Performance of Hybrid WSML Service Matching with WSMO-MX: Preliminary Results

Frank Kaufer[1] and Matthias Klusch[2]

[1] Hasso-Plattner-Institute at University of Potsdam
Prof.-Dr.-Helmert-Strasse 2-3, Potsdam,
frank.kaufer@hpi.uni-potsdam.de
[2] German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, Saarbrücken
klusch@dfki.de

Abstract. The WSMO-MX matchmaker applies different matching filters to retrieve WSMO-oriented service descriptions that are semantically relevant to a given query with respect to seven degrees of hybrid matching. These degrees are recursively computed by aggregated valuations of ontology based type matching, logical constraint and relation matching, and syntactic similarity as well. In this paper, we provide preliminary results of our experimental evaluation of the performance of WSMO-MX. In summary, it turned out that hybrid matching of WSML-MX services performs reasonably well.

1 Introduction

The problem of efficiently retrieving relevant services in the envisioned semantic web has been solved so far by only a few approaches for services described in OWL-S such as [1, 2], and WSML such as [3, 4, 13]. Though, existing proposals for rule based service mediation in WSMO do not provide a general purpose matchmaking scheme for services in WSML. Recently, this gap has been filled by a hybrid semantic matchmaker, called WSMO-MX, that applies different matching filters to retrieve extended WSML services that are semantically relevant to a given query including the goal to be satisfied [5].

For this purpose, both services and goals are described in a Logic Programming (LP) variant of WSML, called WSML-MX, which is based on WSML-Rule. The hybrid matching scheme of WSMO-MX combines and extends the ideas of hybrid semantic matching realized by OWLS-MX [2], the object-oriented structure based matching proposed by Klein & König-Ries [6], and the concept of intentional matching introduced by Keller et. al [7]. WSMO-MX v0.4 is available at http://projects.semwebcentral.org/projects/wsmomx/.

In this paper, we build upon this work and show the results of our experimental evaluation of the performance of WSMO-MX based on a first, admittedly small service retrieval test collection for WSML services derived from the DIANE test collection.

The remainder of this paper is structured as follows. Section 2 provides an overview on how WSMO-MX works, while the testing environment and the preliminary results of the evaluation of its retrieval performance is given in section 3. Related work is strived in section 4, and section 5 concludes this paper.

2 WSMO-MX Overview

In this section, we briefly summarize the functionality of the WSMO-MX matchmaker and provide a brief example. For further details on the functionality and implementation of WSMO-MX, we refer to [5].

2.1 Service description in WSML-MX

WSMO-MX pairwisely matches services in a formally grounded variant of WSML called WSML-MX directly in F-Logic [8, 9]. It adopts the main and clearly motivated elements required for service matching from WSML, that are *goal*, *service*, *capabilities*, *preconditions*, and *postconditions* but not $effect$ and *assumption*. Central to describing services in WSML-MX is the notion of *derivative* which is an extended version of the object set introduced by Klein and König-Ries [6].

A derivative D_T in WSML-MX encapsulates an ordinary concept T (in this context called type) defined in a given ontology by attaching meta-information mainly about the way how T can be matched with any other type. Such information is defined in terms of different meta-relations of the derivative D_T. The type T is defined to be either atomic or a complex type with relations, the derivative D_T can also have a set of relations different from T, though this set is empty by default. 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 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.

This variant of WSML allows for constraints on both relations and derivatives formulated in the full Horn fragment of F-logic. Hence, WSML-MX constraints are as expressive and, in general, only semi-decidable as are WSML-Rule axioms. However, the WSMO-MX matchmaker approximates query containment through means of relative query containment for constraint matching. 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 modelling is motivated not only by clear separation of computationally tractable elements but the fact that it allows the matchmaker for a more detailed explanatory feedback to the user and more differentiated matching valuations. This is a lesson learned from 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.

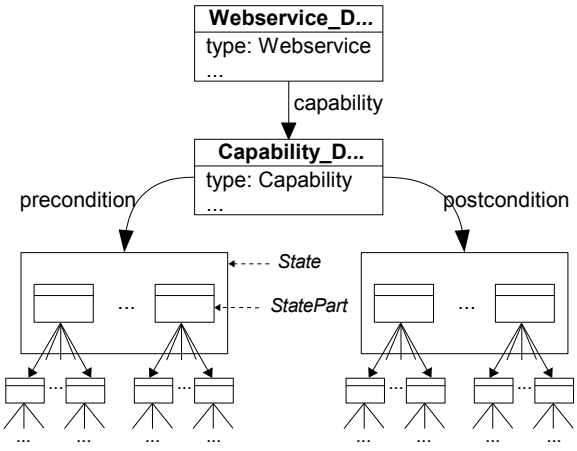


Fig. 1. Service derivative in WSML-MX

An example for a service in WSML-MX is shown in figure 2; the service offers tickets for any trip between any two German towns, but if the user departs from Berlin, her destination must be Hamburg.

