RC23897 (W0603-017) March 1, 2006
Computer Science

IBM Research Report

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 Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[2]IBM Research Division
India Research Laboratory
Block 1, IIT Campus, Hauz Khaus
New Delhi, 11016
India

[3]IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

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

The use of planning for automated and semi-automated composition of web services has enormous potential to reduce costs and improve quality in inter and intra-enterprise business process integration. 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. In addition, we integrate semantic and ontological matching with an indexing method, which we call attribute hashing, to enable fast lookup of semantically related concepts. 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

Enterprise application integration is among the most critical issues faced by many companies today. The problem is caused by the way systems are developed today in large enterprises, i.e., over different periods of time, for different initial purposes, by different organizations, and with different structures, interfaces and vocabulary. The infrastructure also evolves through acquisitions, mergers and spin-offs. This leads to substantial heterogeneity in syntax, structure and semantics. In this setting, companies are under constant pressure to be flexible, to adapt to the changes in the market conditions while keeping their IT expenses under control, and to implement integration projects without delay. An important aspect of quickly implementing new integration projects involves the ability to find and reuse as much of the existing functionality as possible and create new functionality only where needed. In the context of service-oriented architectures, this translates into the technical challenges of discovery, reuse and composition of services.

In implementing service-oriented architectures, Web services are becoming an important technological component. Web services offer the promise of easier system integration by providing standard protocols for data exchange using XML messages and a standard interface declaration language such as the Web Service Description Language (WSDL 2001). The loosely coupled approach to integration by Web services provides encapsulation of service implementations, making them suitable for use with legacy systems and for promoting reuse by making external interfaces explicitly available via a WSDL description. However, this still does not address the vexing issue of dealing with heterogeneity in service interface definitions. For example, what one service interface in one system may encode as *itemID*, *dueDate*, and *quantity* may be referred to by another service interface in a different system as *UPC* (Universal Part Code), *itemDeliveryTime* and *numItems*. At the heart of data and process integration is the need to resolve these types of similarities and differences among various formats, structures, interfaces and ultimately vocabulary. Developing tools to help resolve these types of syntactic, structural and semantic similarities and differences is key to keeping IT expenses in check. In this paper, we address the problem of identifying the appropriate Web services for implementing a required function from a large collection of available Web services. Specifically, we focus on the problem of Web service composition in the absence of a common domain model and where the functionality of multiple services has to be composed in order to achieve a valid implementation.

Web services matching and composition have 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 explored the application of information retrieval techniques for identifying suitable services in the presence of semantic ambiguity from large repositories. The second direction investigated the application of AI planning algorithms to compose services. In the latter approach, Web services are framed as actions that are applicable to states and the inputs and outputs of services are modeled as preconditions and effects of actions. 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. In addition, we integrate semantic and ontological matching with an indexing method, which we call attribute hashing, to enable fast lookup of semantically related concepts. 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.

The rest of the paper is organized as follows. First, we present a scenario to illustrate the need for Web services composition in certain business domains and discuss how our approach helps in resolving the semantic ambiguities better. Second, we present our solution approach and discuss the details of our system SEMAPLAN. Third, we present our experimental results and discuss the planner performance under various conditions. Fourth, we compare our work with related work in this area. Finally, we present our conclusions and directions for future work.

A Motivating Scenario

Composing existing Web services to deliver new functionality is a requirement in many business domains. In this section, we present a scenario from the knowledge management domain to illustrate the need for (semi) automatic composition of Web services and exemplarily highlight how semantic matching combined with planning could yield better results.

The general goal of text analysis is to transform unstructured text into structured information, and to use this information to support higher-level processes of text search, mining, and discovery (Mack, Mukherjea et.al. 2004). This involves writing annotators or software programs that can interpret text documents, parse them, identify phrases, grammar, classify text and eventually create structure from the unstructured information. Research in this area over the years has led to the development of several annotators. Some are general purpose annotators while the others are specific to various application domains all of which could be made available as Web services. Some sample general purpose ones include annotators such as a *Tokenizer*, which identifies tokens, a *LexicalAnalyzer*, which identifies parts of speech, a *NamedEntityRecognizer*, which identifies references to people and things etc. Sample annotators from biological domain include *BioAnnotator*, which identifies biological terms, *ChemFrag*, which identifies biologically significant chemical structuctures, *DrugDosage*, which recognizes drug applications and dosages etc. Typically, the functionality of multiple annotators needs to be combined to meet a specific request. For example, if a user would like to identify names of authors in a given document, annotators *Tokenizer, LexicalAnalyzer* and *NamedEntityRecognizer* could be composed to meet the request. *Tokenizer* annotator tokenizes a given document. *LexicalAnalyzer* performs lexical analysis on tokens. Finally, *NamedEntityRecognizer* annotator identifies and classifies tokens based on their lexical properties into the names of peoples, places and things. Figure 1 summarizes this composition flow. Such dynamic composition of functionality, that could be represented as Web services, saves tedious development time in complex knowledge management domains such as life sciences since explicating all possible and meaningful combinations of annotators in this case is prohibitive. AI Planning algorithms are well suited to generate these types of compositions. However, as discussed earlier, 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.

