.9%)</td>
<td>125 (61.6%)</td>
<td>122 (60.0%)</td>
</tr>
<tr>
<td>QWS-wsdl(1977)</td>
<td>1779 (90.0%)</td>
<td>1204 (60.9%)</td>
<td>1508 (76.3%)</td>
<td>1940 (98.1%)</td>
<td>1823 (92.2%)</td>
</tr>
</tbody>
</table>

of wsdl:document element which is allowed to appear in each WSDL language element. Figure 3 only illustrates the results of description-text-based matching strategies on the JGD dataset, since services in SAWSDL-TC2 do not have any description text. Figure 3(a) shows the macro average recall vs. precision curves of the matching strategies on the description text within the corresponding operation. The results show that the overlap outperforms all other vector similarity measures (with MAP of 0.53). Figure 3(b) additionally shows the macro average recall vs. precision curves of the matching strategies on all the description texts within the service (including all the documentation in each WSDL language element). The curves show that the matching strategies based on description text within operation component have much higher precisions that on all the description texts. That is, description text within operation is more suitable for service discovery than other description texts. The main reason for this maybe that description texts within elements rather than operation describe something else which is not related to the functionality of the service, and thus bring noise to the functionality description. Description text within operation at the other hand actually describes what the operation can do.

Furthermore, we have compiled statistics of description texts in several different real service collections shown in Table 2 where the number of each cell $(r, c)$ represents the number of services in collection $r$ (row) which do not have any description text within the component $c$ (column). These statistics show that the services that do not have documentation within operation have the lowest percentage in each dataset, compared with other components. In another words, description text appears in service operation with much higher probability than it appears in other service components. In real dataset QWS-wsdl, the services without description text in operation has percentage of 60.9%, since many WSDL documents are automatically generated without any configuration of documentation for operations. However, the fact that the description text is useful for Web service discovery can not be easily dismissed.

Semantic-Annotations-based Strategies. Figure 4 shows the performance comparison of matchmaking strategies on semantic annotations, including syntactic and semantic strategies described in section 2.3. On SAWSDL-TC2, Figure 4(a) illustrates that euclidean-distance-based matchmaker slightly outperforms other matchmakers with MAP of 0.692. Generally, there is no big difference between our semantic matching strategy and most of the syntac-
Fig. 3. The performance comparison of description text-based matching strategies on JGD.

tic matching strategies on both two datasets. The semantic matching strategy slightly outperforms several syntactic matching strategies such as jaccard-, cosine-, and overlap based matching strategies until about 45% of the relevant services have been retrieved, after which the average precision of the semantic matching strategy drops rapidly.

Figure 4(b) shows that the overlap-coefficient-based matching strategy outperforms other matching strategies, with MAP of 0.5798 on JGD dataset. The jaccard-coefficient-based matching strategy is the second best one on this dataset with MAP of 0.5795. It outperforms overlap-coefficient-based matcher after half of the relevant Web services are retrieved. Semantic matching strategy has the lowest average precision at the beginning, but outperforms overlap, cosine, euclidean, and manhattan-based matching strategies after half of the relevant services have been retrieved.

The results of Figure 4 also show that the semantic-annotations-based matching strategies perform better on SAWSDL-TC2 than on JGD. This may because of the different quality of annotations between the two datasets. In dataset JGD, there are some semantic annotations that do not exist in the domain ontologies. For example, the output concept “protont.owl# Altitude-AboveSeaLevel” of query “5914_6770_Altitude Request.wsdl” in JDG does not exist in the ontology “protont.owl”. Such error-prone annotations fail to match with other semantic annotations, and thus some relevant services can not be returned.

Furthermore, we also compared the semantic matching strategy with ontology alignment. Experimental results show that there is no improvement when the results of alignment are used in this dataset. This is because there are no test cases such that the request and service descriptions which have heterogenous
semantic annotations are actually similar or related.