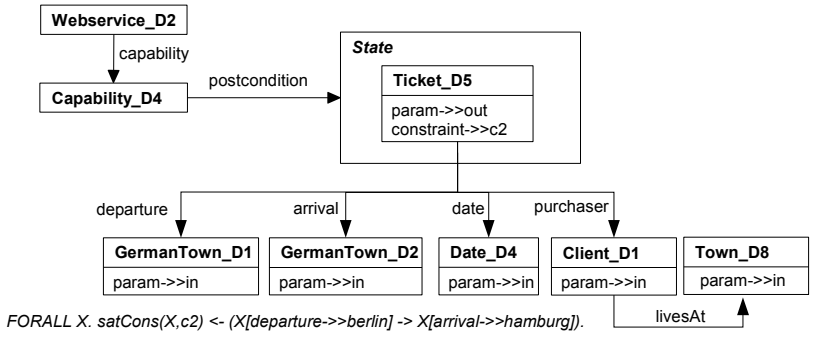


Fig. 2. Example service in WSML-MX

2.2 Hybrid 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 valuations of (a) ontology based type matching, (b) logical constraint matching, (c) recursive relation matching, and (d) syntactic matching. In this respect, the matching of WSMO-MX is hybrid.

Each real-valued entry in the so called (matching) valuation vector $v = (\pi_{\equiv}, \pi_{\sqsubseteq}, \pi_{\sqsupseteq}, \pi_{\sqcap}, \pi_{\sim}, \pi_{\circ}, \pi_{\bot})$ with $\pi_i \in [0,1]$ $(i \in \{\equiv, \sqsubseteq, \sqsupseteq, \sim, \sqcap, \circ, \bot\})$ and $\sum \pi_i = 1$, denotes the extent (similarity score) to which both derivatives D_G and D_W match with respect to the hybrid semantic matching degrees π_i of WSMO-MX. These degrees are the logical relations *equivalence*, $plug-in$ known from software component retrieval [10] or the similar *rule of consequences* from Hoare logic [11], $inverse-plugin$, $intersection$ and $disjunction$ $(fail)$ as degrees of *logic based* semantic match.

The degree of $fuzzy\,similarity$ refers to a non-logic based semantic match such as syntactic similarity or non-subconcept relations with respect to the type graph, while the degree *neutral* stands for neither match nor fail, hence declares the tolerance of matching failure. The set-theoretic semantics of the hybrid matching degrees are given in Table 1 based on the relations between the maximum possible instance sets of the derivatives D_G and D_W, denoted by $\mathcal{G}$ and $\mathcal{W}$. Since we use the heuristic relative query containment for the constraint matching, these sets are restricted to instances in the matchmaker knowledge base which satisfy the constraints. We acknowledge that it can not be taken for granted that the matchmaker is in possession of instances for every derivative with assigned constraints. However, these precedence instances could be retrieved by tracking service executions, sampling services/descriptions (services without real world effects) or - regarding goal derivative instances - by conducting systematic questionnaires. Furthermore, constraint matching could be ignored (configuration option in WSMO-MX) for coarse service discovery and applied for verification purposes in the service composition, when instances are available.

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	$\bot$	disjunction (fail)	$\mathcal{G} \cap \mathcal{W} = \emptyset$	

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

2.3 Hybrid matching process

In order to compute the degrees of hybrid semantic matching of given goal and service derivatives in WSML-MX, WSMO-MX recursively applies different IOPE matching filters 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 [6]), and returns not only the aggregated matching

valuation vector but also 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. The annotations have a generic format and can be employ for several purposes. In the current version, WSMO-MX uses them to generate natural language explanations of the respective matching deviations. In the future they 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 relation range derivatives of equally named 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 matched state parts.

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 (type, constraint, and relation matching) is applied. While type matching, in essence, bases on the subconcept relations and path distance between types (classes) in the matchmaker ontology, F-logic constraint matching is computed by means of relative query containment, and 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 exclusively syntactic matching without any semantic matching.

