SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition

Rama Akkiraju[1], Biplav Srivastava[2], Anca-Andreea Ivan[1], Richard Goodwin[1], Tanveer Syeda-Mahmood[3]

[1]IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY, 10532, USA
[2]IBM India Research Laboratory, Block 1, IIT Campus, Hauz Khaus, New Delhi, 11016, India
[3]IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA
{akkiraju@us, sbiplav@in, ancaivan@us, rgoodwin@us, stf@almaden}.ibm.com

Abstract

Composing existing Web services to deliver new functionality is a difficult problem as it involves resolving semantic, syntactic and structural differences among the interfaces of a large number of services. Unlike most planning problems, it can not be assumed that Web services are described using terms from a single domain theory. While service descriptions may be controlled to some extent in restricted settings (e.g., intra-enterprise integration), in Web-scale open integration, lack of common, formalized service descriptions prevent the direct application of standard planning methods. In this paper, we present a novel algorithm to compose Web services in the presence of semantic ambiguity by combining semantic matching and AI planning algorithms. Specifically, we use cues from domain-independent and domain-specific ontologies to compute an overall semantic similarity score between ambiguous terms. This semantic similarity score is used by AI planning algorithms to guide the searching process when composing services. Experimental results indicate that planning with semantic matching produces better results than planning or semantic matching alone. The solution is suitable for semi-automated composition tools or directory browsers.

Introduction

In implementing service-oriented architectures, Web services are becoming an important technological component. Web services matching and composition has become a topic of increasing interest in the recent years with the gaining popularity of Web services. Two main directions have emerged. The first direction investigated the application of AI planning algorithms to compose services (Ponnekanti, Fox 2002), (Traverso, Pistore 2004), Synthy (Agarwal et al 2005). The second direction explored the application of information retrieval techniques (Syeda-Mahmood 2004). However, to the best of our knowledge, these two techniques have not been combined to achieve compositional matching in the presence of inexact terms, and thus improve recall.

In this paper, we present a novel approach to compose Web services in the presence of semantic ambiguity using a combination of semantic matching and AI planning algorithms. Specifically, we use domain-independent and domain-specific ontologies to determine the semantic similarity between ambiguous concepts/terms. The domain-independent relationships are derived using an English thesaurus after tokenization and part-of-speech tagging. The domain-specific ontological similarity is derived by inferring the semantic annotations associated with Web service descriptions using an ontology. Matches due to the two cues are combined to determine an overall similarity score. This semantic similarity score is used by AI planning algorithms in composing services. By combining semantic scores with planning algorithms we show that better results can be achieved than the ones obtained using a planner or matching alone.

In the remainder of the paper, we start with a scenario to illustrate the need for Web services composition in open business domains and discuss how our approach can help in resolving the semantic ambiguities better. We then give details of the SEMAPLAN system and demonstrate feasibility of the solution with select experimental results. The longer version of the paper is available in (Akkiraju et al., 2006).

A Motivating Scenario

In this section, we present a scenario from the knowledge management domain to illustrate the need for (semi) automatic composition of Web services. For example, if a user would like to identify names of authors in a given

document, text annotators such as a *Tokenizer*, which identifies tokens, a *LexicalAnalyzer*, which identifies parts of speech, and a *NamedEntityRecognizer*, which identifies references to people and things etc. could be composed to meet the request. Figure 1 summarizes this composition flow. Such dynamic composition of functionality, represented as Web services, saves tedious development time in domains such as life sciences since explicating all possible and meaningful combinations of annotators can be prohibitive. AI Planning algorithms are well suited to generate these types of compositions. However unlike most planning problems, in business domains often it can not be assumed that Web services are described using terms from a single domain theory.