4.5.2 Statistical-model-based Strategies

In this evaluation, the selected matching strategies are service-name-based matching strategy, description-text-based matching strategy, the syntactic matching strategies on semantic annotations and semantic matching strategies. We do not consider the matching strategies on XML schema due to the high computational complexity. Description-text-based matching strategies are only used for JGD, since there is no description text in the service information of the test collection SAWSDL-TC2. To avoid the bad influence of noisy data, we choose the best-performing matching strategy for each type of descriptions. Therefore, the matching strategies used on SAWSDL-TC2 are service-name-based matching strategy Dice coefficient, syntactic matching strategy on semantic annotations euclidean, and semantic matching strategy. The matching strategies used on JGD are service-name-based matching strategy Dice coefficient, description-text-based matching strategy pearson, syntactic matching strategy on semantic annotations overlap, and semantic matching strategy.

In this experiment, each test collection is represented by a set of vectors with cardinality $|Q| \times |S|$, in which $Q$ and $S$ represent the sets of queries and services respectively in the test collection. Each vector is described in the Section 2.3.3. The set of vectors are divided into $N$ folds, and each fold consists of all the vectors related to one query. Each time, we take one fold as test set (related to one test query) and learn the model using specific learner on the remaining $N - 1$ folds, and then measure the effectiveness on the test query. Take the query $r_i$ as example, the training vectors are represented as $\bigcup_{r_k \in Q \setminus \{r_i\}} \bigcup_{j=1}^{S}$.
\{(r_k, s_j)\}, and the test vectors are represented as \(\bigcup_{j=1}^{S} \{(r_i, s_j)\}\). Finally, the macro-average of the results of the \(N\) runs is considered as the performance of the statistical-model-based matching strategies on the whole test collection. This approach is the standard practice of \(N\)-fold cross validation in machine learning.

Figure 5 shows the performance comparison of statistical-model-based matchmakers. On SAWSDL-TC2 (shown in Figure 5(a)), the logistic-regression-based matchmaker performs better than other statistical-model-based matchmakers with MAP of 0.749, although it is slightly outperformed by \(\epsilon\)-SVR-based matchmaker at the beginning. The second best statistical-model-based matchmaker is \(\epsilon\)-SVR with MAP of 0.723.

On JGD, \(\epsilon\)-SVR-based matchmaker performs best with MAP of 0.71. The second one is the logistic-based matchmaker with MAP of 0.67. Simple linear regression performs worst on both datasets. AdaBoosting algorithm on the weak learner J48 outperforms J48 decision tree itself on both datasets, but it is still outperformed by logistic and \(\epsilon\)-SVR-based matchmaker on both datasets.

Figure 6(a) indicates that, on SAWSDL-TC2, logistic-based matchmaker and \(\epsilon\)-SVR-based matchmaker perform better than each of the single strategies which are used to learn the model. On JGD (shown in Figure 6(b)), the same conclusion can also be drawn, although they are outperformed by single matching strategies such as semantic-annotations-based overlap strategy and description-text-based Pearson strategy at the very beginning. These results validate that each type of information contributes differently in service matchmaking. Combined matching strategies on different types of description information can improve the effectiveness of matchmaking by learning from other’s strong points to offset one’s weaknesses.
4.5.3 Performance Comparison with Benchmark Matchmakers

This section compares the performance of our matchmakers with others from the S3 contest 2009, in which we participated with a simple linear-regression-model-based matchmaker iMatcher3/1, which was also developed in SAWSDL-iMatcher. In that contest, the non-logical-based matchmaker URBE is the best matchmaker on SAWSDL-TC2, and the ranking is based on the weighted aggregation of structural and text matching scores. Our previous matchmaker iMatcher3/1 was not effective enough on both datasets. Analyzing the results we hypothesized that iMatcher3/1’s drawback may lie in its linear combination of the similarity values. Therefore, we learned the non-linear models for matching in subsequent experiments presented here.

The results from Table 3 show that the logistic-regression-based matchmaker on service name and service interface performs much better than simple linear-regression-based model with MAP of 0.749. It also outperforms all the compared benchmark matchmakers. Meanwhile, euclidean distance-based matching strategy on semantic annotations even outperforms statistical-model-based matchmaker SAWSDL-MX2.

