Web Intelligence and Agent Systems: An International Journal 5 (2008) 1–5
IOS Press

WSMO-MX: A Hybrid Semantic Web Service Matchmaker

Matthias Klusch [a,*] Frank Kaufer [b]
[a] *German Research Center for Artificial Intelligence, Stuhlsatzenhausweg 3, Saarbrücken, Germany*
E-mail: klusch@dfki.de
[b] *Hasso Plattner Institute, University of Potsdam, Prof.-Dr.-Helmert-Strasse 2-3, Potsdam, Germany*
E-mail: frank.kaufer@hpi.uni-potsdam.de

Abstract. The WSMO-MX service matchmaker applies different matching filters to retrieve Semantic Web services written in a dialect of the prominent service description language WSML-Rule. For this purpose, WSMO-MX recursively computes logic-based and syntactic similarity-based matching degrees and returns a ranked set of services that are semantically relevant to a given query. The matching filters perform ontology-based type matching, logical constraint matching, and syntactic matching. In this paper, we present the service description language WSML-MX, the hybrid matchmaker WSMO-MX for WSML services converted to WSML-MX, and the results of our experimental evaluation of its performance in terms of recall and precision over the test collection WSML-TC2.

Keywords: Semantic Services, Web Services, WSML, Matchmaking, Discovery, Retrieval

1. Introduction

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. A Semantic Web service, or in short semantic service, is a Web service which functionality is described by use of logic-based semantic annotation over a well-defined ontology. In the following, we focus on the discovery of semantic services. Semantic service discovery can be performed in different ways depending on the service description language, the means of service selection and its coordination by means of a broker, matchmaker or mediator [23], or in a peer-to-peer fashion.

Service selection encompasses semantic matching and ranking of services to select one or more most relevant services to be invoked, starting from a given set of available services. Semantic service matching is the pairwise comparison of an advertised service with a desired service (query) to determine the degree of their semantic match. This process can be non-logic-based, logic-based or hybrid depending on the nature of reasoning means used by the matchmaker.

Non-logic-based matching can be performed by means of, for example, graph matching, data mining, linguistics, or content-based information retrieval to exploit semantics that are either commonly shared (in XML namespaces), or implicit in patterns or relative frequencies of terms in service descriptions. Logic-based semantic matching of services like those written in the prominent service description languages OWL-S (Ontology Web Language for Services), WSML (Web Service Modeling Language) and the W3C recommendation SAWSDL (Semantically Annotated WSDL) exploit standard logic inferences.

In line with the recently started shift of Semantic Web research towards more scalable and approxima-

*Corresponding author. E-mail: klusch@dfki.de. Partial support provided by BMBF (German Ministry for Education and Research) grants MODEST 01-IWO-8001 and SCALLOPS 01-IW-D02.

tive rather than computationally expensive logic-based reasoning with impractical assumptions [10], we claim that the quality of semantic service selection can be significantly improved by combining both logic-based only and syntactic matching where each of them alone would fail. One example of a hybrid semantic service matchmaker for services in WSML is our matchmaker WSMO-MX. It applies different logic-based and non-logic-based semantic matching filters to retrieve WSML services that are semantically relevant to a given query [14]. For a survey and classification of semantic service matchmakers for different service description formats, we refer to [16].

In this paper, we focus on hybrid matching of WSML services. We first informally introduce the semantic service description language WSML including its variant WSML-Rule, and then the WSML-Rule dialect WSML-MX used by our matchmaker WSMO-MX in section 2. This is followed by an overview of the hybrid semantic matching algorithm of WSMO-MX together with an example in section 3. The implementation of WSMO-MX is briefly described in section 4. The experimental setup of the performance evaluation and its results are described in section 5. Related work on semantic matchmakers for WSML services is summarized in section 6; we conclude in section 7.

2. Semantic service description

In this section, we informally introduce the reader to the basic elements of semantic service description in the Web Service Modeling Language (WSML) and its variant WSML-MX that is used by the matchmaker WSMO-MX.

2.1. Service description in WSML

WSMO. The WSMO (Web Service Modeling Ontology) framework[1] provides a conceptual model and a formal language WSML (Web Service Modeling Language) [2] for the semantic markup of Web services together with a reference implementation WSMX (Web Service Execution Environment). WSMO offers four key components to model different aspects of Semantic Web services in WSML: Ontologies, goals, services, and mediators. Goals in goal repositories specify objectives that a client might have when searching for a relevant Web service. WSMO ontologies provide the formal logic-based grounding of information used by all other modeling components. Mediators bypass interoperability problems that appear between all these components at data (mediation of data structures), protocol (mediation of message exchange protocols), and process level (mediation of business logics) to "allow for loose coupling between Web services, goals (requests), and ontologies". Each of these components, called top-level elements of the WSMO conceptual model, can be assigned non-functional properties to be taken from the Dublin Core metadata standard by recommendation.

WSML. The Web service modeling language WSML allows to describe a Semantic Web service in terms of its actual functionality (service capability) and the interface through which it can be accessed for orchestration and choreography. The formal semantics of elements are specified as logical axioms and constraints in ontologies using one of five WSML variants: WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full. Though WSML has a special focus on annotating Semantic Web services like OWL-S it tries to cover more representational aspects from knowledge representation and reasoning under both classical FOL and non-monotonic LP semantics. For example, WSML-DL is a variant of the description logic SHIQ (D) with frame-syntax and expressivity close to the description logic SHOIN (D), that is the variant OWL-DL of the standard ontology web language OWL. WSML-Rule is a fully-fledged logic programming language with function symbols, complex rules [3], inequality and non-monotonic negation, and meta-modeling facilities such as treating concepts as instances, but does not feature arbitrary use of existential quantifiers, disjunctions in rule heads, strict (monotonic) negation, and equality reasoning. The semantics of WSML-Rule is defined through a mapping to a logic programming variant of F-Logic [19] with inequality and default negation under Well-Founded Semantics (WFS) [33]. Only fragments of WFS are (semi-)decidable with respect to query evaluation [6].

F-Logic. F-Logic is an object-oriented extension of first-order predicate logic with objects of complex

[1] http://www.wsmo.org/TR/d2/v1.4/20061106

[2] http://www.wsmo.org/TR/d16/d16.1/v0.21/20051005/

[3] Unsafe Datalog/Horn rules and FOL rules which can be rewritten into those by means of Lloyd-Topor-transformations [27]

internal structure, class hierarchies and inheritance, typing, and encapsulation in order to serve as a basis for object-oriented logic programming and knowledge representation. For modeling ontologies, it allows to define, for example, is-a object class (or type) hierarchies through subclass relationships like person::human denoting class "person" as a subclass of "human", a class of objects with structured properties/relations (object type signature) like person[hasName ⇒ string, hasChild ⇒ person], and instances of classes (typed objects) like john:person as well as rules like (∀ *X,Y X[hasParent⟶⟶ Y]* ← *Y:person[hasChild ⟶⟶ X].*), denoting the inversion of relation "hasChild", i.e. if person "Y" has a child "X" than "Y" is parent of "X".

F-Logic is primarily a syntactic extension of predicate logic and, in principle, could be used in combination with a classical FOL language and semantics. However, beside an object-oriented syntax, F-Logic has a logic programming connotation. But since neither logic programming in general nor F-Logic in particular are clearly standardized syntax and semantics are dependent on the respective reasoner used. Most F-Logic reasoners like OntoBroker, Flora-2 and Florid operate under (or allow among several) a paradigm based on Van Gelder's well-founded semantics [33]. For more details on the syntax and semantics of F-Logic, we refer to [1,19,35].

Services in WSML. The semantic description of services and requests (goals) in WSML is structured into so-called service capability, service interface used for orchestration and choreography, and shared variables. A WSML service capability describes the state-based functionality of a service. Therefore a distinction in an information space (of the exchanged data) and the real world (background knowledge) is made. The WSML service capability modeling elements *precondition* (*assumption*) and *postcondition* (*effect*) describe the conditions over the information space (real world state) before and after service execution, respectively. The capability also specifies non-functional properties and all-quantified shared variables (with service capability as scope) for which the logical conjunction of precondition and assumption entails that of the postcondition and the effect. The syntax of WSML in general, and WSML-Rule in particular is mainly derived from F-Logic extended with more verbose keywords (e.g., "hasValue" for →, "p memberOf T" for *p:T* etc.), and has a normative human-readable syntax, as well as an XML and RDF syntax for exchange between machines.

Example of service capability in WSML-Rule. The functionality of a ticket reservation service can be defined in WSML-Rule using F-Logic with shared variables ?creditCard, ?initialBalance, ?trip, ?reservationHolder, ?ticket as follows. The service precondition specifies that a reservation request for a trip has to be made by a reservation holder with a credit card that is initially balanced before service execution:

```
precondition definedBy
   reservationRequest[reservationItem hasValue ?trip,
   reservationHolder hasValue ?reservationHolder]
   and ?creditCard[balance hasValue ?initialBalance]
   memberOf po#creditCard.
```

It is further assumed that the given credit card is valid and of a certain type before the service is executed:

```
assumption definedBy
   po#validCreditCard(?creditCard) and
   (?creditCard[type hasValue "PlasticBuy"] or
   ?creditCard[type hasValue "GoldenCard"] ).
```

The service postcondition specifies that the reservation holder obtains the ticket for the trip requested:

```
postcondition definedBy
   ?reservation memberOf tr#reservation[
      reservationItem hasValue ?ticket,
   reservationHolder hasValue ?reservationHolder]
   and
   ?ticket[trip hasValue ?trip] memberOf tr#ticket.
```

The world state effect of executing the service is that the reservation holder's credit card balance is reduced by the amount of the ticket price:

```
effect definedBy
   ticketPrice(?ticket, "euro", ?ticketPrice) and
   ?finalBalance= (?initialBalance - ?ticketPrice) and
   ?creditCard[po#balance hasValue ?finalBalance.
```

2.2. Service description in WSML-MX

The matchmaker WSMO-MX pairwisely matches services in an extension of WSML-Rule called WSML-MX. As a service-IOPE (input, output, precondition and effect) matchmaker, it focuses on match-

ing service profiles or capabilities but not process models; goals are described as desired services in WSML-MX.

WSML-MX. The basic idea behind WSML-MX is to extend the WSML variant WSML-Rule such that the user can specify preferences and relaxation constraints about the way semantic matching of desired service capability elements shall be performed by a matchmaker. For this purpose, WSML-MX introduces an additional language element to WSML-Rule, the so-called $derivative$ of an F-Logic class which is an extended version of the object set introduced by Klein and König-Ries [22].

