OWLS-MX: A Hybrid Semantic Web Service Matchmaker for OWL-S Services

Matthias Klusch [a,1,*] Benedikt Fries [b] Katia Sycara [c,2]

[a] *German Research Center for Artificial Intelligence, Saarbruecken, Germany*
[b] *Morgan Stanley Japan Securities Corporation, Tokyo, Japan*
[c] *Carnegie Mellon University, Robotics Institute, Pittsburgh, USA*

Abstract

In this paper, we describe the first hybrid semantic Web service matchmaker for OWL-S services, called OWLS-MX. It complements crisp logic-based semantic matching of OWL-S services with token-based syntactic similarity measurements in case the former fails. The results of the experimental evaluation of OWLS-MX provide strong evidence for the claim that logic-based semantic matching of OWL-S services can be significantly improved by incorporating non-logic-based information retrieval techniques. An additional analysis of false positives and false negatives of the hybrid matching filters of OWLS-MX led to an even further improved matchmaker version called OWLS-MX2.

Key words: Semantic Web, OWL-S, Semantic Service Matching, Information Retrieval

1. Introduction

Semantic service discovery is the process of locating existing Web services based on the description of their functional and non-functional semantics. Discovery scenarios typically occur when one is trying to reuse an existing piece of functionality (represented as a Web service) in building new or enhanced business processes. Both service-oriented computing and the Semantic Web envision intelligent agents to proactively pursue this task on behalf of their users.

Central to the majority of contemporary approaches to Semantic Web service selection is that the functionality of Web services is logically defined in, for example, the standard first-order description logic-based ontology language OWL [6] or a rule language like SWRL, or a logic programming language like F-Logic. In anycase, intelligent agents can exploit standard means of logic reasoning to automatically understand the Web service semantics, in particular to determine the degree to which the service is semantically relevant to a given service request.

However, the representation of real-world semantics in logics only is known to be inadequate due to its limited expressivity. In addition, automated reasoning on Web service semantics expressed in first-order logics turned out not to be sufficiently scalable to the Web in practice [4].

One pragmatic solution to this problem is hy-

* Corresponding author. Phone: +49-681-302-5297
Email addresses: klusch@dfki.de (Matthias Klusch), benedikt.fries@morganstanley.com (Benedikt Fries), katia@cs.cmu.edu (Katia Sycara).

[1] Partial support provided by BMBF (German Ministry for Education and Research) grants MODEST 01-IWO-8001, SCALLOPS 01-IW-D02, European Commission grant CASCOM IST-FP6-511632.
[2] Partial support provided by the DARPA DAML program under contract F30601-00-2-0592.

brid semantic service selection, that is the combination of both logic-based and non-logic-based approximate reasoning on service semantics. Pioneering work in this direction include the first implemented hybrid semantic service matchmakers like LARKS [17], OWLS-MX, WSMO-MX [8], FC-MATCH [1] and OWLS-iMatcher2 [9].

In this paper, we describe our hybrid Semantic Web service matchmaker for OWL-S services, called OWLS-MX, in detail. Key to OWLS-MX is that it tolerates logical subsumption-based signature matching failures up to a specified extent by complementary approximate matching based on text similarity measurement. Of course, we acknowledge that the adaptation to the latter eventually is on the user's end.

The remainder of this paper is structured as follows. We provide background information on semantic services in OWL-S and semantic service selection with focus on logic-based approaches in sections 2 and 3, respectively. The following section 4 presents our approach to hybrid semantic service profile selection with OWLS-MX including its hybrid matching filters, the generic matching algorithm together with variants and a simple application example. Details on the implementation of OWLS-MX are given in section 5. The experimental results of measuring the service retrieval performance and scalability of OWLS-MX over a given test collection are presented in section 6, followed by an experimental analysis of its false positives and false negatives in section 7. These results led to an improved version of OWLS-MX reported in section 8. We briefly present related work on hybrid semantic service matchmakers in section 9, and conclude in section 10.

2. Semantic services in OWL-S

Our semantic service matchmaker OWLS-MX focusses on semantic services that are described in OWL-S. In the following, we briefly introduce the essentials of OWL-S, and refer to, for example, [15] for more details.

2.1. *Overview*

OWL-S is an upper ontology used to describe the semantics of services based on the W3C standard ontology OWL and is grounded in WSDL. It has its roots in the DAML Service Ontology (DAML-S) released in 2001, and became a W3C candidate recommendation in 2005. The OWL-S ontology consists of three main components: the *service profile* for advertising and discovering services; the *process model*, which gives a detailed description of a service's operation; and the *grounding*, which provides details on how to interoperate with a service, via messages.

In particular, the semantic service profile in OWL-S specifies the semantics of the service signature, that is the inputs required by the service and the outputs generated. Furthermore, since a service may require external conditions to be satisfied, and it has the effect of changing such conditions, the profile also describes the preconditions to be satisfied before, and the expected effects that result from the execution of the service. The majority of existing OWL-S service matchmakers focusses on semantic service profiles.

2.2. *OWL-S service profile*

The OWL-S profile ontology is used to describe what the service does, and is meant to be mainly used for the purpose of service discovery. An OWL-S service profile or signature encompasses its functional parameters, i.e. hasInput, hasOutput, precondition and effect (IOPEs), as well as non-functional parameters such as serviceName, serviceCategory, qualityRating, textDescription, and meta-data about the service provider such as name and location [3] .

Inputs and outputs relate to data channels, where data flows between processes. Preconditions specify facts of the world (state) that must be asserted in order for an agent to execute a service. Effects characterize facts that become asserted given a successful execution of the service in the physical world (state). Whereas, in OWL-S, the semantics of each service input and output parameter is defined in terms of a referenced OWL concept in a given ontology, typically in a decidable description logic OWL-DL or OWL-Lite, the preconditions and effects can be expressed in any appropriate first-order logic (rule) language such as KIF (Knowledge Interchange Format) or SWRL (Semantic Web Rule Language). Besides, the profile class can be subclassed and specialized, thus supporting the creation of profile taxonomies which subsequently describe different classes of services.

[3] Please note that, in contrast to OWL-S 1.0, in OWL-S 1.1 the service IOPE parameters are defined in the process model with unique references to these definitions from the profile.

2.3. *OWL-S service process model*

An OWL-S process model describes the composition (choreography and orchestration) of one or more subservices of a service, that is the controlled enactment of constituent processes with respective communication patterns, exposed IOPEs and parameter bindings of linked subservices. The semantics of OWL-S service process models have not been defined in the specification of OWL-S but various existing approaches to formalize the semantics of the standard Web service orchestration language BPEL (Business Process Execution Language) can be exploited for this purpose. Originally, the service process model was not intended for service discovery by the so-called OWL-S coalition, that is the group of researchers who developed OWL-S.

2.4. *OWL-S service grounding*

The grounding of an OWL-S service description provides a binding between the logic-based semantic service profile, the process model, and the XML-based Web service interface to facilitate service execution. Such a grounding of OWL-S services can be, in principle, arbitrary but has been exemplified for a grounding in WSDL (Web Service Description Language) to concretely connect OWL-S to an existing Web service standard.

In particular, the logic-based description of the service signature is uniquely associated with that of the Web service, and an atomic semantic process model is mapped to a WSDL operation. WSDL 1.0 does not allow to express pre-conditions or effects of services, nor has any formal semantics.

Though OWL-S allows only static and deterministic aspects of the world to describe in the description logic variants of OWL used for semantic annotation, the majority of semantic services available in the public Web happens to be in OWL-S [11]. Refactoring OWL-S to the standard for Semantic Web service description, SAWSDL (Semantically Annotated WSDL), is ongoing work in the Semantic Web services science community.

3. Semantic service selection

What is semantic service selection? Apart from finding available semantic services in the Web or central service directories, the quality of semantic service discovery depends on the process of semantic service selection: The pairwise semantic service matching of a set of semantic services with a given query and respective relevance-based ranking of the results returned to the user. Semantic service selection tools are also called semantic service matchmakers.

In the following, we classify existing Semantic Web service matchmakers, and focus on what most of them perform: Logic-based semantic service profile matching. Related work on hybrid semantic service matchmakers are discussed in section 9. For a more comprehensive survey, we refer to [10].

3.1. *Classification of SWS matchmakers*

Current semantic service matchmaker can be classified according to (a) what kinds and parts of service semantics are considered for matching, and (b) how matching is actually performed in terms of logic-based or non-logic-based reasoning within or partly outside the service description framework, or a combination of both (cf. figure 1).

The majority of them performs logic-based semantic service profile matching, and is restricted to OWL-S. Only a few are available for alternatives like WSML or the standard SAWSDL, and does also take process models and non-functional parameters into account. Though, process model-based matching was not intended by the designers of OWL-S or WSML, while neither SAWSDL nor monolithic service descriptions offer any process model element.