Table 4 shows the comparison results on JGD dataset. Single matching strategies Overlap on semantic annotations can obtain better results with MAP of 0.58, which already outperforms all the compared systems. $\epsilon$-SVR- and logistic-based matchmakers outperform all the benchmark matchmakers, i.e., statistical-model-based matchmaker has much higher performance than single matching strategies.

The different results from SAWSDL-MX2 with our $\epsilon$-SVR-model-based matching strategy in both datasets, also indicate feature selection is also very impor-
### Table 3
Comparison of matchmakers on SAWSDL-TC2

<table>
<thead>
<tr>
<th>Systems</th>
<th>URBE</th>
<th>SAW-SDL-MX2</th>
<th>COM4SWS</th>
<th>iMatcher3/1</th>
<th>SAWSDL-iMatcher</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AP</td>
<td>0.727</td>
<td>0.679</td>
<td>0.681</td>
<td>0.635</td>
</tr>
</tbody>
</table>

### Table 4
Comparison of matchmakers on JGD

<table>
<thead>
<tr>
<th>Systems</th>
<th>WSColab</th>
<th>SAW-SDL-MX2</th>
<th>SAW-SDL-MX1</th>
<th>The-mis-S</th>
<th>IRS-III</th>
<th>iMatcher3/1</th>
<th>SAWSDL-iMatcher</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AP</td>
<td>0.54</td>
<td>0.45</td>
<td>0.41</td>
<td>0.48</td>
<td>0.41</td>
<td>0.53</td>
</tr>
</tbody>
</table>

### Table 5
Characteristics Comparison between SAWSDL-TC2 and JGD

<table>
<thead>
<tr>
<th>Component</th>
<th>Characteristics</th>
<th>SAWSDL-TC2</th>
<th>JGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Description Text</td>
<td>Consistency of queries and services</td>
<td>+++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Service Description Text</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Operation Description Text</td>
<td>-</td>
<td>+++</td>
</tr>
<tr>
<td>Semantic Annotation</td>
<td>Quality of Annotations</td>
<td>+++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Annotating Style</td>
<td>top level</td>
<td>top and bottom level</td>
</tr>
<tr>
<td></td>
<td>Heterogeneous annotations</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

* represents there is no such element or situation.
+++ represents the degree of minimal, middle, and maximum respectively.

tant to get a better model even using the same machine learning method. On SAWSDL-TC2, logistic-regression-based matchmaker outperforms slightly \( \epsilon \)-SVR-based matchmaker, while \( \epsilon \)-SVR-based matchmaker outperforms logistic-based matchmaker one JGD.

### 4.6 Summary

Several kinds of matching strategies in SAWSDL-iMatcher have been evaluated, from service-name-based matching strategies, description-text-based matching strategies, syntactic/semantic matching strategies on semantic annotations, to statistical-model-based matching strategies. It is difficult to determine a clear “winner” for Semantic Web service discovery because all the matching strategies have their strengths and weaknesses and, hence, are suitable for different tasks and contexts. Nevertheless, we can make the following important observations when looking at the results as well as the characteristics comparison of each description component in the SAWSDL-TC2 and JGD datasets (shown in Table 5).

- Service-name-based matching strategies are the worst one compared to description-text-based matching strategies and semantic-annotation-based matching strategies. As the simplest matching method, there is a need for service offers to name their services with as much information as possible. In general, *Dice’s-coefficient*-based matching strategy is a good choice if the
user’s request contains service name.

- For semantic annotations, the semantic matching strategy is slightly worse than syntactic matching strategies, although they perform very close on the whole.
- Semantic annotations can describe service capabilities very well in most cases, and single matching strategy on semantic annotations can obtain better results than that on service name and description text.
- Statistical-model-based matching strategies are good at aggregating the matching values on simple service description component, such as service name, description text, and interface annotations. However, the performance is also decided by the statistical model. Usually, non-linear statistical models outperform linear statistical models. That means it is not a good idea to aggregate the matching values from different matching strategies in a linear way.
- Logistic regression and $\epsilon$-SVR on service name and semantic annotations of the operation elements can get better results than single matching strategies and also other statistical-model-based matching strategies on the same service descriptions.