A derivative D_T in WSML-MX encapsulates an ordinary concept T (in this context called type) with relation (type) signature logically defined in a given ontology by attaching meta-information mainly about the way how T can be matched with any other type. This meta-information is defined in terms of different meta-relations of the derivative D_T. In services, these meta-relations concern the use of service parameters as input ($param \Rightarrow\!\!> in$) or output parameters ($param \Rightarrow\!\!> out$) and logical constraints ($constraint \Rightarrow\!\!> c$). The type T itself is defined to be either atomic or a complex type with relations. By default the derivative D_T does not inherit the relations from T and can have a set of relations different from T. However, by means of rules it is possible to propagate the type relations from T to D_T, though with respective derivatives instead of plain classes like T in the ranges of the relations.

A state is a set of state parts, which are derivatives each defined as atomic, or as complex by means of relations with derivatives as range. Hence, any semantic service in WSML-MX can be represented as a directed object-oriented graph with derivatives considered as nodes and relations between them as edges, as shown in Figure 1.

As mentioned above, WSML-MX allows constraints on both relations and derivatives formulated in full F-logic. Let D be a derivative, C an F-Logic rule body and X_D a free variable in C, then we call c a *constraint* of D, if $D[constraint \twoheadrightarrow c]$ and $\forall X_D.satCons(X_D, c) \leftarrow C$ holds. Variable X_D is bound with potential instances of D, and $satCons$ verifies whether such an instance satisfies c. A derivative can have zero or many constraints including a special constraint for nominals; the respective meta-relation *oneOf* denoted as $D[oneOf \twoheadrightarrow \{i_1, \ldots, i_m\}]$ means that an instance of D has to be one of $i_1, \ldots, i_m$.

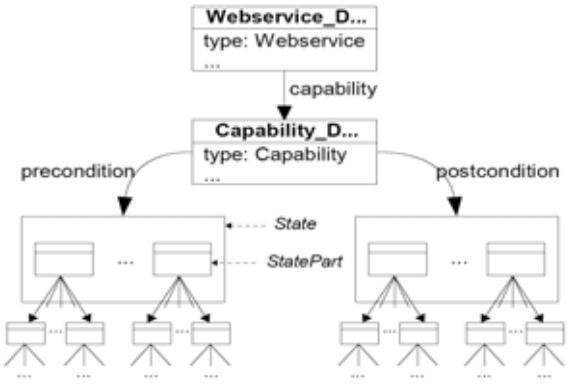


Fig. 1. Service derivative in WSML-MX

Hence, WSML-MX constraints are as expressive and, in general, only semi-decidable as WSML-Rule axioms are. However, the WSMO-MX matchmaker approximates query containment through means of so-called relative query containment for constraint matching (cf. section 3.4.3). Moreover, the matching of parts of WSML-MX expressions represented as acyclic object-oriented graphs without constraints is decidable in polynomial time.

The emphasis of WSML-MX on these parts of service modeling is motivated not only by (a) a clear separation of computationally tractable elements but (b) the option of providing a more detailed explanatory feedback to the user and more differentiated matching valuations by a matchmaker. This is a lesson learned from previous matchmaking approaches relying on pure query containment which requires high ontological homogeneity and results in single match predicates based on overall and undifferentiated logical implication between goal and service descriptions.

Service derivative in WSML-MX. An example of a service derivative in WSML-MX is shown in Figure 2. The functionality of the service (derivative) $Webservice_{D2}$ is described by means of its capability including the postcondition to hold on structured input and output parameters in terms of the logical constraint $c2$.

This constraint (the F-Logic rule at the bottom of the figure) on the output parameter derivative $Ticket_{D5}$ (with meta-relation $param \twoheadrightarrow out$; $constraint \twoheadrightarrow c2$) ensures that the service returns tickets for any trip between any two German towns, but if the departure is from Berlin, the destination must be Hamburg. The service has a nested (input) relation signature:

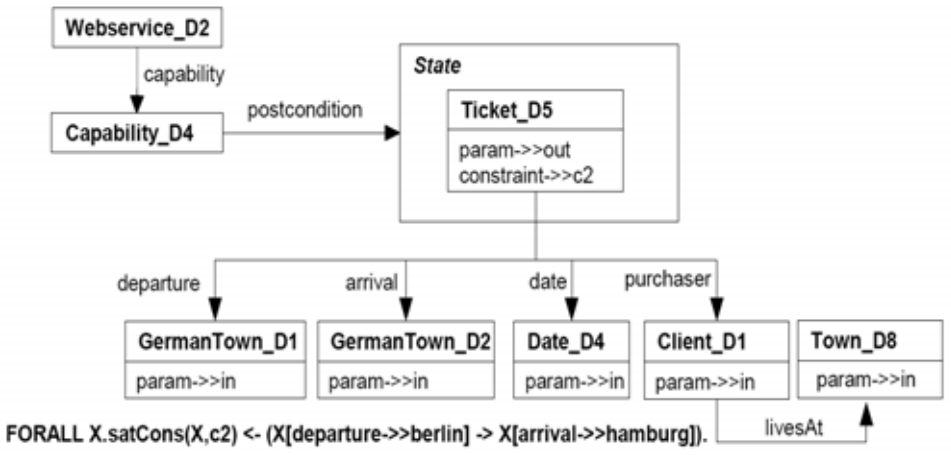


Fig. 2. Example service in WSML-MX

Ticket_D5[departure ⇛ GermanTown_D1;
arrival ⇛ GermanTown_D2;
date ⇛ Date_D4;
client ⇛ Client_D1[livesAt ⇛ Town_D8]].

Goal derivative in WSML-MX. A goal derivative in WSML-MX is described in the same way as a service derivative extended with meta-relations that allow specifying relaxations for syntactic, parameter, relation, constraint and ontology-based type matching of (part of) the derivative with (part of) a given service derivative. Examples of such meta-relation signatures of a goal derivative D_G for its matching with a service derivative D_S are

- $D_G[typeSimRel \rightarrow\!\!\!\!\rightarrow$ TSR] specifying the requested type TSR of ontology-based logical similarity relation between D_G and D_S such as equivalence, subsumption, superclass, sibling etc.;
- $D_G[synSimMetric \rightarrow\!\!\!\!\rightarrow M]$ specifying which text similarity metric M to use for syntactic matching of D_G with D_S;
- $D_G[synSimMinDegree \rightarrow\!\!\!\!\rightarrow \alpha]$ with threshold $\alpha \in [0,1]$ specifying the minimum degree of syntactic similarity of D_G with D_S;
- $D_G[missingStrat@(S_\mu) \rightarrow\!\!\!\!\rightarrow MS_\mu]$, w. $MS_\mu \in \{assumeEquivalent, assumeFailed, ignore\}$ specifying a strategy for missing relation S_μ of D_G in D_S for the matchmaker in terms of either ignoring this fact, or assuming the existence (equivalence), or non-existence of an equivalent relation (failure) for the matching result.

An example of a goal derivative together with more matching related meta-relations for its parts are given in section 3.4 where we demonstrate the hybrid matching process of the WSMO-MX matchmaker with the service example above.

3. WSMO-MX Overview

In the following, we summarize the functionality of the WSMO-MX matchmaker in general and through a simple example in particular. For further details of WSMO-MX, we refer the interested reader to [14].

3.1. Service matching degrees

The result of matching a derivative D_G from a goal description with a derivative D_W from a service description is a vector $v \in R^7$ of aggregated similarity valuations of (a) logical ontology-based concept matching, (b) logical constraint matching, (c) recursive relation matching (identified by name), and (d) syntactic similarity-based matching. In this respect, the semantic service matching of WSMO-MX is hybrid. Each real-valued entry in the so called service matching valuation vector

$$v = (\pi_\equiv, \pi_\sqsubseteq, \pi_\sqsupseteq, \pi_\sqcap, \pi_\sim, \pi_\circ, \pi_\bot)$$

with $\pi_i \in [0,1]$ ($i \in \{\equiv, \sqsubseteq, \sqsupseteq, \sqcap, \sim, \circ, \bot\}$) and $\sum \pi_i = 1$, denotes the extent (also called the semantic similarity score) to which both derivatives D_G and D_W match with respect to the hybrid semantic matching degrees π_i. As shown in Table 1, the logic-based

semantic matching degrees are computed as the logical relations *equivalence* (or *exact*), *subsumption*, *intersection* and *disjunction* (*fail*). If the precondition of a goal implies the precondition of a service or/and if the postcondition of a service implies the postcondition of the goal, then subsumption is called *plug-in* as known from software component retrieval [36] or the similar *rule of consequences* from Hoare logic [13]. Otherwise subsumption is called *inverse-plugin* indicating that goal and service would be a plug-in match if their roles were inverted. For a service requester, this is a more precise indication than intersection and a likely easier adjustment to the offered service. The degree of *fuzzy* similarity refers to a (implicit semantic) match not expressible in discrete logic or set-theoretical predicates such as syntactic similarity or numeric path-length distance in the ontology (excluding parent-child paths), while the degree *neutral* stands for neither match nor fail, hence declares the tolerance of matching failure.

The set-theoretic semantics of these hybrid matching degrees (cf. Table 1) base on the computed relations between the maximum possible instance sets of the derivatives D_G and D_W, denoted by $\mathcal{G}$ and $\mathcal{W}$. We use the heuristic relative query containment for logical constraint matching restricting these sets to the finite sets of known instances in the matchmaker knowledge base which satisfy the given logical constraints in F-Logic.

Please note that, in general, it cannot be taken for granted that a semantic service matchmaker does possess instances for every service or query derivative in its knowledge base (instance store) to check their satisfiability with respect to given constraints. For example, independent third-party matchmakers not hosted by any service provider or consumer might not have such specific object knowledge. However, sets of representative service instances could be obtained from the respective providers at the time of their registering of services at the matchmaker, through tracking of service executions (only in case of additional brokerage by the matchmaker), and sampling descriptions of services without real-world effects, or - regarding goal-derivative instances - by conducting systematic questionnaires with users. Furthermore, logic-based constraint matching could be ignored (which is a configuration option of WSMO-MX) and only used when such instances are available to the matchmaker.

3.2. Hybrid semantic service matching

In order to compute the degrees of hybrid semantic matching of given goal and service derivatives in WSML-MX, WSMO-MX recursively applies different filters of so-called IOPE (input, output, precondition, effect) matching to their preconditions and postconditions inherently including service inputs and outputs as in WSML (but with an explicit parameter flag similar to the variables in [21]), and returns the aggregated matching valuation vector. In addition, it provides annotations of the matching process results as a kind of explanatory feedback to the user. That facilitates a more easy iterative goal refinement by the user in case of insufficient matching results. These matching process annotations have a generic format and can be employed for several purposes. In the current version, WSMO-MX uses them to generate explanation text about the respective matching deviations in English. In the future the annotations could also be used for graph-based visualizations of the matching result.

More concrete, the state of the goal is matched with that of the service by matching their state part derivatives and then recursively by the pairwise matching of derivatives in the range of equally named service and goal relations. Subsequently, WSMO-MX computes the maximum weighted bipartite graph match, where nodes of the graph correspond to the goal and service state parts. The respectively computed valuation vectors act as weights of edges existing between the two state parts to be matched.