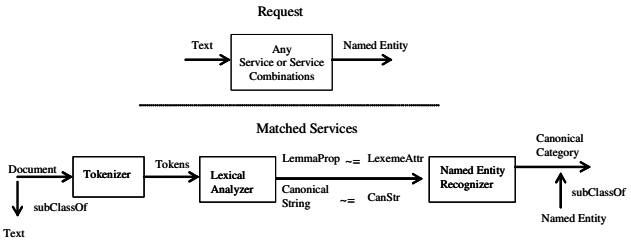

Figure 1. Text Analysis Composition example with semantic matching (~= illustrates semantic match)

For example, the term *lexemeAttr* may not match with l*emmaProp* unless the word is split into l*exeme* and *Attr* and matched separately. Using a linguistic domain ontology one can infer that *lemma* could be considered a match to the term *lexeme*. Abbreviation expansion rule can be applied to the terms *Attr* and *Prop* to expand them to *Attribute* and *Property*. Then a consultation with a domain-independent thesaurus such as WordNet (Miller 1983) dictionary can help match the term *Attribute* with *Property* since they are listed as synonyms. Putting both of these cues together, one can match the term *lexemeAttrib* with *lemmaProp*. In the absence of such semantic cues, two services that have the terms *lexemeAttrib* and *lemmaProperty* as part of their effects would go unmatched during planning thereby resulting in fewer results which adversely impacts recall. In the next section, we explain how we enable a planner to use these cues to resolve semantic ambiguities in our system - SEMAPLAN.

Combining Semantic Matching AI with Planning for Web Service Composition

Figure 2 illustrates the components and the control flow in SEMAPLAN system. Details of SEMAPLAN system are described below.

Service Representation: Service representation involves preparing Web Services with semantic annotations and readying the domain dependent and independent ontologies. We use WSDL-S (Akkiraju et al., 2005) to annotate Web Services written in WSDL with semantic concepts from domain ontologies that are represented in OWL (OWL 2002).

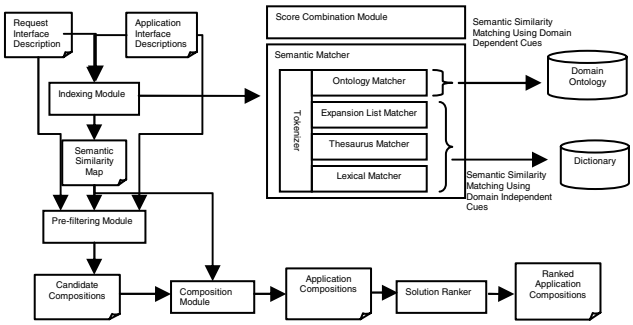

Figure 2. SEMAPLAN system and its components

Term Relationship Indexing: After semantic annotation, the available Web services in the repository are parsed, processed and an efficient index is created consisting of related terms/concepts referred to in the service interface descriptions for easy lookup. This is achieved using the services of a *semantic matcher* which uses both domain-independent and domain-specific cues to discover similarity between application interface concepts. Below we explain how the semantic matcher works.

For finding related terms using domain independent ontologies, we use techniques similar to the one in (Dong 2004). Specifically, multi-term query attributes are parsed into tokens. Part-of-speech tagging and stop-word filtering is performed. Abbreviation expansion is done for the retained words if necessary, and then a thesaurus is used to find the similarity of the tokens based on synonyms. The resulting synonyms are assembled back to determine matches to candidate multi-term word attributes of the repository services after taking into account the tags associated with the attributes. More details are in (Syeda-Mahmood et al., 2005). For finding related terms using domain-specific ontologies, we use relations such as *subClassOf(A,B), subClassOf(B,A)*, *typeOf (A,B)*, and *equivalenceClass(A,B)* in domain ontologies represented in OWL (OWL 2002). We use a simple scoring scheme to compute distance between related concepts in the ontology. subClassOf, typeOf, are given a score of 0.5, equivalentClass gets a score of 1 and no relationship gets a score of 0. The discretization of the score into three values (0, 0.5, 1.0) gives a coarse idea of semantic separation between ontological concepts (other finer scoring mechanisms are possible). The scores from two sources are then combined to obtain an overall semantic score for a given pair of concepts. Several schemes such as winner-takes-all, weighted average could be used to combine domain-specific and domain-independent cues for a given attribute. In SEMAPLAN, these schemes are configurable. The default scheme is winner-takes-all. As a

result an efficient index is created from this semantic matching which we call a s*emantic similarity map*.

Prefiltering: Once indexing of related concepts is accomplished, we perform prefiltering using heuristics to obtain a list of candidate matching services for a given request. We implemented a simple backward searching algorithm to select candidate services in the prefiltering stage.