Finally, all valuation vectors computed during recursive matching of goal and service derivatives are aggregated into one single valuation vector. For aggregation, each individual valuation vector is weighted for the respective matching filter as specified by the user for the given goal; the weighting is assumed to be equal by default. This aggregated valuation of hybrid matching degrees is then recomputed with respect to the intentions of the considered derivatives.

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

2.4 Example

Goal, service, ontology. Suppose the user defines a goal 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.
The part of the type hierarchy in the matchmaker ontology and all instances used in this example are shown in figure 4.

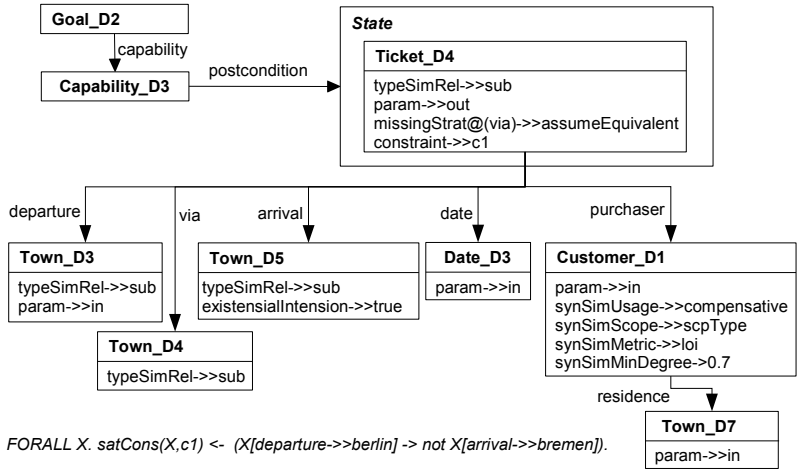
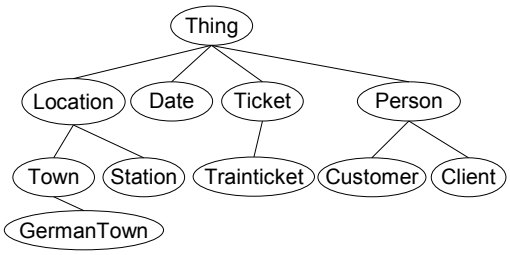


Fig. 3. Example goal in WSML-MX

In this example, the service derivative *Ticket_D5* given in section 2 will be matched against the goal derivative *Ticket_D4* as follows.

Matching. Since the capabilities of both goal and service derivatives do not include any precondition, the hybrid semantic matching of them is restricted to the matching of their postcondition states.