At each step in the recursion, the parameter matching filter is applied first, since its result, an annotation record, is not valuated for any of the hybrid matching degrees. Then each of the logic-based semantic matching filters is applied as follows. While type (or concept/class) matching bases on the logical subclass relations and path distance between types (or concepts) in the matchmaker ontology, the F-logic constraint matching is computed by means of relative query containment restricted to the set of known facts and objects asserted in the knowledge base of the matchmaker. Relation matching recursively matches the ranges of equally named relations with each other. Syntactic matching is performed in case one of these filters fails (compensative), or complementary in any case, if not specified differently. The user can also ask for just a first coarse-grained filtering by means of full-text syntactic matching without any logic-based semantic matching.

order	symbol	degree of match	pre	post
1	$\equiv$	equivalence	$\mathcal{G} = \mathcal{W}$	
2	$\sqsubseteq$	plugin	$\mathcal{G} \subseteq \mathcal{W}$	$\mathcal{W} \subseteq \mathcal{G}$
3	$\sqsupseteq$	inverse-plugin	$\mathcal{G} \supseteq \mathcal{W}$	$\mathcal{W} \supseteq \mathcal{G}$
4	$\sqcap$	intersection	$\mathcal{G} \cap \mathcal{W} \neq \emptyset$	
5	$\sim$	fuzzy similarity	$\mathcal{G} \sim \mathcal{W}$	
6	$\circ$	neutral	*by derivative specific definition*	
7	$\perp$	disjunction (fail)	$\mathcal{G} \cap \mathcal{W} = \emptyset$	

Table 1
Degrees of hybrid semantic matching of WSML service and goal derivatives

The computed hybrid matching degrees for pre- and postconditions of goal and service derivatives are totally sorted in descending order (cf. Table 1) with the logic-based matches $exact > plug-in > inverse-plug-in > intersection$ followed by the non-logic-based ones $fuzzy > neutral$ and $Fail$. The result of type, constraint, and relation (with missing relation strategy valuation) matching is associated to one of these hybrid matching degrees by means of a binary matching valuation vector (cf. Tables 2, 3, 4).

Finally, all valuation vectors computed during recursive relation matching of goal and service derivatives are aggregated into one single valuation vector by means of average. Each individual valuation vector can also be weighted with respect to some matching filter for this purpose by the user in the request; these weights are assumed to be of equal value by default. Optionally, this weighted average of hybrid matching degrees can be recomputed with respect to the intentions of the considered derivatives (cf. section 3.4.6).

The overall result of the matching process is a ranked list of services with their hybrid matching valuation vector and the matching process annotations for explanation. Services are ranked with respect to the maximum value of hybrid semantic matching degrees in descending order (cf. Table 1), starting with $\pi_{\equiv}$.

In the following, we describe each of the aforementioned matching filters of WSMO-MX in more detail, and then show an example of their application.

3.3. Hybrid semantic matching filters

3.3.1. Type matching

The matching of types T_G and T_W of the goal and service derivative D_G and D_W is performed by means of computing the degree of their semantic relation in the matchmaker ontology according to one or more requested (accepted) type similarity relations TSR defined as meta-relation values in $D_G[typeSimRel \twoheadrightarrow TSR]$. WSMO-MX offers the following matching relations between service and goal derivative types (subclass relationships) in F-Logic:

- $equivalent$: $T_W = T_G \vee T_W :: T_G \wedge T_G :: T_W$.
- sub: $T_W :: T_G$ (T_W subtype of T_G)
- $super$: $T_G :: T_W$
- $comAnc$ (common ancestor): $\exists T_P.T_G :: T_P \wedge T_W :: T_P$.
- $comDes$ (common descendant): $\exists T_C.T_C :: T_G \wedge T_C :: T_W$.
- $relative$: exists a path in the undirected ontology graph between T_G and T_W.

To further restrict the TSR a maximum distance $TD \in \mathbb{N}$ ($TD = 0 \equiv \infty$) between derivative types (or classes) in the matchmaker ontology can be specified in terms of $D_G[typeDistance \twoheadrightarrow TD]$. TD is the path length between both types in the undirected ontology graph. For the type relations $comAnc$ and $comDesc$ it must hold that the addition of the path lengths from both derivatives to their nearest common child/parent type is at most TD. Optionally, the same restriction can be imposed on the type relations sub and $super$ with TD greater or equal the path length from D_G to D_W.

There exist two further type similarity relations $spouse$ and $sibling$ which abbreviate $TD = 2$ for $comAnc$ and $comDesc$, respectively:

- $spouse$: $\exists T_C.T_C :: T_G \wedge T_C :: T_W \wedge \neg(\exists T_X.\exists T_Y.T_X \in \{T_G, T_W\} \wedge T_C :: T_Y \wedge T_Y :: T_X)$; types with one immediate common descendant (child).
- $sibling$: $\exists T_P.T_G :: T_P \wedge T_W :: T_P \wedge \neg(\exists T_X.\exists T_Y.T_X \in \{T_G, T_W\} \wedge T_X :: T_Y \wedge T_Y :: T_P)$; types with one immediate common ancestor (parent).

The valuation of the type matching of D_G and D_W for each of the hybrid semantic matching degrees of WSMO-MX is listed in Table 2[4]. If more than one type similarity relation TSR is specified in the goal, the maximum of the valuation vectors is selected as a result.

3.3.2. Relation matching

Given that the D_G and D_W are complex, the hybrid semantic matching must continue recursively with comparing their relations. Let the relation signatures of D_G and D_W be defined as follows: $D_G[R_1 \Rightarrow\!\!> E_1; ...; R_k \Rightarrow\!\!> E_k;\ S_1 \Rightarrow\!\!> F_1; ...;\ S_l \Rightarrow\!\!> F_l; ...;\ S_m \Rightarrow\!\!> F_m]$, and $D_W[R_1 \Rightarrow\!\!> G_1; ...; R_k \Rightarrow\!\!> G_k; T_1 \Rightarrow\!\!> H_1; ...; T_n \Rightarrow\!\!> H_n]$, where $R_1, ..., R_k, S_1, ..., S_m, T_1, ..., T_n$ are unique relation names with $\bigcup_{i\in[1,m]} S_i \cap \bigcup_{j\in[1,n]} T_j = \emptyset$ and derivatives $E_1, ..., E_k, G_1, ...G_k, F_1, ..., F_m, H_1, ..., H_n$ the respective ranges of the relations.

The relations $R_1, ..., R_k$ of the goal derivative D_G for which equally named relations do exist in D_W are valuated for the hybrid degree of matching by recursively matching their ranges with each other. That is, WSMO-MX attempts to match the (goal) derivatives E_τ with the (service) derivatives G_τ for all $\tau \in [1, k]$ and compute the respective valuation vectors.

We assume that for all relations S_μ, $\mu \in [1, l]$ in D_G that cannot be paired with an equally named relation in D_W (under unique name assumption for shared namespaces) there exist one so called *missing strategy* which indicates the matchmaker how to cope with this problem. Such a missing relation strategy is specified in the goal in terms of $D_G[missingStrat@(S_\mu) \rightarrow\!\!\!> MS_\mu]$, with $MS_\mu \in \{assumeEquivalent, assumeFailed, ignore\}$.

The valuations for relations with missing strategies are given in Table 3. It lists also the valuations for the relations without missing strategy ($S_l, \ldots S_m$ and $T_1, \ldots, T_n$), which depend on whether they are part of a pre- or postcondition.

The final valuation vector for the recursive relation matching between D_G and D_W is an equally weighted average of all valuation vectors computed for the missing relations, and those for the relation range derivative matchings.

3.3.3. Constraint matching

In WSMO-MX, the matching of logical constraints of goal and service derivatives in F-Logic is performed by means of so-called relative query containment. That is, any logical clause A is relatively contained in clause B, or A relatively implies B with respect to a given knowledge base $\mathcal{KB}$, denoted by $A \sqsubseteq_{\mathcal{KB}} B$, if the answer set $Q_{\mathcal{KB}}(A)$ of querying $\mathcal{KB}$ with A, is a subset of $Q_{\mathcal{KB}}(B)$. Under the open world assumption, $\mathcal{KB}$ does not contain all possible instances of a query (universal closure), hence relative query containment can only be considered as an approximation of logical implication (query containment) which is, in general, undecidable for many logic programming dialects [7]. An alternative would be to restrict constraints to conjunctive clauses and/or approximate logical implication by means of clause theta-subsumption which is, in general, NP-complete decidable [12]. Since fast deterministic algorithms for partial testing of theta-subsumption are also known [30], the correct but incomplete theta-subsumption relation is used as a consequence relation in many ILP systems [28], and the matchmaker LARKS [32].

However, for pragmatic reasons of implementation and relying on available reasoners for the main inference procedures, WSMO-MX uses relative query containment for matching constraints over the instances stored in the matchmaker ontology. For each derivative D of type T, WSMO-MX determines a set of potential instances against which its constraints are evaluated as queries. This set comprises all instances of the concept T and instances of derivatives of type T:

$$\forall D, X_D.\, potentialInstance(D, X_D) \leftarrow$$
$$\exists T.\, D[type \rightarrow\!\!\!> T] \wedge$$
$$(X_D : T \vee (\exists D_T.\, D_T[type \rightarrow\!\!\!> T] \wedge X_D : D_T)).$$

The constraint matching filter then returns only those instances of this set which satisfy all constraints of D:

$$\forall X_D, D.\, satAllCons(X_D, D) \leftarrow$$
$$potentialInstance(D, X_D) \wedge$$
$$(\forall C.\, D[constraint \rightarrow\!\!\!> C] \rightarrow satCons(X_D, C)) \wedge$$
$$((\exists X.\, D[oneOf \rightarrow\!\!\!> X]) \rightarrow D[oneOf \rightarrow\!\!\!> X_D]).$$

The valuation of constraint matching is determined by the type of the set relation ρ, which is defined as $\mathcal{I}_{\mathcal{KB}}(D_G)\ \rho\ \mathcal{I}_{\mathcal{KB}}(D_W)$ over the set $\mathcal{I}_{\mathcal{KB}}(D) := \{X_D | satAllCons(X_D, D)\}$ of matching instances of derivative D with respect to the given knowledge base $\mathcal{KB}$ of the matchmaker (cf. Table 4).