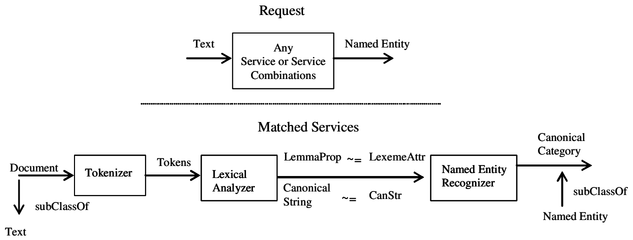

Figure 1. Text Analysis Composition example with semantic matching (~= illustrates semantic match)

Just as with any software development process, annotators are written by multiple authors at different periods of time. These authors could have used different terminology to describe the interfaces of their annotators. In addition, domain specific annotators could have been acquired from external sources (via licensing, acquisition etc). So, it is unlikely that they use a common set of terms to name services (annotators in this scenario) and parameters. This creates semantic ambiguity that, if unresolved, could lead to poor management of available applications. For example, the term *lexemeAttrib* may not match with l*emmaProp* unless the word is split into l*exeme* and *Attrib* and matched separately. Using a domain-dependent ontology one can infer that a lemma in linguistic context is a canonical form of a lexeme and therefore the term *lemma* could be considered a match to the term *lexeme*. Abbreviation expansion rule can be applied to the terms *Attrib* and *Prop* to expand them to *Attribute* and *Property*. Then a consultation with a domain-independent thesaurus such as WordNet 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.

Our Solution Approach

Figure 2 illustrates the components and the control flow in SEMAPLAN system. There are five steps in the system, as explained at a high-level below. Details are given in the subsections following later in this section.

1. *Service Representation*: This step involves preparing Web Services with semantic annotations and readying the domain dependent and independent ontologies.
2. *Term Relationship Indexing*: In step 2, the available Web services in the repository are parsed, processed and an index consisting of related terms/concepts referred to in the service interface descriptions is created 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. The result of indexing is a s*emantic similarity map*. This *semantic similarity map* is capable of returning a semantic score for a given pair of concepts by combining the individual scores from domain-dependent and domain-independent sources. This map is organized for efficient retrieval of related concepts and their scores for a given concept.
3. *Prefiltering*: Once indexing of related concepts is accomplished, in step 3 we perform prefiltering to obtain a list of candidate matching services for a given request. This is done by a *prefiltering module*. The job of the *prefiltering module* is to use smart techniques to obtain a candidate set of interface descriptions from the given set of available interface descriptions from which compositions can be created.
4. *Generating Compositions*: In step 4, these candidate application interfaces are passed to a *metric planner* along with the request interface description, and the *semantic similarity map*. The *metric planner* runs partial order planning algorithms and generates a set of alternative compositions from the given candidate set for the given request interface description. To determine which interfaces can be composed with which others, *metric planner* uses the *semantic similarity map*.
5. *Solution Ranking*: Finally, in step 5, the alternative compositions are ranked by the *ranking module*.

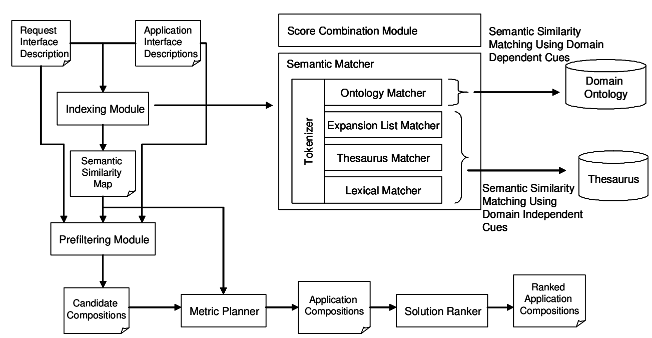

Figure 2 SEMAPLAN system and its components

The main benefit of SEMAPLAN is the ability to compose plans in the presence of inexact terms. This is expected to improve the recall[1] of results. We verify this hypothesis by running experiments, as presented in ‘Experimental Results’ section. We now describe each step in detail.