Logic-based semantic service matchmakers perform deductive reasoning on service semantics. In order to define these semantics, logical concepts and rules are referenced in ontologies as logical background theories. Different ontologies of service provider and requester are matched or aligned either at design time, or at runtime as part of the logic-based service matching process.

Non-logic-based semantic service matchmakers do not perform any logic-based reasoning to determine the degree of a semantic match between a given pair of service descriptions. Examples of non-logic-based semantic matching techniques are text similarity measurement, structured XML/RDF graph matching, and path-length-based similarity of concepts. In particular, service matchmakers that do not at least logically verify given semantic relations between ontological concepts used to describe service semantics classify as non-logic-based.

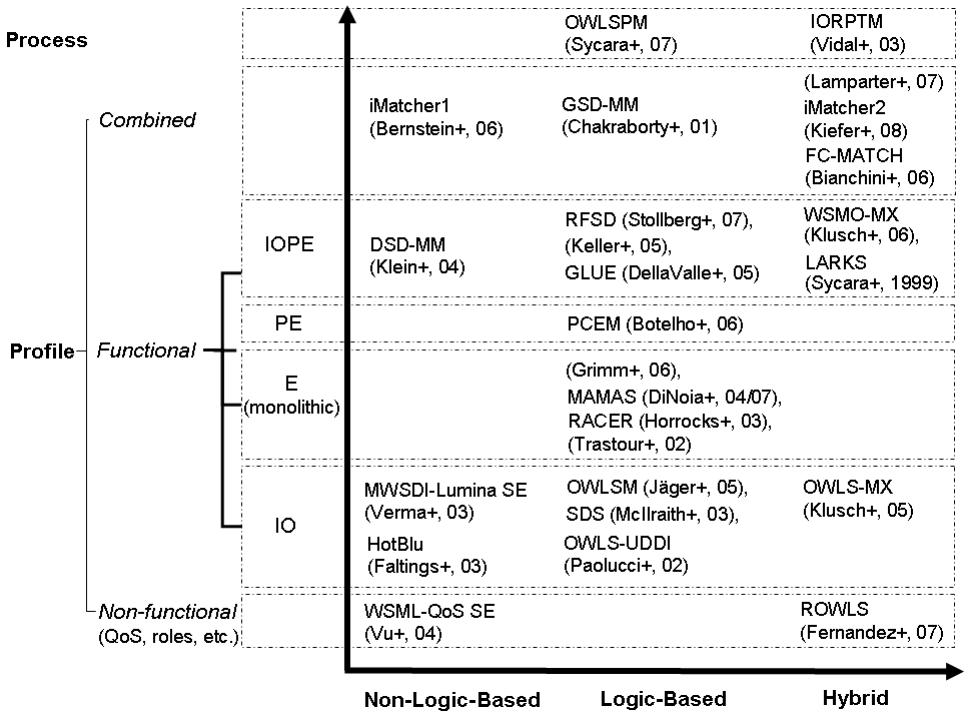

Fig. 1. Categories of existing SWS matchmakers.

3.2. *Logic-based semantic service profile matching*

We distinguish between monolithic and structured logic-based semantic profile matching. In the first case, the functionality of a Web service is exclusively - and agnostic to any parameterized description structure like in OWL-S or WSML - represented by a single logical expression. Semantic comparison of such monolithic logic-based service (effect) descriptions as a whole simply reduces to standard first-order (description) logic reasoning, that is service and query (desired service) concept subsumption, respectively, satisfiability checking completely within the logic theory.

For example, the logical post-plug-in match of an advertised service concept S with a service query concept R is determined by the entailment of service concept subsumption of S by R over a given knowledge base kb extended by the axioms of S and R: $kb \cup S \cup R \models S \sqsubseteq R$. That is, the matchmaker checks if in each first-order interpretation (possible world) I of kb, the set S^I of concrete provider services (service instances) is contained in the set R^I of service instances acceptable to the requester: $S^I \subseteq R^I$. In other words, service S is more specific than the request R, hence considered semantically relevant. A logical subsumes match assures the requester that her acceptable service instances are also acceptable to the provider: $kb \cup S \cup R \models R \sqsubseteq S$ [5]. Prominent examples of monolithic logic-based matchmakers are RACER [13] and MaMaS [4] [14].

Alternatively, structured logic-based profile matching makes additional use of parameterized service descriptions provided by most semantic service description languages such as OWL-S, WSML and SAWSDL. In this case, logic-based semantic profile matching is a combination of logical reasoning within the logic theory of the formal ontology language used for annotation, and algorithmic processing outside the theory. For example, a logic-based plug-in serice (IOPE) profile match requires to check that certain constraints hold on the type and quantification of computed (approximated) logical implications between preconditions and effect, and logical subsumption between input and output concepts (denoted as $in : C$, resp., $out : C$) of service and query [5] : Service S semantically IOPE-

[4] sisinflab.poliba.it/MAMAS-tng/

[5] Originally, a service S plugs into (plug-in matches with) another service R, if the effect of S is more specific than that of R, and vice versa for the preconditions of S and R [21]. Unfortunately, this notion of plug-in match in software

plugs into request R iff $prec_R \Rightarrow prec_S \wedge post_S \Rightarrow post_R$ (specification plug-in match), and $\forall in : C \in Input_S \; \exists in : C' \in Input_R : \; C \sqsubseteq C' \wedge \forall out : C \in Output_R \; \exists out : C' \in Output_S : \; C' \sqsubseteq C$ (signature plug-in match). Alternative for undecidable first-order logical entailment checking is the polynomial, correct but incomplete polynomial theta subsumption (a logical consequence relation) and instance-based query answer set inclusion (query containment).

In general, the complexity of computing logic-based semantic relations depends on the ontology language used for semantic annotation. For example, post-plug-in matching of service concepts in OWL-Full, that is SHOIQ$^+$ (including transitive non-primitive roles) has been shown to be undecidable, but decidable in NEXPTIME for OWL-DL, WSML-DL and DL-safe SWRL.

Examples of logic-based OWL-S service matchmakers are OWLSM [7] and OWLS-UDDI [16] both focussing on service IO-matching. The matchmaker PCEM [2] converts given OWL-S services and request to standard PDDL (Planning Domain Definition Language) actions and then computes logical implications between their preconditions and effects in PROLOG.

4. Hybrid semantic service profile matching

Hybrid semantic service selection performed by our matchmaker OWLS-MX exploits both logic-based reasoning and non-logic-based information retrieval (IR) techniques for OWL-S service profile signature matching. That is, OWLS-MX focusses on service I/O-parameter matching and ignores logical service specification in terms of preconditions and effects. Please note that the vast majority of accessible OWL-S services does not possess any such specification, nor any composite process model yet.

In the following, we define the hybrid semantic service filters of OWLS-MX, the generic OWLS-MX selection algorithm and its five variants according to five different text similarity metrics used by the matchmaker.

engineering has been adopted quite differently by most logic-based Semantic Web service matchmakers for both monolithic and structured service descriptions.

4.1. *Matching filters of OWLS-MX*

OWLS-MX computes the degree of semantic matching for a given pair of service advertisement and request by successively applying five different filters EXACT, PLUG IN, SUBSUMES, SUBSUMED-BY and NEAREST-NEIGHBOR. The first three are logic based only whereas the last two are hybrid due to the required additional computation of syntactic similarity values. Let be

- T the terminology (TBox) of the OWLS-MX matchmaker ontology specified in OWL-DL and CT_T the concept subsumption hierarchy of T;
- LSC(C) the set of least specific concepts (direct children) C' of C, i.e. C' is immediate sub-concept of C in CT_T;
- LGC(C) the set of least generic concepts (direct parents) C' of C, i.e., C' is immediate super-concept of C in CT_T;
- in:C $\in Input_S$ (out:C $\in Output_S$) an input (output) concept C of service S defined in T;
- SynSim(S,R) $\in [0, 1]$ real-valued degree of text similarity between service S and request R. This degree is computed as the averaged syntactic similarity of the serialized input, respectively, output concepts of S and R according to given text similarity metric SynSim(S,R): SynSim(S,R) = (SynSim(S,R)$_{in}$ + SynSim(S,R)$_{out}$)/2. Serialization of a set of concepts includes the terminological unfolding of these concepts in T, followed up by the conjunctive concatenation of the resulting logical expression and its preprocessing to one weighted keyword vector;
- $\alpha \in [0, 1]$ syntactic similarity threshold;
- $\sqsubseteq$ ($\equiv$) terminological concept subsumption (equivalence) relation.

The semantic service matching degrees computed by OWLS-MX are as follows.

Exact match. Service S EXACTLY matches request R $\Leftrightarrow \forall$ in:C $\in Input_S \; \exists$ in:C'$\in Input_R$: C $\equiv$ C' $\wedge \forall$ out:D$\in Output_R \; \exists$ out:D'$\in Output_S$: D $\equiv$ D OUT$_S$. The service I/O signature perfectly matches with the request with respect to logic-based equivalence of their formal semantics.