[4]Please note that we switched the valuation for $comAnc$ and $comDesc$ compared to previous work [14]

type similarity relation	valuation vector	
	val_{pre}	val_{post}
	($\pi_{\equiv}$,$\pi_{\sqsubseteq}$,$\pi_{\sqsupseteq}$,$\pi_{\sqcap}$,$\pi_{\sim}$,$\pi_{\circ}$,$\pi_{\perp}$)	($\pi_{\equiv}$,$\pi_{\sqsubseteq}$,$\pi_{\sqsupseteq}$,$\pi_{\sqcap}$,$\pi_{\sim}$,$\pi_{\circ}$,$\pi_{\perp}$)
$equivalent$	(1 , 0 , 0 , 0 , 0 , 0 , 0)	(1 , 0 , 0 , 0 , 0 , 0 , 0)
sub	(0 , 0 , 1 , 0 , 0 , 0 , 0)	(0 , 1 , 0 , 0 , 0 , 0 , 0)
$super$	(0 , 1 , 0 , 0 , 0 , 0 , 0)	(0 , 0 , 1 , 0 , 0 , 0 , 0)
$comDes$	(0 , 0 , 0 , 1 , 0 , 0 , 0)	(0 , 0 , 0 , 1 , 0 , 0 , 0)
$comAnc$	(0 , 0 , 0 , 0 , 1 , 0 , 0)	(0 , 0 , 0 , 0 , 1 , 0 , 0)
$relative$	(0 , 0 , 0 , 0 , 1 , 0 , 0)	(0 , 0 , 0 , 0 , 1 , 0 , 0)

Table 2

Valuation of type matching for hybrid matching degrees

missing strategy	valuation vector	
	$val_{pre,webservice}/val_{post,goal}$	$val_{post,webservice}/val_{pre,goal}$
	($\pi_{\equiv}$,$\pi_{\sqsubseteq}$,$\pi_{\sqsupseteq}$,$\pi_{\sqcap}$,$\pi_{\sim}$,$\pi_{\circ}$,$\pi_{\perp}$)	($\pi_{\equiv}$,$\pi_{\sqsubseteq}$,$\pi_{\sqsupseteq}$,$\pi_{\sqcap}$,$\pi_{\sim}$,$\pi_{\circ}$,$\pi_{\perp}$)
$assumeEquivalent$	(1 , 0 , 0 , 0 , 0 , 0 , 0)	(1 , 0 , 0 , 0 , 0 , 0 , 0)
none	(0 , 1 , 0 , 0 , 0 , 0 , 0)	(0 , 0 , 1 , 0 , 0 , 0 , 0)
$ignore$	(0 , 0 , 0 , 0 , 0 , 1 , 0)	(0 , 0 , 0 , 0 , 0 , 1 , 0)
$assumeFailed$	(0 , 0 , 0 , 0 , 0 , 0 , 1)	(0 , 0 , 0 , 0 , 0 , 0 , 1)

Table 3

Valuation of relation matching with missing strategies for hybrid matching degrees

set relation	valuation vector	
	val_{pre}	val_{post}
$\mathcal{I}_{\mathcal{KB}}(D_G)\ \rho\ \mathcal{I}_{\mathcal{KB}}(D_W)$	($\pi_{\equiv}$,$\pi_{\sqsubseteq}$,$\pi_{\sqsupseteq}$,$\pi_{\sqcap}$,$\pi_{\sim}$,$\pi_{\circ}$,$\pi_{\perp}$)	($\pi_{\equiv}$,$\pi_{\sqsubseteq}$,$\pi_{\sqsupseteq}$,$\pi_{\sqcap}$,$\pi_{\sim}$,$\pi_{\circ}$,$\pi_{\perp}$)
$\mathcal{I}_{\mathcal{KB}}(D_G) = \mathcal{I}_{\mathcal{KB}}(D_W)$	(1 , 0 , 0 , 0 , 0 , 0 , 0)	(1 , 0 , 0 , 0 , 0 , 0 , 0)
$\mathcal{I}_{\mathcal{KB}}(D_G) \supseteq \mathcal{I}_{\mathcal{KB}}(D_W)$	(0 , 0 , 1 , 0 , 0 , 0 , 0)	(0 , 1 , 0 , 0 , 0 , 0 , 0)
$\mathcal{I}_{\mathcal{KB}}(D_G) \subseteq \mathcal{I}_{\mathcal{KB}}(D_W)$	(0 , 1 , 0 , 0 , 0 , 0 , 0)	(0 , 0 , 1 , 0 , 0 , 0 , 0)
$\mathcal{I}_{\mathcal{KB}}(D_G) \cap \mathcal{I}_{\mathcal{KB}}(D_W) \neq \emptyset$	(0 , 0 , 0 , 1 , 0 , 0 , 0)	(0 , 0 , 0 , 1 , 0 , 0 , 0)
$\mathcal{I}_{\mathcal{KB}}(D_G) \cap \mathcal{I}_{\mathcal{KB}}(D_W) = \emptyset$	(0 , 0 , 0 , 0 , 0 , 0 , 1)	(0 , 0 , 0 , 0 , 0 , 0 , 1)

Table 4

Valuation of constraint matching for hybrid matching degrees

3.3.4. Syntactic matching

The filter of WSMO-MX for syntactic matching of goal and service derivatives, D_G and D_W, is intended to complement those for semantic matching as described above. For this purpose, it transforms the description of each derivative into a weighted keyword vector as known from information retrieval, and applies one of the selected syntactic similarity metrics cosine, extended Jaccard, loss-of-information (LOI), and weighted LOI [25]. Similarity metrics and other syntactic matching parameters can be specified as instances of the following meta-relations of a goal derivative D_G.

- $D_G[synSimUsage \twoheadrightarrow U]$ with $U \in \{alternative, compensative, complementary\}$ specifies whether syntactic matching shall be performed either as an exclusive alternative to semantic matching, or only in case of semantic matching failure, or in any case.
- $D_G[synSimScope \twoheadrightarrow S]$ with $S \in \{scpType, scpRelation, scpDescription\}$ denotes whether only the types, or the relations, or the whole text of the description of the derivatives are used for syntactic matching. In case of *scpType*, all type names (no relation names) of the derivative are recursively unfolded in the matchmaker ontology and the resulting set of primitive components is used to compute a weighted keyword vector, whereas for *scpRelation* only the relation names of the derivative are used for this purpose. If relations are organized in relation hierarchies like classes the unfolding is done as well. Any combination of scopes is allowed.

- $D_G[synSimMetric \twoheadrightarrow M]$ with $M \in \{cosine, loi, loiWeighted, jaccard\}$ specifies which IR similarity metric to use. For details of computation, we refer to [25].
- $D_G[synSimMinDegree \twoheadrightarrow \alpha]$ with $\alpha \in [0,1]$ specifies the minimum degree of syntactic similarity required (threshold).

For the valuation of syntactic matching $\pi_\sim$ is set to the value of the computed syntactic similarity value and $\pi_\perp = 1 - \pi_\sim$. If the similarity value does not exceed α, then $\pi_\sim = 0$ and $\pi_\perp = 1$.

3.3.5. Parameter matching

A derivative can be tagged to be an input and/or output parameter by the meta-relation $param$. The parameter matching filter checks whether goal and service derivative are differently tagged and returns no valuation vector but an annotation indicating the deviations. This allows the service requester to understand the interface of the service and if needed to adjust the interface as it was expected and denoted by the parameters tags in the goal description.

3.3.6. Intentional matching

Optionally, WSMO-MX does perform a kind of intentional matching of goal and service derivatives. For this purpose, we adopt the approach proposed by Keller et al. [18]. In particular, the semantics of their notions of $\exists$-intention and $\forall$-intention correspond with the evaluation of our meta-relation $existentialIntention$ to $true$ and $false$, respectively. The valuation vector of hybrid semantic matching can be "intentionally recomputed" by its multiplication with the transformation matrix that corresponds to the requested combination of intended provision of relevant instances as it is declared for the goal and the service derivative by the requester and provider, respectively.

The case in which $\forall$-intentions are declared for both derivatives, D_G and D_W, is equal to not using intentions at all, hence can simply be ignored by WSMO-MX. As a consequence, there remain three cases for each pre- and postcondition matching. These are computed by means of six intentional matching matrices (to be multiplied with the valuation vector) of which we show only those for the postcondition matching cases (for precondition matching the lines and columns for $\pi_\sqsubseteq$ and $\pi_\sqsupseteq$ in the matrices have to be inverted): (1) $I_{post,\exists G,\forall W}$: only D_G has an $\exists$-intention, (2) $I_{post,\forall G,\exists W}$: only D_W has an $\exists$-intention, (3) $I_{post,\exists G,\exists W}$: both derivatives have $\exists$-intentions. The matrices are defined as follows.

$$I_{post,\exists G,\forall W} = \begin{pmatrix} 1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1 \end{pmatrix}$$

$$I_{post,\forall G,\exists W} = \begin{pmatrix} \frac{1}{2}&\frac{1}{2}&0&0&0&0&0\\ 0&1&0&0&0&0&0\\ \frac{1}{5}&\frac{1}{5}&\frac{1}{5}&\frac{1}{5}&0&0&\frac{1}{5}\\ 0&0&\frac{1}{3}&\frac{1}{3}&0&0&\frac{1}{3}\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1 \end{pmatrix}$$

$$I_{post,\exists G,\exists W} = \begin{pmatrix} 1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0\\ \frac{1}{3}&0&\frac{1}{3}&0&0&0&\frac{1}{3}\\ \frac{1}{3}&0&\frac{1}{3}&0&0&0&\frac{1}{3}\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1 \end{pmatrix}$$

3.4. Example of service matching with WSMO-MX

Suppose the user defines a goal derivative as a desired service derivative *Ticket_D4* as shown in Figure 3. That is, she is looking for any ticket for a trip between two arbitrary towns, but if it starts in Berlin, then it must not end in Bremen. Please note, that the user may specify matching relaxations for any object of the goal as exemplified, but also different weights for the matching filters to be applied. In this example, we assume the filters to be equally weighted. Further, the derivatives T_D are of equally named types (T_D[type$\twoheadrightarrow$T]) that are defined in the matchmaker ontology and not explicitly shown in the example.
The part of the type hierarchy in the matchmaker ontology and all instances used in this example are shown in Figure 4.

In this example, the service derivative *Ticket_D5* given in section 2 will be matched against the goal derivative *Ticket_D4* as follows. Since the capabilities of both goal and service derivatives do not include any precondition, their hybrid semantic matching is restricted to the matching of their postcondition states.