Generating Compositions using Metric Planner: The candidate application interfaces from pre-filtering stage are passed to a *metric planner* along with the request interface description, and the *semantic similarity map*. A planning problem P is a 3-tuple $< I, G, A >$ where I is the complete description of the initial state, G is the partial description of the goal state, and A is the set of executable (primitive) actions. A state T is a collection of literals with the semantics that information corresponding to the predicates in the state holds (is true). An action A_i is applicable in a state T if its precondition is satisfied in T and the resulting state T' is obtained by incorporating the effects of A_i. An action sequence S (called a *plan*) is a solution to P if S can be executed from I and the resulting state of the world contains G. Note that a plan can contain none, one or more than one occurrence of an action A_i from A. A planner finds plans by evaluating actions and searching in the space of possible world states or the space of partial plans.

The semantic distance represents an uncertainty about the matching of two terms and any service (action) composed due to their match will also have uncertainty about its applicability. However, this uncertainty is not probability in the strict sense of a probabilistic event which sometimes succeeds and sometimes fails. A service composed due to an approximate match of its precondition with the terms in the previous state will always carry the uncertainty. Hence, probabilistic planning is not directly applicable and we choose to represent this uncertainty as a cost measure and apply metric planning to this problem. A metric planning problem is a planning problem where actions can incur different costs. A metric planner finds plans that not only satisfies the goal but also does in lesser cost. Note that we can also model probabilistic reasoning in this generalized setting. Table 1 below presents a pseudo-code template of a standard forward state-space planning algorithm, **ForwardSearchPlan.** To support planning with partial semantic matching, we have to make changes at Steps 7 and 10. The heuristic function has to be modified to take the cost of the partial plans into account in addition to how many literals in the goals have been achieved. For Step 10, we have to generalize the notion of action applicability. Conventionally, an action A_i is applicable in a state T if all of its preconditions are true in the state. With semantic distances, a precondition approximately matches the literals in the state. We have a number of choices for calculating the plan cost. In selecting the semantic cost of the action, schemes such as min, max, aggregate of the constituents can be used. In computing the semantic cost of the plan addition, multiplication schemes are possible. These mechanisms are configurable in SEMAPLAN. We have implemented such a metric planner in the Java-based Planner4J framework (Srivastava 2003).

ForwardSearchPlan(I, G, A)

1. If $I \supset G$
2. Return { }
3. End-if
4. N_{init}.sequence = { }; N_{init}.state = I
5. Q = { N_{init} }
6. While Q is not empty
7. N = Remove an element from Q (heuristic choice)
8. Let S = N.sequence; T = N.state
9. For each action A_i in A (all actions have to be attempted)
10. If precondition of A_i is satisfied in state T
11. Create new node N' with:
12. N'.state = Update T with result of effect of A_i and
13. N'.sequence = Append(N.sequence, A_i)
14. End-if
15. If N'.state $\supset$ G
16. Return N' ;; Return a plan
17. End-if
18. Q = Q U N'
19. End-for
20. End-while
21. Return FAIL ;; No plan was found

Table 1. Pseudo-code for the ForwardSearchPlan algorithm.

Solution Ranking: Finally, the alternative compositions generated by the planner are ranked by the *ranking module*. The *ranking module* can use various criteria to rank the solutions. For example, it could rank results based on the overall cost of the plan or rank based on the length of the plan (i.e., the number of services in the plan) or use multidimensional-sorting.

Experimental Results

We ran several experiments on a collection of over 100 Web services. The planner performance is measured through the *recall function (R)*, defined as the ratio between the number of direct plans (no redundant actions) retrieved by the planner and the total number of direct plans in the database. The total number of direct plans in the database was computed empirically. The number of direct plans retrieved by the planner was computed by intersecting the set of planners found by the planner with the set of direct plans defined by the database.

The experiments were executed by varying the following levers in the SEMAPLAN system and observing the planner performance: (a) the *semantic threshold (ST)* allows different levels of semantic ambiguity to be resolved (b) *the number of state spaces explored (#SS)* limits the size of the searching space (c) the *cost function (CF)*, defined as [w*semantic distance+(1-w)*length of the plan] where 0 <= w <= 1, directs SEMAPLAN system to consider the semantic scores alone or the length of the plan alone or a combination of both in directing the search. We ran the following four experiments to measure the performance of SEMAPLAN.