1. **match types**: the types of Ticket_D4 and Ticket_D5 are equal. Hence the valuation is $v_1 = (1, 0, 0, 0, 0, 0, 0)$.
2. **match parameters**: both are output parameters, no annotation necessary
3. **match relations**
   (a) *departure*: the types of Town_D3 and GermanTown_D1 are not equivalent, but Town_D3 allows subtypes. Since GermanTown is a subconcept of Town, the valuation is $v_2 = (0, 1, 0, 0, 0, 0, 0, 0)$.
   (b) *via*: this relation is not defined for Ticket_D3, but the missingStrategy for this relation is *assumeEquivalent* yielding a valuation $v_3 = (1, 0, 0, 0, 0, 0, 0, 0)$.
   (c) *arrival*: analogous to *departure* types of the ranges of *arrival* are subtypes and yield the valuation $v_4 = (0, 1, 0, 0, 0, 0, 0, 0)$.
   (d) *date*: is equal in goal and service, hence valuated as $v_5 = (1, 0, 0, 0, 0, 0, 0, 0)$
   (e) *purchaser*: type matching fails for Customer_D1 and Client_D1, but compensative syntactic matching is allowed using loss of information (LOI) metric. For the unfolding only the types of the derivatives should be used (*scpType*), yielding the term vectors $(Customer : 1, Town : 1, Person : 1, Location : 1, Town : 1)$ and $(Client : 1, Town : 1, Person : 1, Location : 1, Town : 1)$ for Customer_D1 and Client_D1,




```
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)

respectively. The similarity degree is 0.75, and therefore greater than the declared minimum of 0.7. The resulting valuation vector is $v_6 = (0, 0, 0, 0, 0, 0, 1, 0)$.

The aggregated relation valuation is $v_7 = \frac{v_2+...+v_6}{5} = (0.4, 0.4, 0, 0, 0, 0.2, 0)$

4. **match constraints**: Ticket_D4 has the constraint c1. This is satisfied by the instances $t1, \ldots, t5$. The constraint c2, which is imposed on Ticket_D5 is satisfied by the instances $t3, \ldots, t5$. That means the instances for Ticket_D5 are a subset of those of Ticket_D4 and hence the valuation is $v_8 = (0, 1, 0, 0, 0, 0, 0, 0)$

Finally, the aggregated valuation for the derivative matching of Ticket_D4 and Ticket_D5 is

$$v_9 = \frac{v_1+v_7+v_8}{3} = (\frac{7}{15}, \frac{7}{15}, 0, 0, 0, \frac{1}{15}, 0).$$

This means that the advertised service is hybrid semantically matching with the request. In particular, they are exactly and plug in matching to the same extent (0.46).

3 Evaluation of performance

The preliminary experimental evaluation of the retrieval performance of WSMO-MX focuses on measuring its recall, precision, and F1 based on an initial test collection.

3.1 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 had to develop an inital test collection.

Service Retrieval Test Collection WSML-TC1. For this purpose, we borrowed domain ontologies, service offers and requests from the DIANE project[3], and transformed parts of them from their F-DSD format into WSML-MX F-Logic. The resulting test collection WSML-TC1 contains approximately 300 concepts and over 800 instances in the domain ontology and 27 web services (offers) and 21 goals (requests) with over 1000 derivatives.

The goals belong to 5 different domains: book buy (10), cinema ticket booking (4), tv set buy (5), printing request (3), train ticket service (5). They are chosen such that they have large modeling overlaps making it more challenging for syntactical and hybrid matching approaches which rely on syntactic similarity. As usual, the relevance sets of WSML-TC1 were subjectively determined, that is whether a service is relevant for a request depends on a categorical point of view. That is motivated in part by the fact that semantic service matching shall be employed in a wide range of use case scenarios, from crisp composition planning to human oriented service discovery making it impossible for a general purpose matchmaker like WSMO-MX to take every possible prerequisite (e.g. available inputs/mediators, intended usage, etc) into account.

For the formal declaration of relevance in test environments, we define two further derivative meta relations: `declaredRelevanceTo` and `numberDeclaredRelevant`. The first one is used to assign a goal derivative with a service derivative, the latter counts all relevant web service derivatives by means of the following rule:

```
FORALL D,C D[numberDeclaredRelevant->>C] <-
      EXISTS R D[declaredRelevanceTo->>R] AND count(D,R,C).
```

Hardware. For the performance tests, WSMO-MX v0.5 and OntoBroker v4.3 were deployed on a machine with Win XP, Apache Tomcat 6 (also tested with 5.5), Java 6 (also tested with Java 5), 2.39 GHz, and 2 GB RAM.

3.2 Experiments

On the basis of the very first and admittedly small test collection WSML-TC1, we initially conducted six experiments to investigate the matchmakers behavior with respect to different configurations for semantic, syntactic, and hybrid matchings. We measured the retrieval performance of WSMO-MX in terms of its recall, precision, and F1-measure as known from information retrieval [12]. The unfolding of derivatives for the syntactic matching is done online, results are indexed only for one matching request, such that overall time needs are

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

comparable between syntactic and non-syntactic matchings. For non-testing environments at least all offers could be indexed in advance of the user request which reduces the effective matching time to some milliseconds.

Experiment 1: Pure logic based semantic matching At first we examined how well pure semantic matching performs, relaxing only derivative type deviations in three steps of increasing degree of fuzzyness as follows.

- *default*: Only type deviations explicitly granted in the goal descriptions are allowed
- *subSuper*: All service derivatives without explicitly defined type deviation are considered sufficient similar to each other if their types have a logical subconcept relationship
- *relation-3*: Types are only required to have a maximum distance of 3 in the taxonomy graph spanned by logical subconcept relation.

The strict configurations *default* and *subSuper* deliver solely relevant results. While the default configuration only achieves a recall of below 0.7 (Fig. 5), *subSuper* already delivers a full recall which is due to the fact that the test collection relies on one homogenous domain ontology. Only the fuzzier *relative-3* configuration attains 100% recall, but at the expense of a final 70% fallout and a time consumption of almost 1 sec per query which is similar to 0,89 sec per query for *subSuper*, but more than three times as much as needed for *default* matching (0,3 sec).

Experiment 2: Syntactic similarity metrics This experiment is about comparing the four provided metrics for syntactic matching: Jaccard, Extended-Jaccard, Multiset-Jaccard, and Cosine. Therefore the configuration was fixed to alternative syntactic matching with a minimum matching degree of 0.6. For the unfolding relations and types were considered up to a maximum depth of 2.

It could be observed that for the chosen threshold all metrics except Cosine deliver 100% or almost 100% correct results. But only Jaccard and Multiset-Jaccard also achieve a high recall of over 90%. Extended-Jaccard's recall is below 0.6, but additional variations showed that this metric is the least vulnerable with respect to precision. In total, figure 6 shows that Jaccard provides the best overall results. In this configuration, Multiset-Jaccard is only slightly worse.