Plug-in match. Service S PLUGS INTO request R $\Leftrightarrow \forall$ in:C$\in Input_S \; \exists$ in:C'$\in Input_R$: C' *sqsubseteq* C $\wedge \forall$ out:D$\in Output_R \; \exists$ out:D'$\in Output_S$: D'$\in$ LSC(D). All service input parameter concepts are matched by a more specific

one in the request R. If the OWL input concept definitions can be mapped to equivalent WSDL input messages and service signature data types, this constraint guarantees at a minimum that S is executable with any input provided by the requestor. In addition, S is expected to return more specific output data whose logically defined semantics are exactly the same or very close to what has been requested.

This kind of match is borrowed from the software engineering domain, where software components are considered to plug-in match with each other as defined above but not restricting the output concepts to be direct children of those of the query. In particular, the definition of plug-in signature match used for OWLS-MX follows the original notion of software specification plug-in match introduced in [21].

Subsumes match. Request R SUBSUMES service S $\Leftrightarrow \forall$ in:C$\in$ $Input_S$ $\exists$ in:C'$\in$ $Input_R$: C' *sqsubseteq* C $\wedge \forall$ out:D$\in$ $Output_R$ $\exists$ out:D'$\in$ $Output_S$: D' *sqsubseteq* D. This filter is weaker than the plug-in filter in the sense that the returned service output is more specific than requested by the user: It relaxes the constraint of immediate output concept subsumption to arbitrary output concept subsumption.

Subsumed-by match. Request R is SUBSUMED BY service S $\Leftrightarrow \forall$ in:C$\in$ $Input_S$ $\exists$ in:C'$\in$ $Input_R$: C' *sqsubseteq* C $\wedge \forall$ out:D$\in$ $Output_R$ $\exists$ out:D'$\in$ $Output_S$: D' $\equiv$ D $\vee$ D'$\in$ LGC(D). This filter selects services whose output data is slightly more general than requested, hence, in this sense, subsumes the request. We focus on direct parent output concepts to avoid selecting services returning data which we think may be too general. Of course, it depends on the individual perspective taken by the user, the application domain, and the granularity of the underlying ontology at hand, whether a relaxation of this constraint is appropriate, or not.

Logical Fail. OWLS-MX returns a logic-based semantic matching failure degree, iff service S does not match with request R according to any of the above matching filters.

Hybrid subsumed-by match. Request R is SUBSUMED BY service S $\Leftrightarrow \forall$ in:C$\in$ $Input_S$ $\exists$ in:C'$\in$ $Input_R$: C' *sqsubseteq* C $\wedge \forall$ out:D$\in$ $Output_R$ $\exists$ out:D'$\in$ $Output_S$: (D' $\equiv$ D $\vee$ D'$\in$ LGC(D)) $\wedge$ SIM_{IR}(S, R) $\geq \alpha$. This hybrid filter complements logic-based subsumed-by matching with syntactic matching by means of a selected text similarity measurement.

Nearest-neighbor match. Service S is NEAREST NEIGHBOR of request R $\Leftrightarrow$ SIM_{IR}(S, R) $\geq \alpha$. This matching degree is non-logic-based since it checks the degree of text similarity between the input and output concepts of service and request. It is being applied only in case all of the above logic-based and hybrid matching filters fail.

Fail. OWLS-MX returns a total semantic matching failure degree as a result, iff service S does not match with request R according to any of the above matching degrees.

These service matching degrees are sorted according to the order of their semantic relevance degrees as follows: EXACT < PLUG-IN < SUBSUMES < SUBSUMED-BY < LOGICAL FAIL < HYBRID SUBSUMED-BY < NEAREST-NEIGHBOR < FAIL.

A service S that logically matches exactly with request R is assumed to be more semantically relevant to R than a plug-in matching one (EXACT < PLUG-IN). A subsumes match of S with R is considered semantically weaker than a plug-in match due to the relaxation of the service output concept matching condition (relaxation from direct child concept to any arbitrary subconcept in the matchmaker ontology). In other words, a plug-in matching service S is assumed to be semantically closer to R than a subsumes-matching service (PLUG-IN < SUBSUMES).

Further, we assume that a semantic service output concept which is more general than requested relaxes the degree of semantic relevance of this service to a query compared to a more specific service output. In particular, the restriction to direct parent concepts in the ontology in the logic-based subsumed-by matching condition makes a service S with a logical subsumes matching degree with focus on more specific, that is direct child concepts in the ontology, semantically more relevant to a given request R than services with more general output concepts like the subsumed-by-matching services (SUBSUMES < SUBSUMED-BY).

From the perspective of logic-based only semantic matching, the complementary text similarity measurement by the nearest-neighbour filter is con-

sidered weakest with respect to semantic relevance. This filter will only be performed in case all logic-based filters and the hybrid subsumed-by filter fail. Finally, only if none of the above sequentially checked matching degrees hold, the matchmaker returns a matching failure (NEAREST-NEIGHBOR < FAIL).

4.2. *Generic OWLS-MX matching algorithm*

The OWLS-MX matchmaker takes any OWL-S service as a query, and returns an ordered set of relevant services that match the query each of which is annotated with its individual degree of matching, and syntactic similarity value. The user can specify the desired degree, and syntactic similarity threshold. OWLS-MX then first classifies the service request I/O concepts into its local matchmaker ontology.

Matchmaker ontology. The matchmaker ontology emerges from a given initial ontology by classifying all service and request I/O concepts into this ontology each time a service advertisement or request is being received. We assume that service provider, requester and matchmaker share a basic minimal vocabulary of primitive components together with a set of mapping rules such as synonym relations in the thesaurus WordNet. Primitive components are terms out of which complex concepts are canonically defined in a description logic-based terminology.

Upon receipt of a service, the matchmaker focuses only on those parts of referenced service ontologies that are relevant to understand the semantics of the service. For this purpose, it terminologically unfolds each service input and output concept leading to logical concept expressions that include primitive components of a shared basic vocabulary. Each of these concept expressions is self-contained in the sense that the rest of the referenced service ontology is not necessary to understand the semantics of the unfolded concept.

Attached to each concept in the matchmaker ontology are auxiliary data about which registered services are using that concept as an input and/or output concept. The respective lists of service identifiers are used by the matchmaker to compute the set of relevant services that are matching with the given query.

Hybrid matching. Any failure of logical concept subsumption produced by the integrated description logic reasoner of OWLS-MX will be tolerated, if and only if the degree of syntactic similarity between the respective unfolded service and request concept expressions exceeds a given similarity threshold.

The pseudo-code of the generic OWLS-MX matching process is given below (cf. algorithms 1 - 3). Let $\text{INPUTS}_S = \{ \text{IN}_{S,i} | 0 \leq i \leq s \}$, $\text{INPUTS}_R = \{ \text{IN}_{R,j} | 0 \leq j \leq n \}$, $\text{OUTPUTS}_S = \{ \text{OUT}_{S,k} | 0 \leq k \leq r \}$, $\text{OUTPUTS}_R = \{ \text{OUT}_{R,t} | 0 \leq t \leq m \}$, set of input and output concepts used in the profile I/O parameters hasInput and hasOutput of registered service S in the set $Advertisements$, and the service request R, respectively. Attached to each concept in the matchmaker ontology are auxiliary data that informs about which registered service is using this concept as an input and/or output concept.

Algorithm 1 *Match*: Find advertised services S that best match in a hybrid fashion with a given request R; returns set of $(S, degreeOfMatch, SIM_{IR}(R,S))$ with maximum degree of match (*dom*) unequal FAIL (uses algs. 2 and 3 to compute *dom*), and syntactic similarity value exceeding a given threshold α.

```
1:  function MATCH(Request R, α)
2:      local result, degreeOfMatch,
            hybridFilters = { SUBSUMED-BY,
            NEAREST NEIGHBOUR}
3:      for all (S, dom) ∈ CANDIDATES_inputset(INPUTS_R) ∧ (S, dom') ∈
            CANDIDATES_outputset(OUTPUTS_R) do
4:          degreeOfMatch ← MIN(dom, dom')
5:          if degreeOfMatch ≥ minDegree ∧ (
                degreeOfMatch ∉ hybridFilters ∨
                SIM_IR(R,S) ≥ α) then
6:              result := result ∪ { (S,
                    degreeOfMatch, SIM_IR(R,S) ) }
7:          end if
8:      end for
9:      return result
10: end function
```

In the following section, we present five variants of this generic OWLS-MX matchmaking scheme.

4.3. *OWLS-MX variants*

We implemented different variants of the generic OWLS-MX algorithm, called OWLS-M1 to OWLS-M4, each of which uses the same logic-based semantic filters but different IR similarity metric

Algorithm 2 Find services which *input* matches with that of the request; returns set of (S, dom) with minimum degree of match *dom* unequal FAIL.