Service Representation

The functionality of services is represented using the Web Services Description Language (WSDL). Domain independent dictionaries can be used to match the terms used in the WSDL document. However, in order to use domain-specific ontological information, references to the ontology must be present in the service description. The standard WSDL specification does not have a mechanism to denote such ontological information and hence must be augmented before such information can be used to determine matching services. The subject of semantic annotation is an active area of research in the semantic web community with languages such as OWL-S (OWL-S 2001), WSMO (WSMO 2003), WSDL-S (Shivashanmugham et al 2003, Akkiraju et al 2005), etc. In this work, we have adopted the WSDL-S specification due to its simplicity.

```
xmlns:wssem=”http//www.ibm.com/schemas/2004/wssem”
xmlns:TextAnalysisOntology="http://www.ibm.com/ontologies/TextAnalysisOntology.owl" >
<message name="chemicalNameIdentifierRequest">
        <part name="named_entity_in" type="xsd:string"
wssem:modelReference="TextAnalysisOntology#NamedEntity"/>
  </message>
  <message name="chemicalNameIdentifierResponse">
       <part name="chem_out" type="xsd:string"
wssem:modelReference="TextAnalysisOntology#ChemicalName"/>
 </message>
```

We create domain-specific ontologies using OWL (OWL 2002). Using the WSDL-S specification, we annotate elements in the WSDL file using the attribute *wssem:modelReferenc*es. Its value is an OWL ontology concept specified by the name of the ontology and the relevant ontological term. Such an annotated WSDL file corresponding to the text analysis domain is shown above. After parsing the WSDL documents, we create a generalized schema object internally to capture the service definitions, portTypes and other information.

Term Relationship Indexing

In this section, we discuss (a) how semantic matching of service interface descriptions can be accomplished by using both domain-dependent and domain-independent cues, (b) how matches due to the two cues (domain-independent and domain-specific) are combined by the *score combination module* to determine an overall semantic similarity score, and (c) how efficient indexing is performed. In an earlier work, we show that by combining multiple cues, better relevancy of results can be obtained for service matches from a large repository, than that could be obtained using any one cue alone (Syeda-Mahmood et al 2005).

(a.1) Finding related terms using domain independent ontologies. Finding semantic relationship between attributes is difficult because (1) Attributes could be multi-word terms (e.g. *CustomerIdentification, PhoneCountry*, etc.) which require tokenization. Any tokenization must capture naming conventions used by programmers to form attribute names; (2) Finding meaningful matches might need to account for senses of the word as well as their part-of-speech through a thesaurus; (3) Multiple matches of attributes must be taken into account; and (4) Finally, the structure/type information must be exploited so that operations match to operations, messages to messages, etc.

We capture name semantics using a technique similar to the one in (Dong X. 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. For example, customer and client would be considered a match because they are synonyms. *CustID* is matched with *ClientNum* because words such as *custID* get expanded to *CustomerIdentifier* and *ClientNum* gets expanded to *ClientNumber* and are matched separately (*Cust* with *Client* and *ID* with *Num*). Stop words such as *and*, *the*, etc. are filtered out. We used the WordNet thesaurus (Miller 1983) to find matching synonyms to words. Each synonym is assigned a similarity score based on the sense index, and the order of the synonym in the matches returned. The result of this semantic matching process is that a given pair of concepts is given a semantic score based on these domain-independent cues. That score is computed as follows:

Consider a pair of candidate matching attributes (A, B) from the query and repository services respectively. These matching attributes could be a pair of inputs to be matched from a service request and an available service from a repository. Let A, B have m and n valid tokens respectively, and let S_{yi} and S_{yj} be their expanded synonym lists based on domain-independent ontological processing. We consider each token i in source attribute A to match a token j in destination attribute B where $i\ \varepsilon\ S_{yi}$ and $j\ \varepsilon\ S_{yi}$. Let us say that h tokens have a match. Then, the semantic similarity

[1] We define **Recall** as the ratio of the number of relevant services (compositions) retrieved to the total number of relevant services/compositions in the repository. We express it as a percentage. More details can be found in the “Experimental Results” section.

between attributes A and B is then given by: $M_{sem,}$= *min*{h/n, h/m}. This use of the ratio of matched to total terms allows us to deal with services that have vastly different numbers of parameters.