It is remarkable that the syntactic matching achieves full precision and almost the same recall level as the pure semantic matching while consuming only a sixth of the overall computation time. Furthermore the most computation is needed for unfolding and index generation which could be done in advance of the matching. But please note that the computation time of the semantic matching is largely determined by the client-/server communication of the matchmaker and the reasoner. With the new reasoner API of OntoBroker 5 this can be reduced significantly and will be investigated in detail in future evaluations.

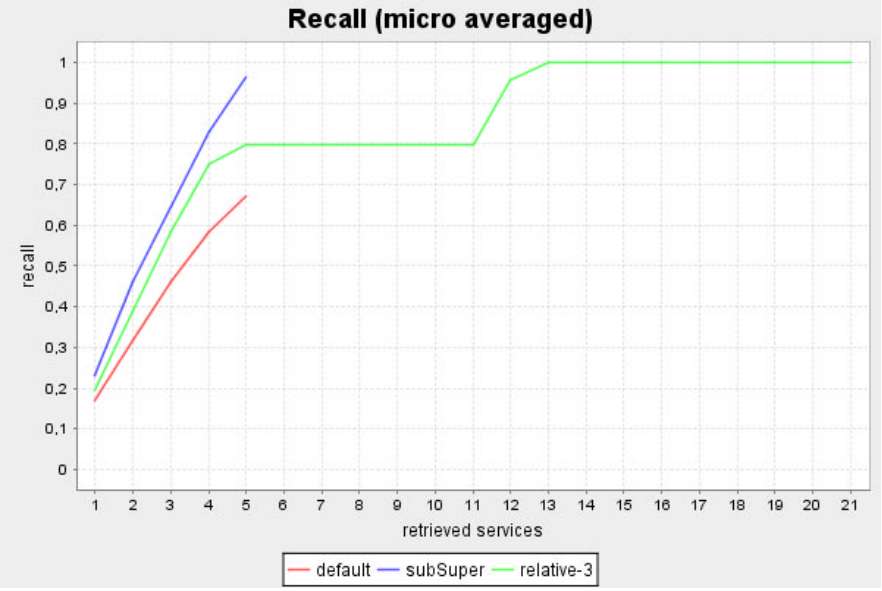


Fig. 5. Type matching

Experiment 3: Syntactic similarity threshold The minimal matching degree is crucial for WSMO-MX because syntactic matching is applied to single derivatives instead of whole service descriptions (like in OWLS-MX [2]). To find a suitable threshold, we used the best configuration from the last experiment (metric: Jaccard) and increased the threshold successively from 0.4 to 0.7 in steps of 0.05. Beginning with 0.6 the syntactic matching delivers only relevant documents, whereas 0.4 and 0.45 entail a final fallout of about 60% (50%) for their final but late full recall. A threshold of 0.6 achieved still a recall of 0.9 (0.65 - 80%, 0.70 - 70%). The results for the thresholds 0.5 and 0.55 are closer to those of 0.6-0.7 with a fallout of 10-20% and almost full recall. For this test collection 0.6 can be considered as most suitable threshold. For ontologies of more heterogeneous origin probably a lower threshold should be chosen.

Experiment 4: Unfolding service derivative scope In this experiment the scope of a derivative used for syntactic matching is varied. For the last experiments, both, types and relation were used. Now, this configuration is compared to only using types or relations. As can be seen from the F1-measure graph in figure 7, only combination of both leads to the best result. Separate consideration of recall and precision shows that types determine the recall (100% but final fallout of 40%) and the relations the precision (almost 100%, but recall only 50%). Syntactic matching with an unfolding of relations *and* types needs twice as much time as with only one of both scopes.

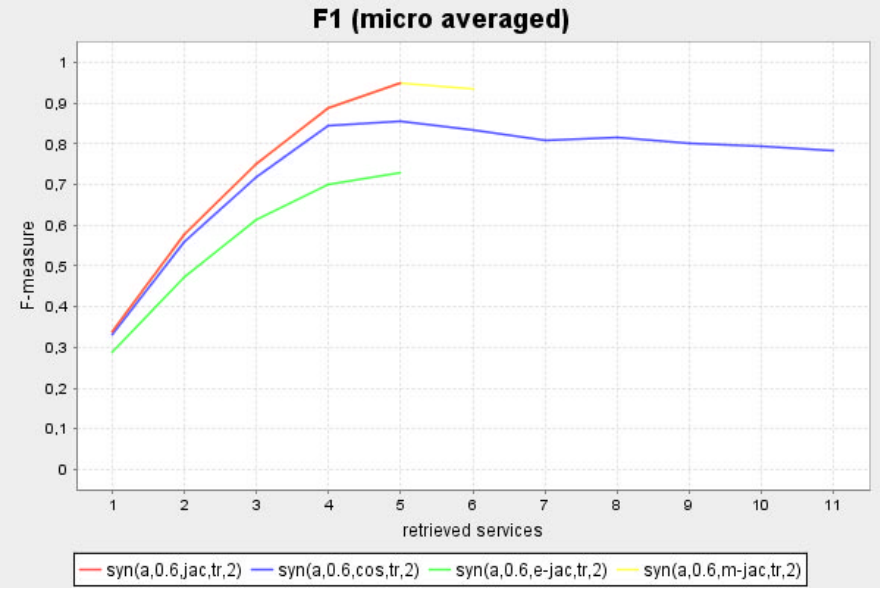


Fig. 6. Metrics experiment: Jaccard, Cosine, Extended Jaccard, Multiset Jaccard

Experiment 5: Unfolding depth Beside the scope the unfolding result of a derivative is dependent from the maximum unfolding depth. In this experiment we increased the depth starting with 0 which means that only the derivative itself is unfolded. From unfolding depth 3 on, no significant changes could be observed. Figure 8 clearly shows that only the unfolding depths 2 and 3 yield good results. Although the depths 0 and 1 achieve a final full recall, the high fallout of 80% (55%) is not acceptable. Unfolding (and matching) for depth 2 takes twice as much time as for 1 which in turn takes twice as much as for depth 0.