1: **function** CANDIDATES$_{inputset}$(INPUTS$_R$)
2: **local** H, dom, r
3: ▷ *If a service input matches with multiple request inputs the best degree is returned*
4: $H := \{ (S, \text{IN}_{S,i}, dom) \in \bigcup_{j=1..n} \text{CANDIDATES}_{input}(\text{IN}_{R_j}) \mid dom = argmax_l\{ (S, \text{IN}_{S,i}, dom_l) \mid 1 \leq l \leq n, 1 \leq i \leq s\} \}$
5: ▷ *If all inputs of service S are matched by those of the request, S can be executed, and the minimum degree of its potential match is returned*
6: **for all** $S \in Advertisement$ **do**
7: **if** $\{ (S, \text{IN}_{S_1}, dom_1), \cdots, (S, \text{IN}_{S_s}, dom_s) \} \subseteq H$ **then**
8: $r := r \cup \{ (S, \text{MIN}(dom_1, \cdots, dom_s)) \}$
9: **end if**
10: **end for**
11: ▷ *Services with no input can always be executed and are preliminary* EXACT*-match candidates:* SERVNOIN() = { *(S,* EXACT*)* | $S \in Advertisements \wedge \text{INPUTS}_S = \emptyset$ }
12: ▷ *Remaining, unmatched services are at least* NEAREST NEIGHBOUR*-match candidates:* REMSERV() = { *(S,* NEAREST NEIGHBOUR*)* | $S \in Advertisements \wedge \langle S, degreeOfMatch' \rangle \notin r$ }
13: **return** $r := r \cup \text{SERVNOIN}() \cup \text{REMSERV}()$
14: **end function**
15:
16: **function** CANDIDATES$_{input}$(IN$_{R,j}$) ▷ *Classify request input concept into ontology, and use the auxiliary concept data to collect services that at least plug-in match with respect to its input.*
17: **local** r
18: $r := r \cup \{ (S, \text{IN}_S, \text{EXACT}) \mid S \in Advertisements, \text{IN}_S \in inputs_S, \text{IN}_S \doteq \text{IN}_{R,j}, \}$
19: $r := r \cup \{ (S, \text{IN}_S, \text{PLUG-IN}) \mid S \in Advertisements, \text{IN}_S \in inputs_S, \text{IN}_S \dot{\geq} \text{IN}_{R,j}, \}$
20: **return** r
21: **end function**

$\text{SIM}_{IR}(R, S)$ for content-based service I/O matching. The variant OWLS-M0 performs logic based only semantic service I/O matching.

Algorithm 3 Find services which *output* matches with that of the request; returns set of (S, dom) with minimum degree of match unequal FAIL.

1: **function** CANDIDATES$_{outputset}$(OUTPUTS$_R$)
2: **local** r, dom
3: **if** $\text{OUTPUTS}_R = \emptyset$ **then**
4: **return** { $(S,$ EXACT$)$ | $S \in Advertisements$ }
5: **end if**
6: **for all** $S \in Advertisements$ **do**
7: **if** $(S, dom_t) \in \text{CANDIDATES}_{output}(\text{OUT}_{R,t}) \wedge dom_t \geq$ SUBSUMES **for** $t = 1..m$ **then**
8: $r := r \cup \{ (S, \text{MIN}\{dom_1, \cdots, dom_m\}) \}$
9: **else if** $(S, dom_t) \in \text{CANDIDATES}_{output}(\text{OUT}_{R,t}) \wedge dom_t \in$ { EXACT, SUBSUMES } **for** $t = 1..m$ **then**
10: $r := r \cup \{ (S,$ SUBSUMED-BY $\}$
11: **end if**
12: **end for**
13: ▷ *Any remaining, unmatched service is a potential* NEAREST NEIGHBOUR*-match:* REMSERV() = { *(S,* NEAREST NEIGHBOUR*)* | $S \in Advertisements \wedge S \notin r$ }
14: **return** $r := r \cup \text{REMSERV}()$
15: **end function**
16:
17: **function** CANDIDATES$_{output}$(OUT$_{R,t}$) ▷ *Classify request output concept into ontology, and use the auxiliary concept data to collect services with output concepts that match with* OUT$_{R,t}$.
18: **local** r
19: $r := r \cup \{ (S,$ EXACT$) \mid \text{OUT}_S \doteq \text{OUT}_{R,t} \}$
20: $r := r \cup \{ (S,$ PLUG-IN$) \mid \text{OUT}_S \in \text{LSC}(\text{OUT}_{R,t}) \wedge S \notin r \}$
21: $r := r \cup \{ (S,$ SUBSUMES$) \mid \text{OUT}_S \dot{\leq} \text{OUT}_{R,t} \wedge S \notin r \}$
22: $r := r \cup \{ (S,$ SUBSUMED-BY$) \mid \text{OUT}_S \in \text{LGC}(\text{OUT}_{R,t}) \}$
23: **return** r
24: **end function**

OWLS-M0. The logic-based semantic filters EXACT, PLUG-IN, SUBSUMES and SUBSUMED-BY are applied as defined in section 3.1.

OWLS-M1 to OWLS-M4. The hybrid variants OWLS-M1, OWLS-M3, and OWLS-M4 also compute the syntactic text similarity value SIM_{IR}

(OUT_S, OUT_R) by use of the loss-of-information measure, extended Jacquard similarity coefficient, the cosine similarity value, and the Jensen-Shannon information divergence based similarity value, respectively.

Based on the experimental results of measuring the performance of similarity metrics for text information retrieval provided by Cohen and his colleagues [3], we selected the top performing ones to build the OWLS-MX variants. These symmetric token-based string similarity measures are defined as follows.

- The *cosine* similarity metric $Sim_{Cos}(S,R) = \frac{\vec{R}\cdot\vec{S}}{||\vec{R}||_2^2 \cdot ||\vec{S}||_2^2}$ with standard TFIDF term weighting scheme, and the unfolded concept expressions of request R and service S are represented as n-dimensional weighted index term vectors $\vec{R}$ and $\vec{S}$ respectively. $\vec{R}\cdot\vec{S} = \sum_{i=1}^{n} w_{i,R} \times w_{i,S}$, $||X||_2 = \sqrt{\sum_i^n w_{i,X}^2}$, and $w_{i,X}$ denotes the weight of the i-th index term in vector X.

- The *extended Jaccard* similarity metric $Sim_{EJ}(S,R) = \frac{\vec{R}\cdot\vec{S}}{||\vec{R}||_2^2 + ||\vec{S}||_2^2 - \vec{R}\cdot\vec{S}}$ with standard TFIDF term weighting scheme.

- The intensional *loss of information* based similarity metric $Sim_{LOI}(S,R) = 1 - \frac{LOI_{IN}(R,S) + LOI_{OUT}(R,S)}{2}$ with $LOI_x(R,S) = \frac{|PC_{R,x} \cup PC_{S,x}| - |PC_{R,x} \cap PC_{S,x}|}{|PC_{R,x}| + |PC_{S,x}|}$, $x \in \{IN, OUT\}$, $PC_{R,x}$ and $PC_{S,x}$ set of primitive components in unfolded logical input/output concept expression of request R and service S.

- The *Jensen-Shannon information divergence* based similarity measure $Sim_{JS}(S,R) = log2 - JS(S,R) = \frac{1}{2log2} \sum_{i=1}^{n} h(p_{i,R}) + h(p_{i,S}) - h(p_{i,R} + p_{i,S})$ with probability term frequency weigthing scheme, *e.g.*, $p_{i,R}$ denotes the probability of i-th index term occurrence in request R, and $h(x) = -xlogx$.

The extended Jaccard metric is a standard for measuring the degree of overlap as the ratio of the number of shared terms (primitive components) of unfolded concepts of both service and request, and the number of terms possessed by either of them. In contrast to the TFIDF/cosine similarity metric, it does not favor documents with most common terms. The Jensen-Shannon measure is based on the information-theoretic, non-symmetrical Kullback-Leibler divergence measure. It measures the pairwise dissimilarity of conditional probability term distributions between service and request text rather than looking at the whole collection as it is the case for the TFIDF/cosine, or the extended Jaccard metric. The loss of (intensional) information in case some concept A is terminologically substituted by concept B, can be measured as the inverse ratio of the number of matching primitive components with those which remain unmatched in terminologically disjoint unfolded concept constraints. The symmetric LOI-based similarity value for a given pair of service and request is then computed analogously for all I/O-concept definitions involved.

4.4. *Example*

Let us illustrate the hybrid service matching with OWLS-MX by means of a simple example. Figure 2 shows the concept subsumption hierarchy or taxonomy of the OWLS-MX matchmaker ontology, the service request R for physicians of some hospital h that provide treatment to patient p, and relevant service advertisements S_1 and S_2.