1. **Type matching**: The goal derivative type "Ticket_D4" is logically equivalent to the service derivative type "Ticket_D5" according to the matchmaker ontology. Therefore, the valua-

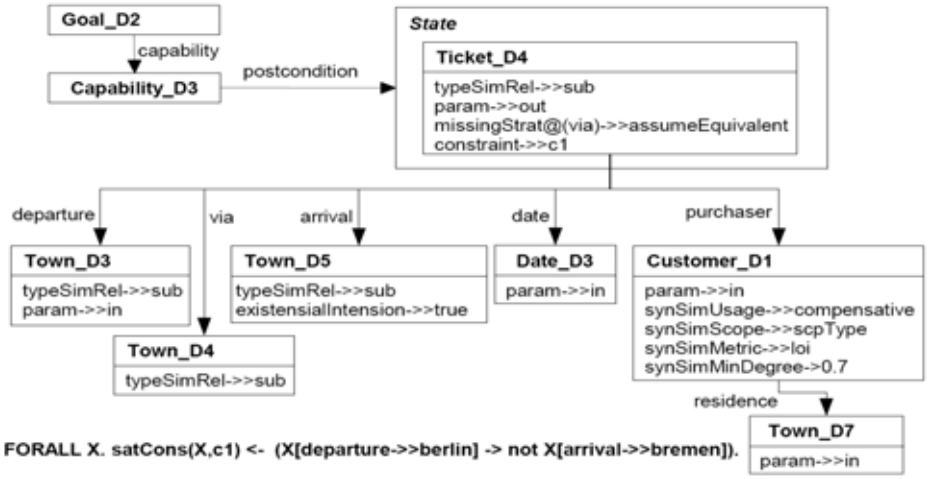
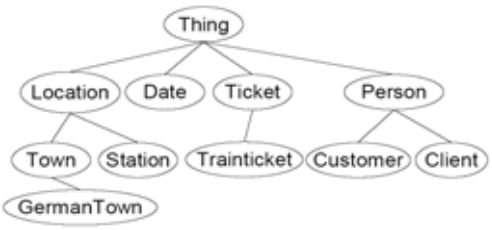


Fig. 3. Example goal in WSML-MX




```
t1:Ticket_D4[departure->>berlin;
             arrival->>leipzig; ...].
t2:Ticket_D4[departure->>berlin;
             arrival->>kiel; ...].
t3:Ticket_D5[departure->>hamburg;
             arrival->>bremen; ...].
t4:Ticket_D5[departure->>hamburg;
             arrival->>hannover; ...].
t5:Ticket_D5[departure->>berlin;
             arrival->>hamburg; ...].
t6:Ticket_D6[departure->>berlin;
             arrival->>bremen; ...].
```

Fig. 4. Example ontology (type hierarchy and instances)

tion vector for type matching is v_1 = (1,0,0,0,0,0,0).

2. **Parameter matching**: Both derivatives are marked as output parameters. No annotation necessary.

3. **Relation matching** Sorted pairs of equally named relations of goal and service derivatives are recursively matched as follows.

   Relation *departure*: The range of the relation "departure" of goal derivative "Ticket_D4" is the derivative "Town_D3" of type "Town" for which a subtype matching is allowed (meta-relation Town_D3[typeSimRel→↠sub]). Since the type "GermanTown" of the derivative "GermanTown_D1" as range of the equally named relation "departure of the service derivative "Ticket_D5" is not equivalent to but a subtype of "Town" according to the ontology, we get a type matching valuation in terms of a logical plug-in match, that is v_2 = (0,1,0,0,0,0,0).

   Relation *via*: There is no equally named relation in the service derivative "Ticket_D5", hence "via" is a missing relation. However, since the user specified a missing relation strategy for this relation in the goal (Ticket_D4[missingStrat(@via) →↠ assumeEquivalent]) the matchmaker assumes an equivalent relation for "via" in the service and returns a missing relation strategy matching valuation in terms of logical equivalence: v_3 = (1,0,0,0,0,0,0).

   Relation *arrival*: Since the type "GermanTown" of the range derivative of the relation *arrival* in the service is a subtype of the type "Town" of the same relation of the goal derivative according to the ontology, we obtain a type matching valuation in terms of a logical plug-in match, i.e. (0,1,0,0,0,0,0). But with the existential intention declaration of the derivative, the requester explicitly states to be satisfied with every subset of the actual requested derivative. Hence, by intentional re-computation the valuation is: v_4 = (1,0,0,0,0,0,0).

Relation *date*: The range types of this relation are equivalent in both goal and service which yields a type matching valuation in terms of logical equivalence: v_5 = (1,0,0,0,0,0,0).

Relation *purchaser*: Since the type "Customer" of the range derivative of this relation is a sibling of the type "Client" of the matched relation in the service derivative, they do not logically match, hence the matching of "Customer_D1" and "Client_D1" fails. However, the user allowed a relaxed matching of derivative "Customer_D1" by means of a compensative syntactic matching of its type "Customer" (Customer_D1[synSimScope $\longrightarrow\!\!\!\!\rightarrow$ scpType]). For this purpose, the loss-of-information (LOI) metric shall be applied to the weighted keyword vector representations of type definitions "Customer" and "Client" logically unfolded in the matchmaker ontology. These vectors are (Customer:1, Town:1, Person:1, Location:1, Town:1), respectively, (Client:1, Town:1, Person:1, Location:1, Town:1) with LOI-based similarity degree 0.75. Since this syntactic similarity value exceeds the given threshold (Customer_D1[synSimMinDegree $\longrightarrow\!\!\!\!\rightarrow$ 0.7]), this yields a syntactic type matching valuation for fuzzy matching: v_6 =(0,0,0,0,.75,0,.25).

The overall result of this relation matching of goal derivative "Ticket_D4" with service derivative "Ticket_D5" is the average of the individual matching valuations in terms of the seven matching degrees:
$v_7 = \frac{v_2+...+v_6}{5}$ = (0.6, 0.2, 0, 0, 0.15, 0, 0.05).

4. **Constraint matching**: Any instance of the goal derivative "Ticket_D4" has to satisfy the logical constraint c1 (Ticket_D4[constraint $\longrightarrow\!\!\!\!\rightarrow$ c1). This is satisfied by the instances $t1, \ldots, t5$ of the matchmaker knowledge base. On the other hand, the constraint c2, which is imposed on instances of the service derivative "Ticket_D5" is satisfied by the instances $t3, \ldots, t5$ of the same knowledge base. That is, the answer set of instances for "Ticket_D5" is included in that for "Ticket_D4" which means that the service (output) constraint implies that of the goal yielding a constraint matching valuation in terms of a logical plug-in match: v_8 = (0,1,0,0,0,0,0).

Finally, the aggregated matching valuations of service and goal derivatives in terms of the seven matching degrees of WSMO-MX is

$$v_9 = \frac{v_1+v_7+v_8}{3} = (.53\ .4,\ 0,\ 0,\ .05,\ 0,\ .02)$$

Informally, that means that the service is semantically relevant to the goal according to 53% equivalence, 40% plug-in and 5% fuzzy matching while only 2% can not be matched at all. Though, it has to be recalled that the vector elements are valuation *scores* and (partially) based on heuristics. The ranking of multiple matching services is described in the following.

3.5. Ranking of matched services

While valuation vectors are the final outcome of the semantic service matching process, WSMO-MX provides the user with a flexible service ranking mechanism based on the matching results and user preferences. Suppose the matching of a given goal yielded two relevant services s and t with valuation vectors $v_s = (\pi_{s,1}, ..., \pi_{s,7})$ and $v_t = (\pi_{t,1}, ..., \pi_{t,7})$, respectively. Furthermore, $rank(x, y, z)$ denotes a generic and polymorphic ranking function (denoted as predicate with its last argument being the return value) which takes two services x and y and binds z to the one ranked higher. A trivial ranking function is $rank_{rand}(x, y, z)$ which randomly binds z to either x or y.

The first versions of WSMO-MX (until WSMO-MX v0.5) provide a strict lexicographic ranking based on the total order of the seven matching degrees of returned valuation vectors. That means, valuation vectors are compared from left to right, and as soon as one element is greater in one vector the respective service is ranked higher; if the values of all matching degrees are equal for both services, one of them is randomly ranked higher than the other:

$$rank_{lex}(s,t,s) \Leftrightarrow \exists k.\forall i < k.(\pi_{s,i} = \pi_{t,i}) \wedge (\pi_{s,k} > \pi_{t,k})$$
$$rank_{lex}(s,t,t) \Leftrightarrow \exists k.\forall i < k.(\pi_{s,i} = \pi_{t,i}) \wedge (\pi_{s,k} < \pi_{t,k})$$
$$rank_{lex}(s,t,r) \Leftrightarrow \forall i.(\pi_{s,i} = \pi_{t,i}) \wedge rank_{rand}(s,t,r)$$

The idea was to prefer services of which some parts at least match with strong logic-based matching degrees like semantic equivalence or plug-in over those which have only significant fuzzy similarity. However, this kind of ranking can yield counter-intuitive results, even if no fuzzy similarity is involved. As-

sume services s and t are rated with valuation vectors $v_s = (.1, 0, ..., 0, .9)$ and $v_t = (0, 1, 0, ..., 0)$, respectively. While only very few parts of s did equivalently match with the corresponding ones of the given goal, every part of the service t semantically plug-in matches with the goal, hence received a 100% plug-in valuation. However, the lexicographic ranking of both yields $rank_{lex}(s, t, s)$, that is, service s is ranked higher than t.

The current version v0.6 of WSMO-MX provides an alternative ranking mechanism which (a) fully exploits the differentiation enabled by hybrid matching degrees of valuation vectors, and (b) allows flexible adjustment by the user. This is achieved by weighting each valuation with an individual real-valued matching degree weight vector $w = (w_{\equiv}, w_{\sqsubseteq}, w_{\sqsupseteq}, w_{\sqcap}, w_{\sim}, w_{\circ}, w_{\perp})$, where $w_i \in [0, 1]$ and $\sum w_i \leq 7$. The ranking function $rank_{wesca}$ is defined through the user-specified weighting scheme w with real-valued scalar ranking value $rv(s)$ ($rv(t)$) as a result as follows:

$$rank_{wesca}(s, t, s) \Leftrightarrow v_s \cdot w^T > v_t \cdot w^T$$
$$rank_{wesca}(s, t, t) \Leftrightarrow v_s \cdot w^T < v_t \cdot w^T$$
$$rank_{wesca}(s, t, r) \Leftrightarrow$$
$$v_s \cdot w^T = v_t \cdot w^T \wedge rank_{rand}(s, t, r)$$

This rather simple preference-based ranking also allows to express $rank_{lex}$ by restricting π_i to d digits ($d - 1$ decimals or 1.00) and using the weighting scheme $w_{lex} = (1, 10^{-1*d}, 10^{-2*d}, ..., 10^{-5*d}, 0)$. For convenience, WSMO-MX offers the following predefined weighting schemes with different kinds of matching degree preferences:

- $w_{plugin} = (1, 1, 0, 0, 0, 0, 0)$ (plug-in or exact matches prefered)
- $w_{logic-dec} = (1, \frac{6}{7}, \frac{5}{7}, \frac{4}{7}, 0, 0, 0)$ (logic-based matching degrees prefered with decreasing weights)
- $w_{fuzzy-dec} = (1, \frac{6}{7}, \frac{5}{7}, \frac{4}{7}, \frac{3}{7}, 0, 0)$ (logic-based and fuzzy matching degree prefered with decreasing weights)
- $w_{fuzzy} = (1, 1, 1, 1, 1, 0, 0)$ (no preferences)

Increasing the weight for fault-tolerance, that is $w_{\circ}$, can be considered as an orthogonal option. On the other side, there should not be any good reason to set the weight for failure, that is $w_{\perp}$, to a value greater than 0. Reconsidering the above example, the preference-based ranking $rank_{wesca}$ ranks service t higher than service s for each of the above weighting schemes. In particular, the ranking value of t for w_{plugin} and w_{fuzzy} is $rv(t) = 0.9$, and for $w_{logic-dec}$ and $w_{fuzzy-dec}$ it is $rv(t) = 0.86$. The ranking value for service s is the same for all these schemes: $rv(s) = 0.1$.