(a.2) Finding related terms using domain-specific ontologies. We use a semantic network-based ontology management system known as SNoBASE (Lee et al 2003) that offers DQL-based (DQL 2003) Java API for querying ontologies represented in OWL. The OWL-specified ontologies loaded into SNOBASE are parsed to populate its internal data store with facts and instances. The engine models four different types of relationships: (1,2) *subClassOf(A,B), subClassOf(B,A)* – which is essentially superClassOf, (3) *type (A,B)* – which is instanceOf, and (4) *equivalenceClass(A,B)* are modeled where A and B are two given concepts. 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. This score between a given two concepts is represented as M_{ont}. More refined scoring schemes are possible, but the current choice works well in practice without causing a deep semantic bias. Given a domain-specific ontology and a query term, the related terms in an ontology are found using rule-based inference. In the SNoBASE system we used, IBM's ABLE (Bigus et al 2001) engine for inference. The ABLE library includes rule-based inference using Boolean and fuzzy logic, forward chaining, backward chaining etc. The result of this domain-dependent ontology based inferencing is that a given pair of concepts is given a semantic score based on these domain-dependent cues.

(b) Score Combination. Once semantic scores from domain-independent and domain-dependent cues are obtained, these individual scores 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, where the best possible score (ontology-wise or semantic-matching-wise) is taken as the match score for a given pair of attributes. For each potential matching attribute pairs, let M_{sem} be the matching score using semantic matching. Let M_{ont} be the matching score using ontological matching. Then the combined score is: $M = max\ \{M_{sem}, M_{ont}\}$

(c) Indexing. With the approach we have described so far, all services attributes would have to be searched for each query service to find potential matches and to assemble the overall match results. We now present attribute hashing, an efficient indexing scheme that achieves the desired savings in search time.

To understand the role of indexing, let us consider a service repository of 500 services. If each service has about 50 attributes (quite common for enterprise-level services), and 2 to 3 tokens per word attribute, and about 30 synonyms per token, the semantic matching alone would make the search for a query of 50 attributes easily around 50 million operations per query! Indexing of the repository schemas is, therefore, crucial to reducing the complexity of search. Specifically, if the candidate attributes of the repository schemas can be directly identified for each query attribute without linearly searching through all attributes, then significant savings can be achieved.

The key idea in attribute hashing can be explained as follows. Let 'a' be an entity derived from a repository service description. Let F(a) be the set of related entities of 'a' in the entire service repository (also called feature set here). In the case of domain-independent semantics 'a' refers to a token and F(a) is the set of synonyms of 'a'. In the case of ontological matching, 'a' refers to an ontological annotation term, and F(a) are the ontologically related concepts to a (e.g. terms related by subclass, equivalenceClass, is-a, etc. relationships) . Now, given a query entity q derived from a query service Q, q is related to a iff q ε F(a). Thus instead of indexing the set F(a) using the attribute a as a key as may be done in normal indexing, we use the terms in the set F(a) as keys to index a hash table and record 'a' as an entry in the hash table repeatedly for each such key. The advantage of this operation is that since q ε F(a), q is indeed one of the keys of the hash function. If this operation is repeated for all entities in the service repository, then each hash table entry indexed by a key records all entities whose related term set includes the key. *Thus indexing the hash table using the query entity q directly identifies all related entities from the service repository without further search*! This is the key idea of attribute hashing. Of course, this is done at the cost of redundant storage (the entity 'a' is stored repeatedly as an entry under each relevant key). However, with the growth of computer memory, storage is a relatively inexpensive tradeoff.

Prefiltering

The prefiltering module selects a set of candidate pool of services from which compositions can be accomplished. If the number of services in the repository is relatively small (of the order of dozens), then prefiltering may not be necessary. However, in data warehousing type of scenarios or in asset reuse scenarios, there could be typically hundreds of interfaces from which suitable applications have to be constructed; thus, obtaining a manageable set of candidate services via filtering is crucial to returning results in reasonable amount of time. Of course, as with any filtering process, there is the possibility of filtering out some good candidates and bringing in bad candidates. However, prefiltering can reduce the search space and allow planning algorithms to focus on a viable set. We employ a simple backward searching algorithm to select candidate services in the prefiltering stage. The algorithm works by, first, collecting all services that match at least one of the outputs of the request – denoted as $S_1 1$, $S_1 2$, $S_1 3$.. $S_1 n$ where n is the number of services obtained in step 1 and S_1 denotes services collected in step 1. Let $S_1 i$ represent a service collected from step 1 where $1 \leq i \leq n$... Then, for each service $S_1 i$, we collect all those services whose outputs match at least one of the inputs of S_1i. This results in a set of services added to the collection in step 2 – denoted as $S_2 1$, $S_2 2$, $S_2 3$.. $S_2 m$ where m is the number of services obtained in step 2. This process of collecting services is repeated until either a predefined set of iterations are completed or if at any stage no more matches could be found. The criteria for filtering could have significant influence on the overall quality of results obtained. One can experiment with these criteria to fine-tune the prefiltering module to return an optimal set of candidate pool of services. *The prefiltering module* uses the *semantic similarity map* obtained from the indexing stage to determine whether a given interface description concept is a match to another concept in a different interface description.