Service S_1 is considered semantically relevant to request R, since it returns for any given person p and hospital h, the individual surgeon of h that operated on p. Likewise, service S_2 is relevant to R, since it returns those emergency physicians who provided emergency treatment to p before her transport to hospital h. Hence, both services S_1 and S_2 should be returned as matching results to the user.

However, the logic-based only variant OWLS-M0 determines S_1 as plug-in matching with R but fails to return S_2, since the logic-based semantics of the output concept siblings "emergency physician" and "hospital physician" in the ontology are terminologically disjoint. Please note that this concept disjointness is not defined in any of the concepts but has been computed by the matchmaker in due course of its classifying these concepts into its matchmaker ontology. In this example, the set of terminological constraints of unfolded concepts c corresponds to the set of primitive components (c^p) of which the individual concepts are canonically defined in the matchmaker ontology T. Hence, the unfolded concept expressions are as follows.

- unfolded(Patient, T) = (and Patient^p Person^p)

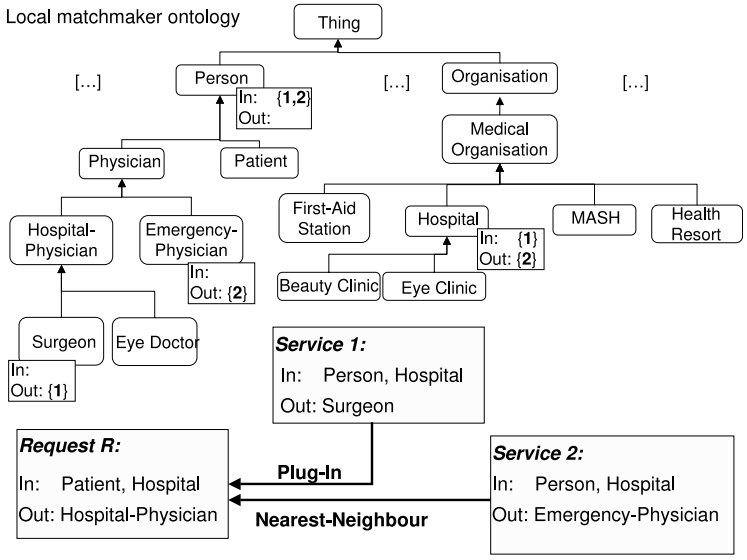

Fig. 2. Example of hybrid service matching with OWLS-MX

- unfolded(Hospital, T) = (and Hospital^p (and MedicalOrganisation^p Organisation^p))
- unfolded(HospitalPhysician, T) = (and HospitalPhysician^p (and Physician^p Person^p))
- unfolded(Surgeon, T) = (and Surgeon^p (and HospitalPhysician^p (and Physician^p Person^p)))
- unfolded(EmergencyPhysician, T) = (and EmergencyPhysician^p (and Physician^p Person^p))

As a result, for example, OWLS-M1 would return S_1 as semantically plug-in matching service with syntactic similarity value of Sim_{LOI} $(R, S_1) = 0.87$. In contrast to OWLS-M0, it also returns S_2, since this service is nearest-neighbor matching with the request R: Their implicit semantics exploited by the text similarity metric LOI (cf. (3), (4)) with Sim_{LOI} $(R, S_2) = \frac{(1-\frac{5-4}{5+4})+(1-\frac{4-2}{3+3})}{2} = 0.78 \geq \alpha = 0.7$ is sufficiently similar, that is, it exceeds a threshold given by the requester.

5. Implementation

We implemented the OWLS-MX matchmaker variants (current version 1.1c) in Java using the OWL-S API 1.1 beta with the tableaux OWL-DL reasoner Pellet developed at university of Maryland (cf. `http://www.mindswap.org`). As the OWL-S API is tightly coupled with the Jena Semantic Web Framework, developed by the HP Labs Semantic Web research group (cf. `http://jena.sourceforge.net/`), the latter is also used to modify the OWLS-MX matchmaker ontology. Figure 3 shows a screenshot of the OWLS-MX version 1.1 graphical user interface.

After parsing service advertisements and requests, the respective input and output concepts are analysed and, if necessary, added to the local matchmaker ontology together with auxiliary data on their unfolding. As a consequence, the matchmaker ontology is dynamically built and growing with the number of services and underlying ontologies loaded. In addition, the matchmaker ontology is extended with auxiliary information for each concept, for example whether it is used as an input or output concept of a service registered at the matchmaker. Service requests are treated similarly, except that they are not stored in the extended matchmaker ontology.

For each service request concept, the service identifiers attached to its immediate parent and child concepts of the enhanced matchmaker ontology are retrieved. The semantic degree of matching for each service is then determined by applying the semantic filters on this set of matching candidates. After this step, the syntactic similarity is computed by applying the selected IR similarity metric to the strings of unfolded concepts of the query and each registered service. Both the semantic degree of match and the syntactic similarity value determine the hybrid de-

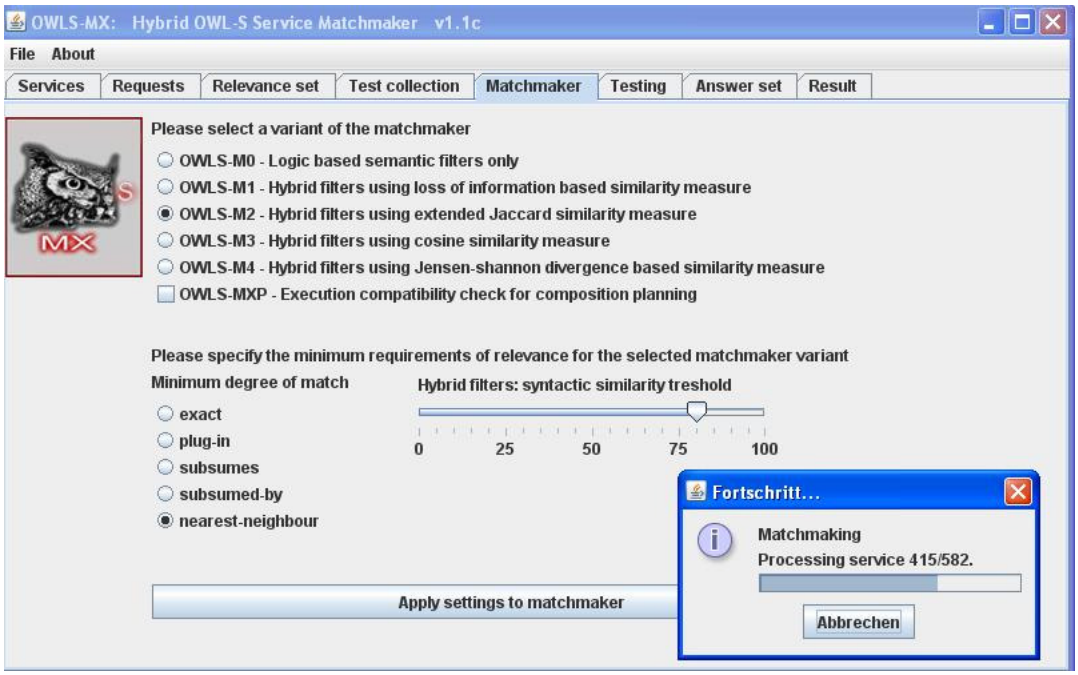

Fig. 3. OWLS-MX v1.1c screenshot: OWLS-MX configuration

gree of matching of one service with the request. If this hybrid degree is better than or equal to the minimum degree specified by the user, then this service will be returned as potentially relevant.

OWLS-MX spends the largest amount of time with classifying the service I/O-concept related parts of OWL-DL ontologies used by newly registered services into the matchmaker ontology. That is, it classifies new service I/O concepts not yet known to the matchmaker into its current ontology (see section 4.2, matchmaker ontology). For example, the processing of 582 services of the test collection OWLS-TC 2.1 takes about ten minutes (on an IBM ThinkPad T41p with 1.7GHz and 2GB RAM) presuming that there are no additional timeouts due to unavailability of service related OWL ontologies at remote sites. Once this preprocessing has been completed, the average query response time of OWLS-MX appears reasonable but probably not acceptable in practice (about 10 secs per query to check over 582 services). There is definitely space for improving on that by means of applying appropriate caching and indexing techniques.

OWLS-MX v1.1c comes with an integrated evaluation tool for measuring its peformance in terms of precision and recall over a given OWL-S service retrieval test collection. Alternatively, we developed the SME2 (Semantic MatchMaker Evaluation Environment) tool which provides more functionality for testing arbitrary Semantic Web service matchmakers for OWL-S, WSML and SAWSDL; the evaluation tool SME2 is available at projects.semwebcentral.org/projects/sme2/.

6. Evaluation of Performance

In this section, we provide our experimental results of the retrieval performance of logic-based and hybrid OWLS-MX variants in terms of recall and precision.