Experiment 6: Hybrid vs logic based matching Finally, we compared the performance of hybrid matching based on integrated and compensative syntactic matching with best syntactic only matching as well as *default* and *subSuper* matching from the first experiment. All four matchings have no fallout and only differ with respect to recall (figure 9) and computation time. Though not significant, the hybrid matching performed best, but only slightly better than *subSuper* and the pure syntactic matching. However, on average, the syntactic matching takes only 0,15 sec per query, whereas hybrid matching takes 0,61 and *subSuper* 0,89. Overall, this indicates the usefulness of syntactic matching in combination with, or instead of logical reasoning.

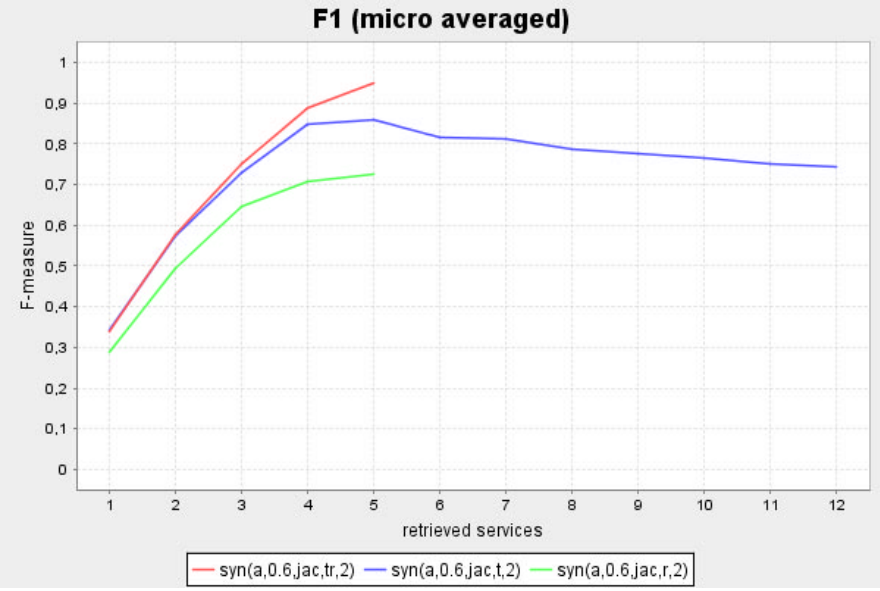


Fig. 7. Syntactic matching with different scopes: types (t), relations (r), both (tr)

4 Related work

WSMO-MX has been the first implemented full-fledged matchmaker for WSMO-oriented services. The mediator based discovery approaches and discovery models for WSML presented, for example, in [3, 4, 13] do not allow for general goal-service matching, but require problem specific mapping, or construction rules.

In general, semantic service matching determines whether the abstract semantic description of a requested service (or goal) conforms to that of an advertised service based on their functional and non-functional semantics. This is at the very core of any semantic service discovery framework. Current approaches to semantic service matching can be classified according to

- what kinds and parts of service semantics are considered for matching, and
- how matching is actually performed in terms of syntactic similarity measurements, logic based reasoning within or partly outside the service description framework, or a hybrid combination of both

Most matchmakers for the semantic Web today perform service profile matching while only very few perform process model matching or even a combined functional service matching. Since all semantic Web service description frameworks, except SAWSDL, provide a strict logic based profile semantics (IOPE), it comes at no surprise that most matchmakers rely on crisp logic based rather than hybrid semantic matching of semantic Web services of which only a few approaches exist. Process matching approaches or even combined approaches are more than rare leaving a lot of space for further research.

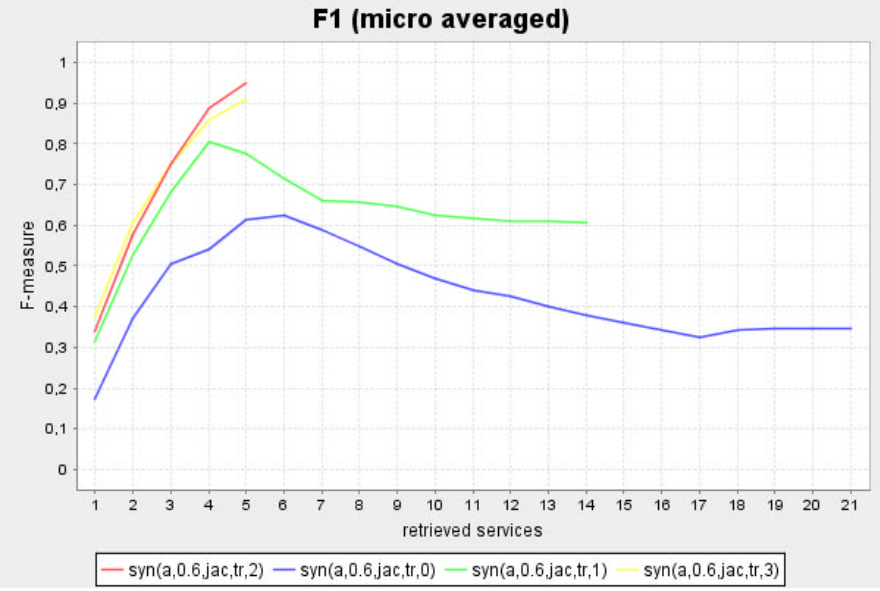


Fig. 8. Influence of unfolding depth

For WSMO-MX, we did improve on the idea of hybrid matching that has been applied to OWL-S service matching by OWLS-MX [2] and in general by LARKS [14] through allowing for a more fine-grained parametrisation, and integrated interleaving of syntactic and semantic matching. In any case, the lack of a sufficiently large and commonly agreed test collection for evaluating the performance of semantic Web service matchmakers for any of the current description frameworks is a general problem which can only be tackled by the community as a whole. In this respect, the presented results of the performance evaluation of WSMO-MX can only be considered preliminary.

5 Conclusions