In summary, however, we find that the best performing similarity strategy seems to be domain dependent, echoing the results of [13]. It would, therefore, be prudent to further investigate this issue. Also, we should enable users to easily adopt matchmaking strategies to their own domains, which is one of the main goal of our approach.

5 Related Work

Semantic Web service matchmaking is a hot topic in the fields of both Semantic Web and Web service. An abundance of approaches for service matchmaking mainly focus on comparing different aspects of service description including functional and non-functional ones (such as the work presented in [40]). The work presented in this paper concerns only the function-based matchmaking. The functional properties of SWS mainly include service inputs, service outputs, preconditions and effects [3]. In this paper, we also consider other service descriptions as functional properties of SWS, such as service name, description text, service structure.

Some of the research work in the literatures [37,41,8,27,42] focus on the matchmaking of service I/O, i.e., the data semantics of the Web service. In [37], a ranked matching algorithm is proposed, in which the matched service descriptions include service inputs, service outputs, service quality and service categories. And the matching of service inputs and outputs depends on the subsumption relations defined in the domain ontology. Paolucii et al. [27] use
logic-based matching approach to compare service inputs/outputs in the service profile. Hull et al. [41] describe stateless service by inputs, outputs and the relationships between them, and provide a service matching algorithm that takes all these descriptions into account. The work presented in [42] extracts the semantic constraints for service I/O concepts from description text and extends the matching of service I/O concepts together with the matching of semantic constraints.

Some hybrid SWS matchmakers have been proposed, which support the matching of semantic description as well as syntactic description. Klusch et al. [8,9] exploit logic- and non-logic-based matching strategies to match service I/O concepts. Kiefer et al. [7] propose an OWL-S matchmaker, which supports several syntactic matching strategies as well as semantic matching strategy, and machine-learning-based methods are also proposed to aggregate the results from several matching strategies.

There are also different SAWSDL matchmakers. Table 6 summarizes the comparison among different matchmakers with respect to the different matching strategies. In these matchmakers, only Themis-S [24] and IRS-III [26] use single matching strategy to compute the similarity between query and service. Themis-S only uses description text of Web service to compare the similarity value between a Web service and the query. The matching method used in Themis-S is enhanced Topic-based Vector Space Model (eTVSM), which can be regarded as a meet-in-the-middle approach between heavyweight Semantic Web technologies and easy-to-use syntactic information retrieval models. IRS-III uses semantic matching on semantic annotations to compare the similarity between a Web service and the query. IRS-III is an ontology-based reasoning and SWS broker environment based on OCML and LISP, and domain ontologies are used to define service input and output types whose inheritance structure is used for the matchmaking.

Some other matchmakers match different kinds of description information of Web service and aggregate all the matching results into the overall similarity value with different aggregating schemes. URBE [11] is a novel approach for Web Service retrieval based on the evaluation of similarity between Web Service interfaces. It is a non-logical-based matchmaker and the ranking is based on the weighted aggregation of structural and text matching scores. WSColab considers three kinds of set-based similarity measure to compute the similarity among each kind of tags (input, output, and behavior, which are not intrinsic description for Web services) information and uses unsupervised aggregation scheme average to aggregate all these similarity values. SAWSDL-iMatcher and SAWSDL-MX2 [12] use supervised aggregation scheme to aggregate several similarity values computed by different single matching strategies. The combined matching strategy used in SAWSDL-MX2 is support vector machine, whose features are the similarity values on semantic annotations and
Table 6
Comparison of matching strategies in different matchmakers