4. Implementation

WSMO-MX has been implemented in Java and F-Logic, and its software architecture is divided in two main parts, WSMO-MX-Core and WSMO-MX-Tapestry. WSMO-MX-Core comprises the basic functionality including the matching algorithm, several tools for the analysis of matching results, and an ontology manager interface for WSML-MX ontology management. The ontology manager abstracts reasoners and language (F-Logic) specifics into a common space. So far only a plugin module for OntoBroker is available. WSMO-MX-Tapestry is a service-oriented Web application on top of WSMO-MX-Core, Apache-Tapestry and Apache-Hivemind, and can be run by a JEE container like Apache-Tomcat. Tapestry and Hivemind are included in the version WSMO-MX 0.4. For future releases of WSMO-MX, it is intended to make WSMO-MX-Core also accessible by Web service communication protocols and/or agent communication languages.

Although the matchmaker is designed to work with different F-Logic reasoners, the current version necessitates the deployment of the commercial Ontobroker reasoner, which is not part of this WSMO-MX version, but can be obtained including appropriate (research) licences from Ontoprise GmbH, Karlsruhe. One alternative is to obtain the open source ontology manager system Flora-2.

WSMO-MX comes with a convenient Web based user interface. Among other features, it allows to flexibly configure the matchmaker, shows which ontology manager (reasoner) is plugged in (as a HiveMind service), disconnects and synchronizes ("reload ontology") with it, and also provides a rudimentary natural language explanation of returned matches. WSMO-MX version 0.4 is available under MPL license at the open software portal SemWebCentral[5].

5. Evaluation of performance

The experimental evaluation of the retrieval performance of WSMO-MX focuses on measuring its recall

[5] http://projects.semwebcentral.org/projects/wsmomx/

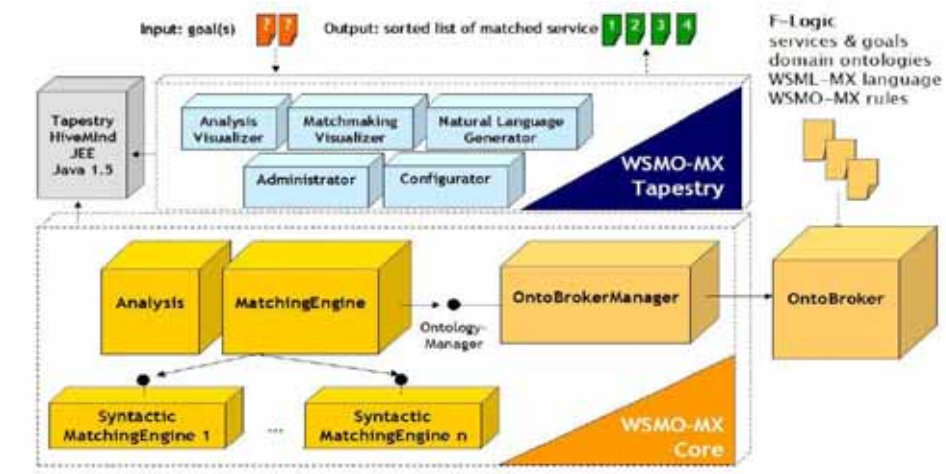


Fig. 5. Architecture of the WSMO-MX matchmaker

and precision based on an extension of our test collection WSML-TC2. The performance measures are defined as follows:

$$Recall = \frac{|A \cap B|}{|A|}, Precision = \frac{|A \cap B|}{|B|},$$

where A is the set of all relevant documents for a request, and B the set of all retrieved documents for a request. The so-called F1-measure equally weights recall and precision and is defined as:

$$F1 = \frac{2 \times Precision \times Recall}{Recall + Precision}.$$

We adopt the prominent macro-averaging of precision. That is, we compute the mean of precision values for answer sets returned by the matchmaker for all queries in the test collection at standard recall levels $Recall_i$ ($0 \leq i < \lambda$). Ceiling interpolation is used to estimate precision values that are not observed in the answer sets for some queries at these levels; that is, if for some query there is no precision value at some recall level (due to the ranking of services in the returned answer set by the matchmaker) the maximum precision of the following recall levels is assumed for this value. The number of recall levels from 0 to 1 (in equidistant steps $n/\lambda, n = 1..\lambda$) we used for our experiments is $\lambda = 10$. Thus, the macro-averaged precision is defined as follows:

$$Precision_i = \frac{1}{|Q|} \times \sum_{q \in Q} \max\{P_o | R_o \geq Recall_i \wedge (R_o, P_o) \in O_q\},$$

where O_q denotes the set of observed pairs of recall/precision values for query q when scanning the ranked services in the answer set for q stepwise for true positives in the relevance sets of the test collection. For evaluation, the answer set is the set of all services registered at the matchmaker which are ranked by the matchmaker with respect to their (totally ordered) hybrid matching degree.

5.1. Testing environment

At the time of writing, there is no service retrieval test collection for WSML available. As a consequence, for testing the performance of WSMO-MX, we developed our own collection, called WSML-TC.

Service Retrieval Test Collection WSML-TC2. Our first initial test collection WSML-TC1 has been built upon domain ontologies, services and queries developed in the DIANE project [21][6]. We transformed services and queries from the project-specific F-DSD format into WSML-MX, and subjectively determined binary relevance sets for each query in the collection WSML-TC1. The results of our first experimental testing of WSMO-MX over this WSML-TC1 are presented in [17]. Meanwhile, the test collection has been updated and extended to a second version WSML-TC2 which contains 97 services and 22 queries with 325 concepts (types) and 810 instances in a given ontology together with relevance sets and over 2200 derivatives used by service and query descriptions in WSML-MX.

Software. For the retrieval performance test runs, we used our open-source tool SME[2] (*Semantic Web Service Matchmaker Evaluation Environment*) which

[6] *http://hnsp.inf-bb.uni-jena.de/wiki/index.php/DSD*

is available at SemWebCentral[7]. The SME[2] was designed as an extensible tool with a plugin design for different Web service matchmakers that allows to run retrieval performance tests over different test collections not restricted to a specific format. It is also utilized in the international S3 (Semantic Service Selection) contest[8].

Hardware. For the performance tests, OntoBroker v5.0, SME[2] and WSMO-MX v0.5 were deployed on a machine with Windows 2000, Java 6, CPU 1,7 GHz and 2 GB RAM.

5.2. Experiments

On the basis of version WSML-TC2 of our test collection, we conducted the following five experiments to investigate the matchmaker behavior with respect to different configurations of its semantic, syntactic and hybrid matching. The retrieval performance of WSMO-MX is classically measured in terms of its recall, precision and F1-values well-known from information retrieval [2]. The preprocessing of derivatives for syntactic matching is done online for each matching request but not persistently indexed such that the time efforts for syntactic and logic-based matchings remains comparable. In practice, however, all services can easily be indexed in prior which would reduce the time for one matching process to some milliseconds.

5.2.1. Logic-based semantic matching

In the first experiment, we investigated the performance of logic-based only matching of WSMO-MX. For this purpose, we consider service matching deviations from goal derivative types with increasing degree of relaxation as follows:

- *default*: Only service derivative type deviations that are explicitly granted in the goal derivative are allowed.
- *subSuper*: Service derivatives which types have a logical subclass relationship with the goal derivative are allowed.
- *relative-3*: Service derivative types are only required to have a maximum distance of three in the ontology.

The logical subtype relations are implemented directly in F-Logic and type deviations are classified by OntoBroker. OntoBroker also manages the integration

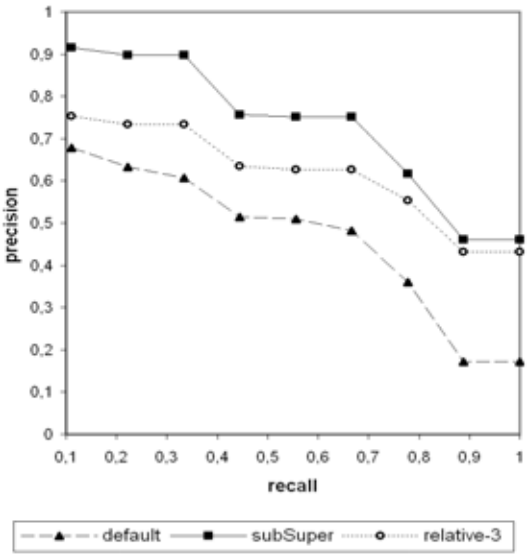


Fig. 6. Logic-based type matching

of service related domain ontologies into one matchmaker ontology. As shown in Figure 6, the logic-based matching configuration *subSuper* yields highest precision at all recall levels, since the test collection still relies on rather flat and homogenous domain ontologies. Not surprising, the configuration *default* is too restrictive in general which results in lower precision than *subSuper* for top-ranked results but it performs almost as good as *subSuper* for high recall values. The most relaxed logic-based matching configuration *relative-3* performs worse than each of the above with impractical average query response time of 24 seconds (*subSuper* and *default* require 8, respectively, 2.5 seconds).