6.1. *Service retrieval test collection*

For measuring the service I/O retrieval performance of each OWLS-MX variant, we used the OWL-S service retrieval test collection Owls-TC v2.2. This collection consists of more than 1000 services specified in OWL-S 1.1 in seven application domains, that are education, medical care, food, travel, communication, economy, and weaponry. The majority of these services were retrieved from public IBM UDDI registries, and semi-automatically transformed from WSDL to OWL-S. Owls-TC v2.2 provides a set of 28 test queries each of which is associated with a set of 10

to 20 services that a dozen people subjectively defined as relevant according to the standard TREC definition of binary relevance [19]. The collection OWLS-TC v2.2 is available as open source at projects.semwebcentral.org/projects/owls-tc/. We are working on OWLS-TC v2.2G with graded relevance sets.

6.2. *Overall R/P performance*

We adopted the evaluation strategy of macro-averaging the individual precision values over all requests $q \in Q$ for λ recall levels [20] of each OWLS-MX variant over the test collection OWLS-TC 2.1. The matchmaker returns a rank list of all services for evaluation, that is the answer set for evaluation purposes is the set S of registered services. For all queries $q_i \in Q, i \in \{1..n\}$ and recall levels $\lambda_j = j/\lambda \in [0,1], j = 1..\lambda$, we select the precision $Prec_i(r)$ value that is maximum (ceiling interpolation) for recall $Rec_i(r) \geq \lambda_j$ at some rank $r = 1, ..., |S|$. Finally, we average these observed precision values to obtain the macro-averaged precision $Prec(\lambda_j)$ (over all queries) at each recall level λ_j.

In summary, the evaluation results showed that hybrid semantic matching can improve logic-based only service selection in terms of both precision and recall (cf. figure 4). While OWLS-M0 reached precision of 0.67 and recall of 0.50 in average for its top-20 ranked services, the hybrid OWLS-M3 achieved that with a higher precision (0.74) and recall (0.557).

The reason of higher precision of the hybrid variants OWLS-M1 to OWLS-M4 is that they avoided most of logic-based false positives in case of logical subsumed-by matches in the given test collection by complementary syntactic similarity measurement. In addition, the hybrid semantic matchmakers avoided logic-based false negatives caused by wrongly returned matching degree of logical fail through complementary syntactic similarity measurements (nearest-neighbour match) which led to a better recall. All hybrid variants showed almost equal performance in average. In the following, we show the main cases of logic-based and hybrid false positives and false negatives lowering precision, respectively, recall of OWLS-MX.

The quantitative impact of the above mentioned false positive and false negative cases on the matchmaker performance, of course, depends on the used test collection. In this respect, our experimental evaluation results are preliminary as long as there is no (quasi-)standard collection for semantic service retrieval available, similar to TREC in the IR domain.

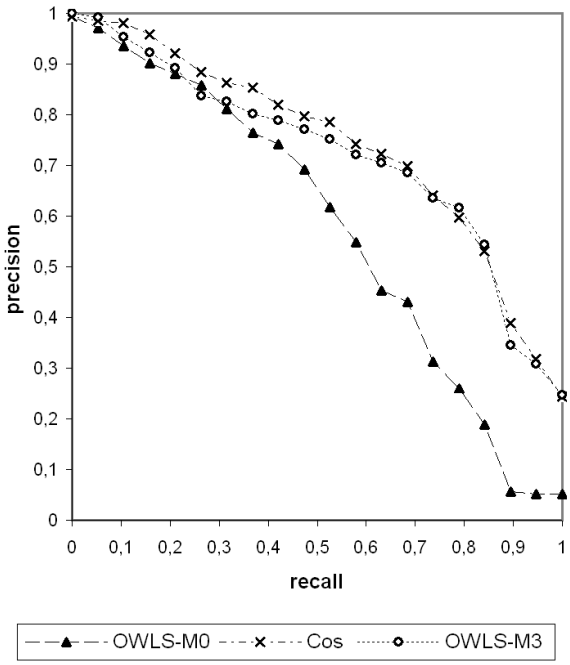

Fig. 4. R/P performance of logic-based OWLS-M0 vs. syntactic matching (Cosine/TFIDF, threshold 0.6) vs. hybrid matching with OWLS-M3.

7. Anaylsis of false positives and negatives

In this section, we analyze the retrieval performance of OWLS-MX in terms of false positive and false negatives to reveal the benefits and pitfalls of its logic-based and hybrid semantic matching filters.

7.1. *Logic-based false positives*

There are two main reasons for logic-based false positives of OWLS-MX: First, in the context of service matching, the known logical mismatch problem of knowledge representation is manifested by inappropriate logical definitions of input or output concepts used to define the semantics of I/O concepts of services in the logic-based matchmaker ontology. Second, the all-quantified logical matching constraints wrongly tolerates the missing of input or output concepts. These types of logic-based false positives of OWLS-M0 are illustrated by example

in the following.

Granularity of matchmaker ontology. Any logic-based semantic service matchmaker risks to return false positives if the given logical concept definitions in its ontology are not capturing the real-world semantics of the concepts used to defined the service semantics. This kind of logical mismatch is a general problem of symbolic knowledge representation. In the context of semantic service matching, the decision whether some service is a false positive for a given query is subjective for each individual user. The same holds for the definition of relevance sets for each query in the test collection OWLS-TC2.2 we used to evaluate the retrieval performance of OWLS-MX.

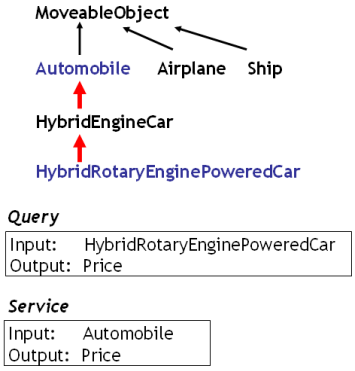

Fig. 5. Example: False positive due to tolerated unlimited logical parent-child relation between input concepts.

For example, in figure 5, the service at best logically plug-in matches with the query, since the (equally named) output concepts "price" are determined logically equivalent, and the query input concept "HybridRotaryEnginePoweredCar" is far more specific than the service input concept "Automobile". According to the developers of the test collection, the real-world semantic distance between both input concepts in the matchmaker ontology can be considered too large for being of any interest which renders the service irrelevant. The reason why all logic-based matching filters of OWLS-M0 fail to reckognize this, hence return the service as relevant, is that they accept an unlimited input concept distance in the matchmaker ontology.

Similarly, in the second example (cf. figure 6) the logical comparison of service and query output definitions result in a direct subsumption relation in the matchmaker ontology which could be (subjectively) considered wrong, hence produce a false positive.

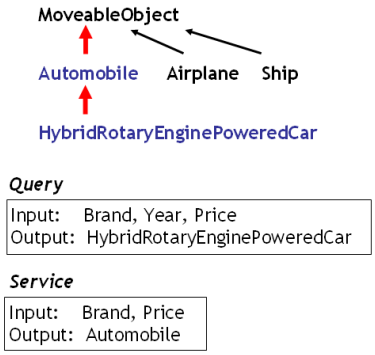

Fig. 6. Example: False positive due to logical mismatch of output concepts with direct parent-child relation.

In this case even a restrictive least generic concept match of the logical subsumed-by filter of OWLS-M0 does not help to avoid this. However, in both cases the additional syntactic matching of the hybrid subsumed-by matching filter of OWLS-M1 to OWLS-M4 can potentially avoid logical subsumed-by matches that are classified as false positives. This holds under the IR assumption that the degree of syntactic similarity sufficiently corresponds with the degree of real-world semantic similarity.

All-quantified logical matching constraints. Many false positives of OWLS-M0 are caused by the restrictive all-quantified logical matching constraints. Since the hybrid variants inherit the decision of OWLS-M0 in case of a logical match, these become false positives of OWLS-M1 to OWLS-M4 too.

Query input without corresponding service input. The surjective mapping of service input concepts to query input concepts ($\forall$ IN$_S$ $\exists$ IN$_R$) can lead to false positives: It tolerates the missing of service input concepts that correspond to those query input concepts that are important part of or even key for defining the intended query semantics. For example, in figure 7, the input "SFNovel" of the query "SFNovelPrice" does not match with any input of the service "EntranceFee" but "Author" with "Person". As a result, OWLS-M0 determines a plug-in match, hence wrongly returns the service as relevant.

Even worse, the surjective concept mapping by OWLS-MX can lead to false logical exact matches in case of no input or output concepts provided. For example, in figure 8, the query "BuyBook" and service "DatingService" are returned as semantically

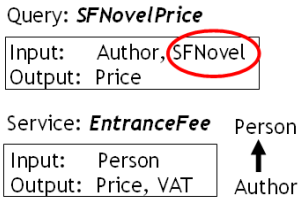

Fig. 7. Example: False positive due to all-quantified matching. Incomplete coverage of query input by service input is tolerated.

equivalent by OWLS-M0. The reason is that in this particular case there does not exist any query output concept which matches with the existing service output concept. Similarly, the same holds for the query "RoutingService" and the service "Map" without any input concept to match which makes the service-centred input matching constraints of all logical filters of OWLS-MX true by default.