<table>
<thead>
<tr>
<th>Systems</th>
<th>mc</th>
<th>sm</th>
<th>aggregation scheme as supervised</th>
<th>unsupervised</th>
</tr>
</thead>
<tbody>
<tr>
<td>URBE [10]</td>
<td>p_name, p_name, p_type</td>
<td>information theory, information loss based sm</td>
<td>-</td>
<td>weighted average</td>
</tr>
<tr>
<td>SAWSDL-MX2 [12]</td>
<td>annotations, WSDL structure</td>
<td>SVM</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SAWSDL-iMatcher</td>
<td>p_name, description text, annotation</td>
<td>Dice’s coefficient, vector-based sm, semantic sm</td>
<td>Logistic, $\epsilon$-SVR</td>
<td>-</td>
</tr>
<tr>
<td>WSColab</td>
<td>input tag, output tag, behavior tag</td>
<td>set-based TF-IDF</td>
<td>-</td>
<td>average</td>
</tr>
<tr>
<td>Themis-S [23]</td>
<td>description text</td>
<td>$\epsilon$TSVM</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IRS-III [26]</td>
<td>annotation</td>
<td>semantic matching</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The whole WSDL structure. SAWSDL-iMatcher supports several statistical-model-based aggregation schemes, such as logistic, $\epsilon$-SVR, which are used to aggregate the similarity values from the matches on different parts of service description. COM4SWS \(^{15}\) is a hybrid matchmaker, which ranks the services based on numeric results of bipartite graph matching.

The comparison shows that different matchmakers use different matching strategies, which are designed based on assumptions about the conceptualization of Web service matchmaking. Usually, it is difficult to say which matchmaker is the best for a query, since the performance of matching strategies is often context- and data-dependent \([13]\). Facing so many matching strategies, a good matchmaker should support as many as possible matching strategies so that it can adaptively perform well in different applications, and allow users to customize the specific matching strategies for their requests according to their preference. SAWSDL-iMatcher is a such customizable matchmaker, which provides a general framework for Semantic Web service discovery, together with several effective matching strategies from syntactic, semantic to hybrid aggregated matching strategies.

6 Conclusion and Future Work

In this paper we have presented a customizable and effective Semantic Web service matchmaker: SAWSDL-iMatcher, which is based on iXQuery that extends XQuery with similarity joins. We have shown how iXQuery combines structured query with similarity joins to perform SAWSDL service matchmaking and how users can easily customize their preferred matching strategies in SAWSDL-iMatcher. Then, a relaxed semantic matching strategy has been proposed, and several statistical-model-based matching strategies have been built on top of it in SAWSDL-iMatcher. Various single matching strategies

\(^{15}\) http://www-ags.dfki.uni-sb.de/~klusch/s3/s3-2009-summary.pdf
and statistical-model-based matching strategies have been evaluated on two datasets. The evaluation has shown that it is difficult to determine a “winner” for SWS matchmaking, since different matching strategies have their strengths and weaknesses and are thus suitable for different tasks and contexts. Nevertheless, several empirical evidence have been observed when looking at the experimental results and the characteristics comparison of test datasets, which would be helpful for users to customize their requests. We observe that the semantic annotations of Web service operations are sufficient to describe the semantics of Web service, and syntactic matching strategies like Euclidean distance are suitable for measuring the similarity of semantic annotations effectively. Statistical-model-based matching strategies are good at aggregating the matching values on simple service description components, such as service name, description text, semantic annotations. Nevertheless, not all the statistical models are good for such task, and non-linear statistical models such as logistic and $\epsilon$-SVR seem to be better than linear statistical model for aggregating results from different single matching strategies. Moreover, logistic- and $\epsilon$-SVR-model-based matching strategies with selected features in this paper perform well.

Future work will focus on extensions of SAWSDL-iMatcher. First of all, several matching strategies will be built in SAWSDL-iMatcher, such as service structure- or XML-Schema-based matching strategies. We plan to investigate much more effective structure matching strategies and analyze their performance on more datasets. For this work, we need to construct a more comprehensive schema to represent all possible Semantic Web service descriptions, and the description extractor also needs to be extended. Second, we only consider the cost sensitive method to improve the quality of the statistical model at this stage, and the cost is set by empirical values. Therefore, we also plan to investigate training methods to get an improved statistical model by considering more appropriate weights for training data. Third, we plan to extend SAWSDL-iMatcher to support heterogeneous retrieval of different Semantic Web services, such as WSDL-S, OWL-S. Description extractors for these Semantic Web services also need to be investigated. Finally, we plan to investigate Web service matchmaking in specific domain. From our observation, different domains have different characteristics. The matching strategies could be improved by considering these specific characteristics.

Acknowledgements

This research is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302603, the Program for New Century Excellent Talents in University (NCET-06-0926), and the National Natural Science Foundation of China under Grant No. 60873097.
References