5.2.2. Syntactic similarity-based matching

In this experiment, we compared the R/P performance of four selected token-based IR metrics for syntactic matching by WSMO-MX, that are Jaccard, Extended-Jaccard, Multiset-Jaccard and Cosine/TFIDF with syntactic similarity threshold of 0.6 and structural unfolding of service (and goal) derivative types and relations in the ontology. As shown in Figure 7, for this setting, all metrics except Extended-Jaccard perform almost as good as logic-based semantic matching by WSMO-MX in significantly less computational time most of which spent for unfolding and index generation per query (which can be done in prior for practical applications of WSMO-MX). Regarding the top-ranked results (corresponding to the leftmost part of the R/P curve), the Jaccard similarity metric

[7] http://projects.semwebcentral.org/projects/sme2/
[8] http://www.dfki.de/-klusch/s3/

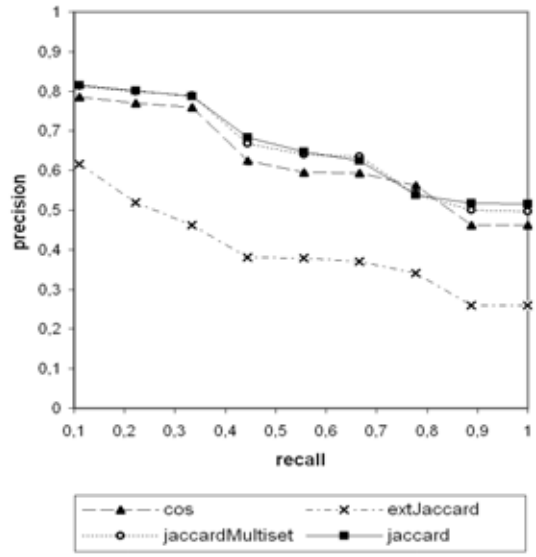


Fig. 7. Syntactic similarity-based matching (Jaccard, Cosine, Extended Jaccard, Multiset Jaccard)

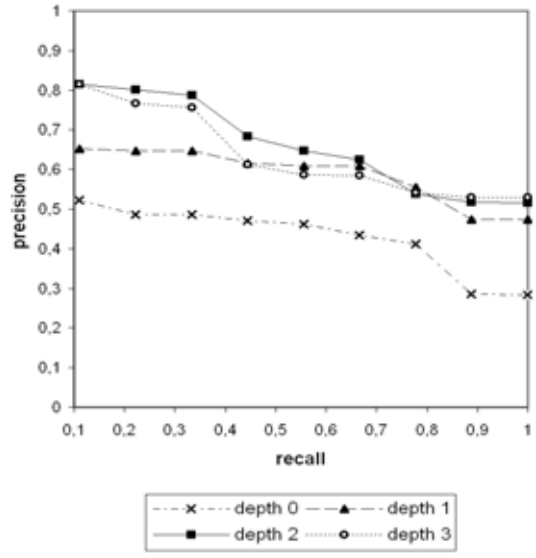


Fig. 8. Influence of structural service unfolding depth on syntactic similarity matching performance

performed best and is exclusively used for syntactic matching in the following experiments.

5.2.3. *Syntactic matching with varying depth of structural service unfolding*

The performance of both syntactic and logic-based matching depends on how much information about the given goal and service derivative can be taken into account by the matchmaker. In particular, for syntactic similarity measurement, this information is determined by the structural unfolding of a derivative type T, that is the maximum depth of the (nested) relational structure of T the matching algorithm (of WSMO-MX) is allowed to inspect for processing T into a weighted keyword vector.

Intuitively one would expect that the more complete the matching of T's structure, the better the result of its syntactic matching with any other service (goal) derivative type. However, this largely depends on the details to which services and goals are described in terms of their nested relation and type signatures with subtypes and relations defined in the ontology.

In this experiment, we successively increased the depth of service derivative structures to which the syntactic matching is allowed to unfold relations and types in the ontology for text similarity measurement from 0 (only the service derivative type itself, no relations and respective types of derivatives as ranges of these relations) to 3, that is the maximum depth of relational structures of service and goal derivatives in WSML-TC2. As shown in Figure 8, a value of 2 yields the best recall/precision result compared to the results for structural unfolding depths of 0 or 1 caused by too much of the derivative structures, hence information for semantic matching, being cut off. Computational time of structural unfolding and syntactic matching of derivatives is linearly dependent on the unfolding depth.

5.2.4. *Syntactic matching with varying scope of structural service unfolding*

In the last experiment the limited structural unfolding of service (and goal) derivatives covered both the complete (logical) unfolding of reached types and relations in the ontology. This scope of structural unfolding can be varied by means of logically unfolding either types or relations but not both in the ontology as follows:

- *types*: Only the set of all types of a complete service (or goal) derivative structure are logically unfolded in the matchmaker ontology and the resulting set of primitive components used for weighted keyword-based syntactic matching.
- *relations*: Only the names of all relations of a complete service (or goal) derivative structure are used for weighted keyword-based syntactic matching.

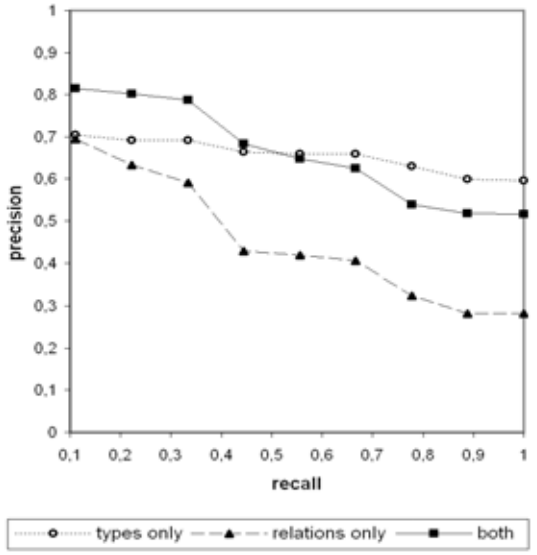


Fig. 9. Syntactic matching with different scopes: types, relations, both

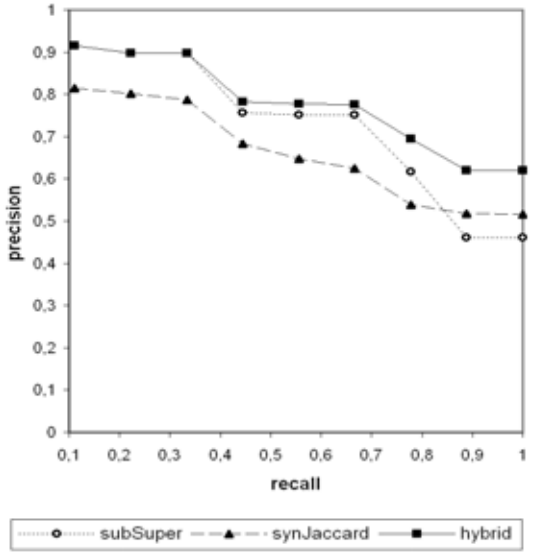


Fig. 10. Recall/Precision of logic-based, syntactic and hybrid semantic matching

For example, the logical unfolding of derivative Customer_D1 (Fig. 3 in section 3.4) with types as scope yields the weighed keyword vector (Customer:1, Person:1, Town:1, Location:1). If relations are the scope of the unfolding of this derivative, the resulting term vector is (residence:1). The combination of both would result in a vector containing all of the above weighted keyword entries. Not surprisingly, as shown in Figure 9, only syntactic matching with combined scope of structural unfolding of derivatives performed best with reasonable tradeoff between recall and precision, and average computational time twice as much as for only one of both scopes.

5.2.5. Hybrid vs. logic-based semantic matching

In this experiment, we compared the recall-precision performance of logic-based only, syntactic and hybrid semantic matching. For this purpose, we used matching configurations that performed best in the experiments above: (a) logic-based only *subSuper* matching, (b) syntactic matching with the Jaccard-metric, similarity threshold of 0.6 and combined scope of structural unfolding of derivatives to a maximum depth 2. The hybrid matching configuration combines both matching configurations and uses compensative syntactic matching.

As shown in Figure 10, all matching variants of WSMO-MX perform reasonably well in terms of precision and recall over the test collection WSML-TC2. Figure 11 shows the corresponding F1-values with

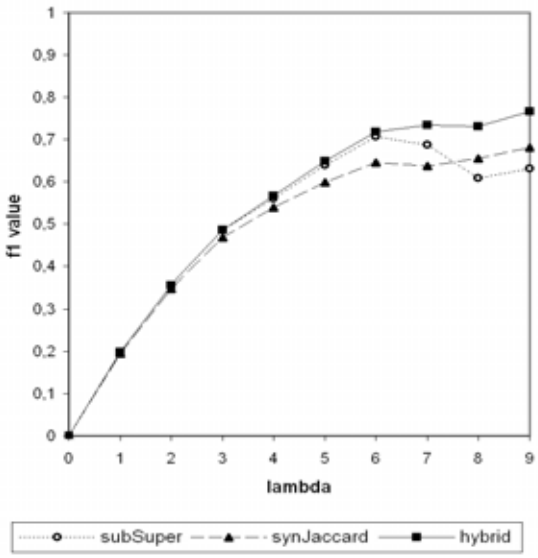


Fig. 11. F1 graph of semantic, syntactic and hybrid matching variants

equally weighted importance of recall and precision. The hybrid matching variant of WSMO-MX performs best since it avoids false-positives and false-negatives of both its syntactic and logic-based only matching. For example, logic-based only false-negatives can be avoided by compensative syntactic similarity measurement, while syntactic matching only false-

positives can be avoided by logic-based only *subSuper* matching. A more detailed false-positive/false-negative analysis is given in the following section. The hybrid matching variant of WSMO-MX over WSML-TC2 took four minutes to complete its run, that is slightly more than its logic-based only matching variant but significantly slower than its syntactic similarity matching which took only 47 seconds to complete.

5.3. Analysis of false-positives and false-negatives

In the following, we briefly discuss selected cases of logic-based false-positives (FP), that are irrelevant services classified as relevant, and logic-based false-negatives (FN), that are relevant services classified as irrelevant by logic-based only service matching of WSMO-MX. Many of these cases are avoided by WSMO-MX through complementary or compensative syntactic similarity-based matching.

5.3.1. Logic-based false positives

Main reasons for logic-based false positives returned by WSMO-MX are (a) its closed-world constraint matching by means of relative instance-based query containment, and (b) the ignorance of missing parameters by default in case there are no related missing-relation-strategies specified by the user in the goal.

Constraint matching by relative query containment. For example, consider the logical constraint in F-Logic of goal derivative "Ticket_D4" in Figure 3 which, informally, restricts the set of potential tickets (instances of X) for a trip between two arbitrary towns such that if the trip starts in Berlin, then it must not end in Bremen:

$$\forall X, X.satCons(X, c1) \leftarrow (X[departure \twoheadrightarrow berlin] \rightarrow \neg X[arrival \twoheadrightarrow bremen]).$$

A service offer that is considered irrelevant to this goal is a service with logical constraint $c2$ defined as

$$\forall X, X.satCons(X, c2) \leftarrow (X[departure \twoheadrightarrow berlin] \rightarrow X[arrival \twoheadrightarrow bremen]).$$

which requires that, in contrast to $c1$, if the trip starts in Berlin, then it must end in Bremen.

However, logical constraint matching by WSMO-MX can fail to detect this difference and return the service as relevant. The reason is that constraint matching relies on the computation of instance set relations (relative query containment). For example, if the instance sets of both constraints are empty, hence trivially equal, the constraint matching filter returns the top matching degree of logical equivalence not checking any valid but lower degree, hence fails to detect a logical failure (cf. Table 4). Similarly, if there are instances in the knowledge base that satisfy the goal but the instance set of the service constraint is empty, the (postcondition) constraint matching filter wrongly determines the constraint relationship between goal and service as logical inverse-plugin.

On the other hand, in case of non-empty instance sets of both goal and service constraints as shown in Figure 4, the constraint matching filter detects that the intersection of instance sets $\{t1, t2, t3, t4, t5\}$ and $\{t6\}$ satisfying $c1$, respectively, $c2$ is empty, hence correctly valuates the logical constraint relationship as *fail*. One general solution to this problem would be to apply alternative means of approximating logical implication such as checking of theta-subsumption between logical constraints [32].