1. Metric Planner alone Vs. SEMAPLAN
2. SEMAPLAN: *f(ST)* where *CF*, and *#SS* are constants.
3. SEMAPLAN: *f(#SS)* where *CF*, and *ST* are constants.
4. SEMAPLAN: *f(#CF)* where *ST*, *#SS* are constants.

Keeping space considerations in mind, in this paper we report the result of the first experiment. Details of other experiments are available in (Akkiraju et al., 2006).

Metric Planner alone Vs. SEMAPLAN: In this experiment, our hypothesis was that a planner with semantic inferencing would produce more relevant compositions than a planner alone. The intuition is that the semantic matcher allows concepts such as *lexemeAttr* and *lemmaProp* to be considered matches because it considers relationships such as word tokenization, synonyms, and other closely related concepts (such as *subClassOf, typeOf, instanceOf, equivalentClass*) defined by the domain ontologies; such relationships are not usually considered by the planner. As Figure 3 shows, SEMAPLAN finds more relevant results than a classic metric planner, thus confirming our hypothesis.

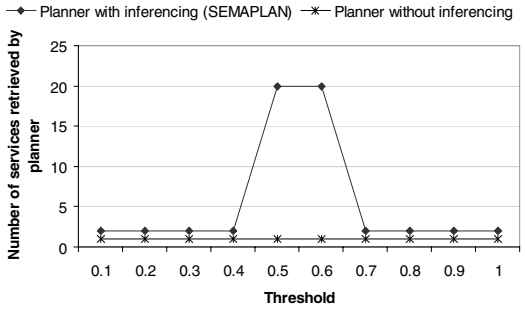

Figure 3: Comparison of Metric Planner Vs. SEMAPLAN

The increased number of solutions is more prevalent with certain semantic thresholds. This behavior is explained in the context of next experiment. For a given number of state spaces explored, as the semantic threshold decreased, more and more loosely related concepts are considered matches by the semantic matcher. This increased the number of services available for the planner to plan from thereby increasing the search space. Therefore, for a given number state spaces to be explored, SEMAPLAN could not come up with some of the good results that it was able to find at higher semantic threshold. This indicates that for a given cost function and for a given number of state spaces to be explored, there is an *optimal threshold*. In most domains, this was found to be 0.6.

Conclusions

In this paper, we presented SEMAPLAN, a novel approach and system to compose Web services in the presence of semantic ambiguity using a combination of semantic matching and AI planning algorithms. The experimental results confirmed our intuition that planning with semantic matching produces more as well as better (when optimal semantic threshold is set) results than using a planning alone. The notion of planning in the presence of semantic ambiguity is conceptually similar to planning under uncertainty. In the future we intend to investigate the application of probabilistic planning techniques to consider semantic differences and compare the results.

References

Akkiraju R, Srivastava, B, et al. 2006. SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition, IBM Research Report.

Akkiraju R, Farrell J, Miller, et al. 2005 Web Services Semantics - WSDL-S. A W3C submission.

Agarwal V. et al 2005. Synthy. A System for End to End Composition of Web Services. Journal of Web Semantics, Vol. 3, Issue 4, 2005

Srivastava B. 2004. A Framework for Building Planners. In the Proc. Knowledge Based Computer Systems.

Christenson E., Curbera F., et al.. "Web services Description Language" (WSDL) 2001. www.w3.org/TR/wsdl

Dong X. et al. 2004. Similarity search for Web services. *In Proc. VLDB, pp.372-283*, Toronto, CA.

Miller G.A 1983. WordNet: A lexical database for the English language, in Comm. ACM 1983.

OWL Technical Committee. "Web Ontology Language (OWL)". 2002. http://www.w3.org/TR/2002/WD-owl-ref-20021112/

Ponnekanti S. Fox A. 2002. SWORD: A Developer Toolkit for Web Service Composition. Proc. 11th WWW.

Syeda-Mahmood T., Shah G., et al. 2005 Searching Service Repositories by Combining Semantic and Ontological Matching. *Third ICWS, Florida.*

Syeda-Mahmood 2004. Minelink: Automatic Composition of Web Services through Schema Matching. Poster WWW 2004.

Traverso P. and Pistore M., 2004. Automated Composition of Semantic Web Services into Executable Processes. 3rd Int. Semantic Web Conf., November.