WSMO-MX performs hybrid semantic web service matching based on both logic programming, and syntactic similarity measurement. It applies different matching filters to retrieve WSMO-oriented service descriptions that are semantically relevant to a given query with respect to seven degrees of hybrid matching. These degrees are recursively computed by aggregated valuations of ontology based type matching, logical constraint and relation matching, and syntactic similarity as well. WSMO-MX v0.4 is available at http://projects.semwebcentral.org/projects/wsmomx/. In this paper, we provided preliminary results of our experimental evaluation of the performance of WSMO-MX. In summary, it turned out that hybrid matching of WSML-MX services performs reasonably well, and can outperform crisp logic based match-

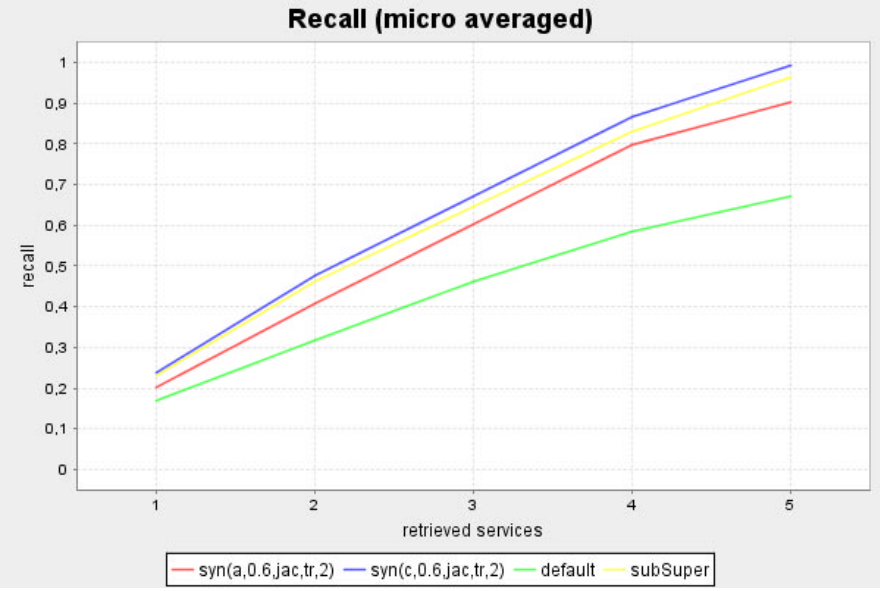


Fig. 9. Hybrid (with syntactic similarity) vs. pure logic based (default, subsumption) matching

ing. Furthermore, the experiments indicated that pure syntactic matching - if parametrized appropriately - can keep up with logic matching regarding recall/ precision and significantly outperforms it with respect to computation time. We are currently working on extending the test collection WSML-TC1, and the updating of WSMO-MX for an upcoming release.

References

1. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. Journal of Web Semantics **1**(1) (2003) 28
2. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with owls-mx. In: Proceedings of 5th International Conference on Autonomous Agents and multiagent Systems AAMAS, Hakodate, Japan (2006)
3. Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., Fensel, D.: A logical framework for web service discovery. In: Proceedings of the ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications. Volume 119., Hiroshima, Japan, CEUR Workshop Proceedings (November 2004)
4. Valle, E.D., Cerizza, D.: Cocoon glue: a prototype of wsmo discovery engine for the healthcare field. In: Proceedings of the WIW 2005 Workshop on WSMO Implementations. Volume 134., Innsbruck, Austria, CEUR Workshop Proceedings (June 2005)

5. Kaufer, F., Klusch, M.: 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)
6. Klein, M., König-Ries, B.: Coupled signature and specification matching for automatic service binding. In: Proceedings of European Conference on Web Services (ECOWS 2004). LNCS 3250, Erfurt, Germany, Springer (September 2004) 183
7. Keller, U., Lara, R., Lausen, H., Polleres, A., Fensel, D.: Automatic location of services. In: Proceedings of the 2nd European Semantic Web Symposium (ESWS2005), Heraklion, Crete (June 2005)
8. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM **42** (1995)
9. Angele, J., Lausen, G.: Ontologies in f-logic. In Staab, S., Studer, R., eds.: Handbook on Ontologies. Springer (2004) 29–50
10. Zaremski, A.M., Wing, J.M.: Specification matching of software components. In: 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering. (10 1995)
11. Hoare, C.: An axiomatic basis for computer programming. Communications of the ACM (CACM) **12**(10) (10 1969) 576–580
12. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. ACM Press, Addison-Wesley. pages 75ff, ISBN 0-201-39829-X (1999)
13. Lara, R., Corella, M.A., Castells, P.: A flexible model for web service discovery. In: Proceedings of the 1st International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives, Seoul, South Korea. (2006)
14. Sycara, K., Widoff, S., Klusch, M., Lu, J.: Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and Multi-Agent Systems **5** (June 2002) 173–2003