Missing input and output parameters tolerated by default. WSMO-MX ignores the fact of missing input or output parameters by default if the user did not specify a respective missing relation strategy by returning logical plugin or inverse-plugin by default (cf. Table 3, "none"). In cases where the logically defined types of these parameters are key for a semantic mismatch between goal and service this default matching leads to false-positives. Similar cases arise when the specified missing relation strategy appears too relaxed (with values "assumeEquivalent or "ignore").

The matching process annotation produced by the parameter and relation matching filter may help the user to overcome this problem by refining the original goal description. The additional use of syntactic matching to mitigate logic-based false-positives is proposed in [25].

5.3.2. Logic-based false-negatives

The relaxation of logic-based matching by means of compensative or complementary syntactic matching can avoid logic-based false negatives. This holds for cases where the ontology modeling is too coarse-grained such that logical type matching would fail. One example are class or type siblings in the ontology with similar real-world semantics and sufficient syntactic similarity.

5.3.3. False-positives of hybrid matching

Syntactic matching can also produce false-positives due to its inherent ignorance of the logical definition of the semantics of service and goal elements to be compared. In fact, the additional (compensative or complementary) use of syntactic matching by WSMO-MX may wrongly override the correct result of its logic-based matching.

6. Related work

Other implemented approaches to WSML service discovery are rare. To the best of our knowledge, these are the logic-based matchmaker GLUE (Della Valle et al., 2005)[8,9] and the syntactic search engine (part of the WSMO studio) for QoS-enabled WSML service discovery in P2P networks (Vu et al., 2006)[34]. In addition, approaches to logic-based semantic matching of so-called rich functional service descriptions (WSML-oriented) in abstract state spaces based on concurrent transaction logic are proposed in (Keller et al., 2005; Stollberg et al., 2006)[18,31], though it is unclear to what extent they have been implemented. WSMO-MX has been the first implemented general and hybrid matchmaker for WSML-oriented services. In particular, other mediator-based discovery approaches such as those presented in [20, 9,26] do not allow for general goal-service matching, but require problem-specific mapping, or construction rules. Hybrid matchmakers for services in formats different from WSML are in particular available for OWL-S such as the matchmakers OWLS-MX [25,24], ROWLS [11], FC-MATCH [3], and iMatcher [15]. Of those only OWLS-MX and iMatcher have been experimentally evaluated against a service retrieval test collection for OWL-S services, called OWLS-TC2. In summary, these results also show that the performance of logic-based only matching of semantic services can be improved by syntactic similarity-based matching. DIANE [22,21] inspired WSMO-MX in terms of its object-graph-driven matching and strategies for declaring mismatch tolerance. However, its graph-matching-based service selection does not perform any logic-based reasoning nor is it related to prominent logic-based semantic service description models like OWL-S or WSML. A comprehensive survey and classification of semantic service matchmakers for different service description formats is provided in [16].

7. Conclusions

The matchmaker WSMO-MX performs hybrid semantic service matching based on both logic programming in F-Logic and syntactic similarity measurement. WSMO-MX applies different matching filters to retrieve services that are semantically relevant to a given query with respect to seven, totally ordered degrees of matching. These degrees are recursively computed by aggregated valuations of ontology-based type matching, logical constraint, recursive relation matching, and syntactic similarity-based matching. WSMO-MX v0.4 is available at the open software portal SemWebCentral and requires a license for using the ontology management system OntoBroker.

The results of our experimental evaluation of the performance of WSMO-MX over an extended version of our initial test collection WSML-TC1, namely WSML-TC2, showed in particular that (a) hybrid semantic matching can outperform logic-based only matching, and that (b) syntactic matching - if parametrized appropriately - can easily keep up with logic-based matching regarding recall/precision, and significantly outperforms it in terms of computation time.

We are currently working on an extension of the test collection WSML-TC2 and further experimental evaluation, as well as an updated version of WSMO-MX with an additional ontology manager plugin for the open-source F-Logic reasoner Flora-2. Furthermore we are preparing a Web demo of WSMO-MX at `http://www.wsmo-mx.net`.

Acknowledgement. We would like to thank Patrick Kapahnke at DFKI for helping with the experimental evaluation and software maintenance of WSMO-MX.

References

[1] J. Angele, G. Lausen: Ontologies in F-logic. In: *Handbook on Ontologies, eds. S. Staab, R. Studer*, Springer, 2004.

[2] R. Baeza-Yates, B. Ribeiro-Neto: *Modern Information Retrieval*. ACM Press, Addison-Wesley. pages 75ff, 1999.

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

[4] A. Bernstein, C. Kiefer: Imprecise RDQL: Towards Generic Retrieval in Ontologies Using Similarity Joins. *Proceedings ACM Symposium on Applied Computing*, Dijon, France, ACM Press, 2006.

[5] L. Botelho, A. Fernandez, B. Fries, 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. Birkhäuser Verlag, Springer, 2008.

[6] Natalia Cherchago, Pascal Hitzler, Steffen Hölldobler Decidability Under the Well-Founded Semantics In Massimo Marchiori, Jeff Z. Pan, Christian de Sainte Marie, *Proceedings of the First International Conference on Web Reasoning and Rule Systems, RR2007*, Innsbruck, Austria, June 2007, volume 4524 of Lecture Notes in Computer Science, pp. 269-278. Springer, June 2007.

[7] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov: Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3), 2001.

[8] E. Della Valle, D. Cerizza, I. Celino: The Mediators Centric Approach to Automatic Web Service Discovery of GLUE. *Proceedings of 1st International Workshop on Mediation in Semantic Web Services (MEDIATE)*, CEUR Workshop proceedings, 168, 2005.

[9] E. Della Valle, D. Cerizza: COCOON Glue: a prototype of WSMO Discovery engine for the healthcare field. *Proceedings of the WIW 2005 Workshop on WSMO Implementations*, CEUR Workshop Proceedings, 134, 2005.

[10] D. Fensel, F. van Harmelen: Unifying reasoning and search to Web scale. *IEEE Internet Computing*, March/April 2007.

[11] A. Fernandez, M. Vasirani, C. Caceres, S. Ossowski: A role-based support mechanism for service description and discovery. In: Huang et al. (eds.), *Service-Oriented Computing: Agents, Semantics, and Engineering*. LNCS 4504, Springer, 2007.

[12] G. Gottlob, A. Leitsch: On the efficiency of subsumption algorithms. *Journal of the ACM*, 32(2), 1985.

[13] C.A.R. Hoare: An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), 1969.

[14] F. Kaufer, M. Klusch: WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker. *Proceedings of the 4th IEEE European Conference on Web Services (ECOWS 2006)*, IEEE CS Press, Zurich, Switzerland, 2006.

[15] C. Kiefer, A. Bernstein: The Creation and Evaluation of iSPARQL Strategies for Matchmaking. *Proceedings of European Semantic Web Conference*, Springer, 2008.

[16] M. Klusch: Semantic Web Service Coordination. In: M. Schumacher, H. Helin, H. Schuldt (Eds.), *CASCOM - Intelligent Service Coordination in the Semantic Web*. Chapter 4. Birkhäuser Verlag, Springer, 2008..

[17] M. Klusch, F. Kaufer: Performance of Hybrid WSML Service Matching with WSMO-MX: Preliminary Results. *Proceedings of the 1st Intl. Joint Workshop on Semantic Matchmaking and Resource Retrieval, at Intl. Semantic Web Conference*, Busan, Korea, 2007.

[18] U. Keller, R. Lara, H. Lausen, A. Polleres, D. Fensel: Automatic Location of Services. *Proceedings of the 2nd European Semantic Web Conference (ESWC)*, Heraklion, Crete, LNCS, 3532, Springer, 2005.

[19] M. Kifer, G. Lausen, J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4), 1995.

[20] M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, D. Fensel: A Logical Framework for Web Service Discovery. *Proceedings of the ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*, CEUR Workshop Proceedings, 119, 2004.

[21] U. Küster, B. König-Ries, M. Stern, M. Klein: DIANE: an integrated approach to automated service discovery, matchmaking and composition *Proceedings of the 16th International Conference on World Wide Web, WWW 2007*, Banff, Alberta, Canada, May 8-12, 2007. ACM 2007.

[22] M. Klein, B. König-Ries: Coupled Signature and Specification Matching for Automatic Service Binding. *Proceedings European Conference on Web Services (ECOWS 2004)*, Erfurt, 2004.

[23] M. Klusch, K. Sycara: Brokering and Matchmaking for Coordination of Agent Societies: A Survey. In: *Coordination of Internet Agents: Models, Technologies and Applications*. Springer, 2001.

[24] M. Klusch, B. Fries, P. Kapahnke: Hybrid Semantic Web Service Retrieval: A Case Study with OWLS-MX. *Proceedings of 2nd IEEE International Conference on Semantic Computing*, USA. IEEE Press, 2008.

[25] M. Klusch, B. Fries, K. Sycara: Automated Semantic Web Service Discovery with OWLS-MX. *Proc. 5th Intl. Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Hakodate, Japan, ACM Press, 2006

[26] R. Lara, M. Angel Corella, P. Castells: A flexible model for Web service discovery. *Proceedings of the 1st International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives*, Seoul, South Korea, 2006.

[27] , J. Lloyd and R. Topor: Making Prolog More Expressive. *Journal of Logic Programming*, 1(3), pp. 225-240, 1984.

[28] S. Muggleton and L. De Raedt: Inductive Logic Programming: Theory and Applications. *Logic Programming*, 19(20), 1994.

[29] J.A. Robinson: A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1), 1965.

[30] T. Scheffer, R. Herbrich, F. Wysotzki: Efficient Algorithms for theta-Subsumption. *Inductive Logic Programming, 6th International Workshop, Selected Papers*, LNAI 1314, pp. 212-228, Springer Verlag Berlin, 1996.

[31] M. Stollberg, U. Keller, H. Lausen, S. Heymans: Two-phase web service discovery based on rich functional descriptions. *Proceedings of European Semantic Web Conference*, Buda, Montenegro, LNCS, Springer, 2007.

[32] K. Sycara, S. Widoff, M. Klusch, J. Lu: LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2), 2002.

[33] A. van Gelder, K. Ross, J. S. Schlipf: The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620-650, 1991.

[34] L.H. Vu, M. Hauswirth, F. Porto, K. Aberer: *A Search Engine for QoS-enabled Discovery of Semantic Web Services*. Ecole Politechnique Federal de Lausanne, LSIR-REPORT-2006-002, Switzerland, 2006. Also available in the Special Issue of the International Journal of Business Process Integration and Management (IJBPIM) (2006).

[35] G. Yang and M. Kifer: Well-Founded Optimism: Inheritance in Frame-Based Knowledge Bases. *Proceedings of 1st International Conference on Ontologies, Databases and Applications of Semantics (ODBASE)*, Irvine, California, 2002.

[36] A. Zaremski, J. Wing: Specification Matching of Software Components. *Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 1995.