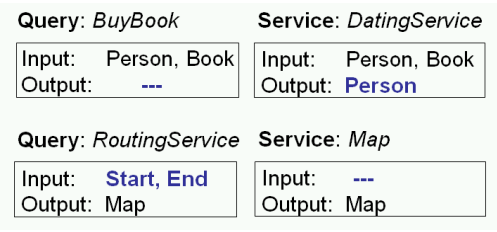

Fig. 8. Example: False positives due to tolerated lack of service or query I/O.

Same I/O concepts used to express different query and service semantics. Besides, this surjective matching of I/O concepts by all logical filters of OWLS-MX ignores the possible use of the same concept to describe different real-world semantics of a query or a service. For example, the real-world semantics of service "BookCopyCheck" and query "BookReview" in figure 9 are assumed to be not related at all, that is the service is not relevant to the query.

However, OWLS-M0 classifies the service as semantically equivalent with the query, hence produces a false positive. The same concept "Book" is used twice in the service input but with obviously different real-world semantics than in the query. In these cases, even syntactic similarity measurement would return a high relevance degree but at least not identity between service and query, since the term unfolded concept "Book" can be detected as a surplus term of the input string by means of fine-grained syntactic overlap measurement like the extended Jaccard coefficient.

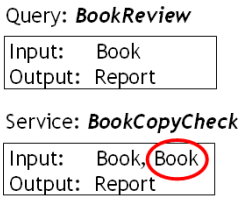

Fig. 9. Example: False positive caused by using the same concept "book" for describing different service and query semantics

7.2. *Avoiding logic-based false positives*

The hybrid variants OWLS-M1 to OWLS-M4 can increase their precision compared to OWLS-M0 by avoiding its false logic-based subsumed-by matches through additional syntactic similarity measurement. In fact, the hybrid subsumed-by filter allows to detect the irrelevance of a service S that logically subsumes the query R with insufficient syntactic similarity value (SynSim(S,R)$\leq \alpha$).

Hybrid false positives. However, due to sequential execution of ordered logic-based and hybrid matching filters, the hybrid filters inherit the remaining logic-based false positives from OWLS-M0. That can be avoided by complementary syntactic matching for all logic-based filters (except the exact match) which led to the development of OWLS-MX2 (cf. section 8).

Syntactic false positives only. On the other hand, the complementary syntactic matching can also cause hybrid false positives in case of sufficient syntactic similarity but non-matching real-world semantics between service and request which would be correctly determined by OWLS-M0 (logical matching failure). For example, logical connectives like "and", "or" in the logically unfolded concept expressions are ignored by syntactic matching, since they are eliminated as classical stop-words in the preprocessing step of unfolded service and query I/O concept expressions to weighted keyword vectors for text similarity measurements (SynSim(S,R)$_{out}$; SynSim(S,R)$_{in}$).

For example, in figure 10 we are asking for a service that is capable of either colouring or framing a given picture, and consider a service that is restricted to jointly perform both actions irrelevant. In this case, the logic-based OWLS-M0 correctly returns a logical matching failure while OWLS-M1 to OWLS-M4 ignore the subtle difference between the

use of the terms "and" and "or" in the concept expressions, hence detect a syntactic exact match of both pairs of I/O strings and return the service as relevant with a hybrid nearest-neighbour matching degree (text similarity value of 1.0).

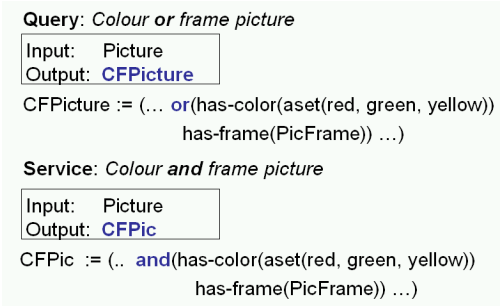

Fig. 10. Example: Hybrid false positives due to ignorance of logical connectives by complementary syntactic matching.

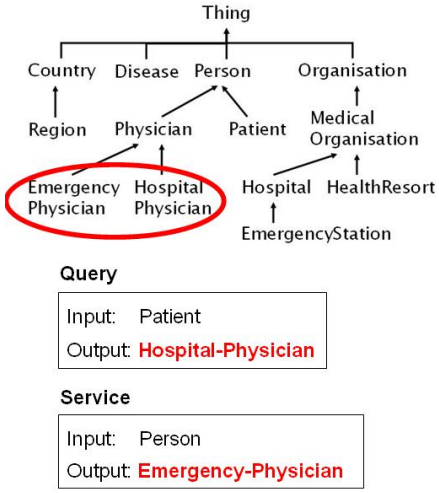

Fig. 11. Example: False negative caused by semantically similar but (not defined as) logically disjoint concept siblings in the matchmaker ontology.

7.3. *Logic-based false negatives*

Like for logic-based false positives, the reasons of logic-based false negatives are mainly due to the logical mismatch problem of the matchmaker ontology and the implication of the all-quantified matching constraints of OWLS-MX.

Ontology granularity: Similar concept siblings with logical disjoint definitions. The problem of logical mismatches due to insufficient ontology modeling can also cause false negatives, that are services wrongly classified as irrelevant by OWLS-M0. One example of such logic-based only false negatives is the case of logically disjoint concept siblings with similar real-world semantics in a fine-grained ontology. Please note that these concepts are not explicitly defined disjoint in the ontology but determined to be disjoint by the matchmaker while matching the service with the query. For example, in figure 11, the query output "Hopital-Physician" and service output "Emergency-Physician" are assumed to be semantically close such that the service is considered relevant to the query. However, the matchmaker classifies both conjunctive concept definitions differing in only one pair of their (equally weighted) logical constraints as logically disjoint, hence produces a false negative.

All-quantified logical matching constraints: More generic service input only. The all-quantified matching filters of OWLS-M0 require that the service input must be logically more generic than or equal to the query input. In case of a linear mapping of service and query I/O concepts to corresponding XMLS signature data types on the service grounding level, this guarantees that the WSDL service can be invoked with the information specified in the query by the user, in principle. However, this can lead to false negatives as shown in figure 12, since no logical filter of OWLS-M0 evaluates to true in cases where the service input is more specific than requested.

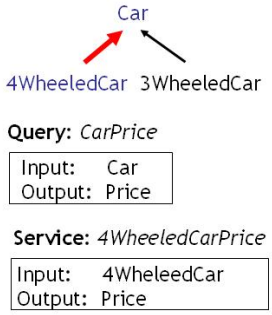

Fig. 12. Example: False negative due to required genericity of service input.

All-quantified logical matching constraints: Logical concept relations of same type. Finally, the matching filters of OWLS-M0 require each pair of service and query I/O concepts having the same type of logical subsumption relation. For example, in figure 13, the logical subsumption relations between output concepts of query "CarPlusBike" and service

"4WheeledCarPackage" are different. As a result, OWLS-M0 fails to detect the service as relevant.

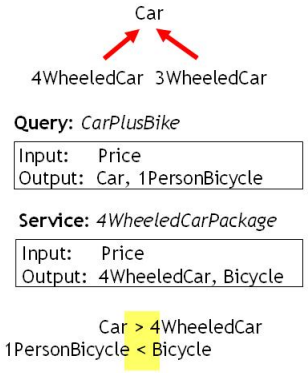

Fig. 13. Example: False negative due to different concept subsumption relations not accepted by the all-quantified filter constraints of OWLS-MX

7.4. *Avoiding logic-based false negatives*

Due to complementary syntactic matching in case of logical matching failure returned by OWLS-M0, the hybrid variants OWLS-M1 to OWLS-M4 can avoid the above cases of logic-based false negatives, thereby increasing their recall compared to OWLS-M0. This is achieved by detecting (hybrid) nearest-neighbour matches, if the degree of syntactic similarity between the considered pairs of concepts or service and query I/O-signature as a whole is sufficient.

8. OWLS-MX2

The version OWLS-MX2 integrates syntactic similarity-based matching with logic-based subsumes and plug-in matching like the hybrid subsumed-by filter in OWLS-MX. That avoids false-positives the hybrid OWLS-M1 to OWLS-M4 inherit from OWLS-M0. Our experiments over the OWLS-TC 2.2 that contains cases for all of the above mentioned false positives and false negatives showed that OWLS-MX2 did outperform OWLS-MX for this reason, and performed slightly better than text IR by avoiding syntactic similarity-based only false positives.

Since the number of cases for false positives of text IR in the collection OWLS-TC 2.2 is still significantly less than those for logic-based false positives, the hybrid OWLS-M3 did not outperform syntactic matching only. In fact, we observed that in most

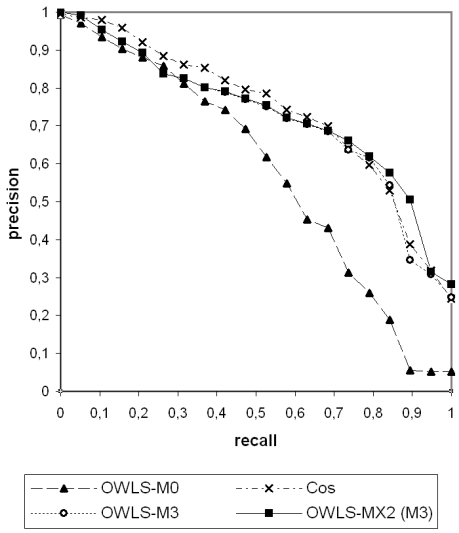

Fig. 14. R/P performance of hybrid OWLS-MX2, OWLS-M3, and IR metric cosine/TFIDF.

domain ontologies fine-grained logical concept definitions other than pure subclass relations are rare in practice which still handicaps logic-based only semanric service matching in practice. However, the quantitative relation between logic-based and text IR only false positives and false negatives in the Semantic Web is unknown.

9. Related Work

There are only a few other hybrid semantic service matchmakers available for OWL-S service profiles. We discuss each of them in very brief only (see also figure 1 in section 3), and refer to [10] for a coverage of SWS matchmaking in general.

Our OWLS-MX matchmaker is strongly inspired by the hybrid matchmaker LARKS [17]. However, LARKS differs from OWLS-MX in several aspects: LARKS performs IOPE matching of service profiles written in a proprietary capability description language with a description logic different from OWL-DL. Besides, LARKS does not offer logical subsumes nor subsumed-by nor hybrid nearest-neighbour matching, and has never been experimentally evaluated.

The logic-based variant OWLS-M0 of OWLS-MX is similar to the prominent OWLS-UDDI matchmaker [18] but is different with respect to the following issues: OWLS-UDDI makes use of a different notion of plug-in matching and does not perform additional subsumed-by matching. Further, OWLS-M0

allows to use arbitrary rather than known service query concepts into its local matchmaker ontology, and is not integrated with the UDDI registry standard for Web service discovery.

The hybrid semantic OWL-S service profile matchmaker iMatcher [9] uses multiple edit- or token-based text similarity metrics (Bi-Gram, Levenshtein, Monge-Elkan and Jaro similarity measures) to determine the degree of semantic matching between a given pair of OWL-S service profiles. Like OWLS-MX, the iMatcher transforms each structured service profile description into a weighted keyword vector that includes not only the names but terms derived by means of logic-based unfolding of its service input and output concepts. In this sense, iMatcher classifies as a hybrid matchmaker. However, it does not perform logic-based matching which resulted in lower precision and recall compared to OWLS-MX. In its adaptive mode iMatcher2 learns (over a test collection like OWLS-TC2.2) which of its ten text similarity measures to select best for a given query. It has been experimentally shown that the combined logical deduction and regression-based learning of text similarities of iMatcher2 is superior to logic-based only matching; iMatcher2 did outperform OWLS-MX in terms of precision.

The hybrid semantic service matchmaker FC-MATCH [1] performs a combined logic-based and text similarity-based matching of monolithic service and query concepts written in OWL-DL. In this approach, a service concept S is defined as logical conjunction of existential qualified role expressions where each role corresponds to a selected profile parameter: $S = \exists\text{hasCategory}(C_1) \sqcap \exists\text{hasOperation}(C_2) \sqcap \exists\text{hasInput}(C_3) \sqcap \exists\text{hasOutput}(C_4))$. Unlike monolithic logic-based service matching, FC-MATCH determines hybrid matching degrees by means of logic-based subsumption of their profile parameter concepts (C_i) together with computing the so-called Dice (name affinity) similarity coefficient between terms occuring in these concepts according to given terminological relationships of the thesaurus WordNet. However, to the best of our knowledge, FC-MATCH has not been experimentally evaluated yet.

[12] presents an approach to hybrid matching of monolithic logic-based semantic service descriptions in OWL-DL extended with pricing policies (modeled in DL-safe SWRL rules) according to given preferences by means of SPARQL queries to a given service repository.

10. Conclusions

The presented approach to hybrid semantic Web service matching, called OWLS-MX, utilizes both logic based reasoning and non-logic based IR techniques for semantic Web services in OWL-S. Experimental evaluation results provide strong evidence in favor of the proposition that the performance of logic-based matchmaking can be considerably improved by incorporating non-logic based information retrieval techniques into the matching algorithms.

The hybrid matchmaker OWLS-MX has been successfully used in two fielded mobile e-health sysems for emergency medical assistance and repatriation planning, namely the Health-SCALLOPS system (www.dfki.de/scallops) and the CASCOM system (www.ist-cascom.org).

References

[1] D. Bianchini, V. D. Antonellis, M. Melchiori, D. Salvi, Semantic-enriched service discovery, in: Proceedings of IEEE ICDE 2nd International Workshop on Challenges in Web Information Retrieval and Integration (WIRI06), Atlanta, Georgia, USA, 2006.

[2] L. Botelho, A. Fernandez, M. Klusch, L. Pereira, T. Santos, P. Pais, M. Vasirani, Service discovery., in: M. Schumacher, H. Helin (Eds.): CASCOM - Intelligent Service Coordination in the Semantic Web. Chapter 10. Birkh"auser Verlag, Springer, 2008.

[3] W. Cohen, P. Ravikumar, S. Fienberg, A comparison of string distance metrics for name-matching tasks, in: Proc. IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), DBLP at http://dblp.uni-trier.de, 2003.

[4] D. Fensel, F. van Harmelen, Unifying reasoning and search to web scale., in: IEEE Internet Computing, March/April 2007.

[5] S. Grimm, Discovery - identifying relevant services., in: Semantic Web Services. Concepts, Technologies, and Applications. Springer, 2007.

[6] I. Horrocks, P. Patel-Schneider, F. van Harmelen, From shiq and rdf to owl: The making of a web ontology language, Web Semantics, 1(1), Elsevier.

[7] M. Jaeger, G. Rojec-Goldmann, C. Liebetruth, G. M"uhl, K. Geihs, Ranked matching for service descriptions using owl-s., in: Proceedings of 14. GI/VDE Fachtagung Kommunikation in Verteilten Systemen KiVS, Kaiserslautern, 2005.

[8] F. Kaufer, M. Klusch, Wsmo-mx: A logic programming based hybrid service matchmaker., in: Proceedings of the 4th IEEE European Conference on Web Services (ECOWS 2006), IEEE CS Press, Zurich, Switzerland, 2006.

[9] C. Kiefer, A. Bernstein, The creation and evaluation of isparql strategies for matchmaking., in: Proceedings of European Semantic Web Conference, Springer, 2008.
[10] M. Klusch, Semantic web service coordination., in: In M. Schumacher, H. Helin (Eds.): CASCOM - Intelligent Service Coordination in the Semantic Web. Chapter 4. Birkh"auser Verlag, Springer, 2008.
[11] M. Klusch, Z. Xing, Deployed semantic services for the common user of the web: A reality check., in: Proceedings of the 2nd IEEE International Conference on Semantic Computing (ICSC), Santa Clara, USA, IEEE Press, 2008.
[12] S. Lamparter, A. Ankolekar, Automated selection of configurable web services., in: 8. Internationale Tagung Wirtschaftsinformatik. Universitaetsverlag Karlsruhe, Karlsruhe, Germany, March 2007.
[13] L. Li, I. Horrocks, A software framework for matchmaking based on semantic web technology, in: Proceedings of the Twelfth International Conference on World Wide Web, ACM Press, 2003.
[14] T. D. Noia, E. Sciascio, F. Donini, M. Mogiello, A system for principled matchmaking in an electronic marketplace., in: Electronic Commerce, 2004.
[15] OWL-S, Semantic markup for web services; w3c member submission 22 november 2004, http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.
[16] M. Paolucci, T. Kawamura, T. Payne, K. Sycara, Semantic matching of web services capabilities., in: Proceedings of 1st International Semantic Web Conference (ISWC), 2002.
[17] K. Sycara, M. Klusch, S. Widoff, J. Lu, Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace, Autonomous Agents and Multi-Agent Systems, 5(2), Kluwer.
[18] K. Sycara, M. Paolucci, A. Anolekar, N. Srinivasan, Automated discovery, interaction and composition of semantic web services, Web Semantics, 1(1), Elsevier.
[19] TREC, Text retrieval conference, http://trec.nist.gov/data/.
[20] C. van Rijsbergen, Information Retrieval, 1979.
[21] A. M. Zaremski, J. M. Wing, Specification matching of software components, in: 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering, 1995.