Generating Compositions using Metric Planner

The set of candidate services obtained from the prefiltering step are then presented to the planner. 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 (Weld 1999). 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 $\boldsymbol{A_i}$ 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 $\boldsymbol{A_i}$. 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 $\boldsymbol{A_i}$ 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 (Kushmerick et al. 1995) 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.

We now illustrate the changes needed in a standard metric planner to support planning with approximate distances. Our approach uses planning in the state of world states (state space planning) but it is applicable to searching in space of plans as well. Table 1 below presents a pseudo-code template of a standard forward state-space planning algorithm, **ForwardSearchPlan.** The planner creates a search node corresponding to the initial state and inserts it into a queue. It selects a node at Step 7 from the queue guided by a heuristic function. It then tries to apply actions at Step 10 whose preconditions are true in the corresponding current state. The heuristic function is an important measure to focus the search towards completing the remainder part of the plan to the goals.

```
ForwardSearchPlan(I, G, A)
1. If I ⊃ 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 ⊃ 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.

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 $\boldsymbol{A_i}$ is applicable in a state ***T*** if all of its preconditions are true in the state. With semantic distances, a precondition is approximately matching the literals in the state. We have a number of choices for calculating the plan cost:

a) In matching of an action's precondition with the literals in the state, which semantic distance should be selected? We can use the first one, the least distance, or any other possibility.
b) In selecting the semantic cost of the action, how is the contribution of the preconditions aggregated? We can take the minimum of the distances, maximum, or any other aggregate measure.
c) In computing the semantic cost of the plan, how is the contribution of each action in the plan computed? We can simply add the costs of the actions, take their products, or use any other function.

We have implemented such a metric planner in the Java-based Planner4J framework (Biplav et. al 2003). Planner4J provides for planners to be built exposing a common programmatic interface while they are built with well-tested common infrastructure of components and patterns so that much of the existing components can be reused. Planner4J has been used to build a variety of planners in Java and it has eased their upgrade and maintenance while facilitating support for multiple applications. The planner can be run to get all plans within a search limit, different plans by changing the threshold for accepting semantic distances and by experimenting with various choices of cost computations for the actions and plans as outlined above.

Solution Ranking

The *ranking module* can use various criteria to rank the solutions. For example, one way to rank the criteria would be to sort the compositions in ascending order of the overall cost of the plan. Another way is to rank the compositions based on the length of the plan (i.e., the number of services in the plan). Multidimensional sorting approach could be used to sort based on both cost and the length of the plan. Multiplying the normalized costs is another approach. This approach brings in notions of probabilistic planning and enables to take both cost and length into account at once. In SEMAPLAN, these approaches are configurable.

Experimental Results

The goal of our evaluation section is to demonstrate the value of combining domain-independent and domain-dependent semantic scores with a metric planner when composing Web services. For this, we ran several experiments on a collection of over 100 Web services in three domains: (1) Text analysis - 20 WSDLs that provide text analysis services, (2) Alphabet – 7 WSDLs manually built to test the correctness of the planner when the relationships between services are very complex, and (3) Telco – 75 WSDLs defined from a real life telecommunication scenario. For clarity, we report only the results for the first domain, as the other two domains presented similar behavior. The planner performance is measured through the ***recall function (R)***, defined as the ratio between the number of direct plans retrieved by the planner and the total number of direct plans in the database. We define a direct plan as a correct plan (i.e., it reaches the goal state, given the initial state) with a minimum number of actions (i.e., we discard plans that contain loops or redundant actions). Our definition of the recall function was due to an interesting observation we made when contrasting the results returned by a search engine in the information retrieval domain and the ones returned by a planner in the Web Services domain. In Web search, recall is defined by relevancy. A search query that is looking for 'Soprano' could find results consisting of Sopranos (the HBO show) as well as information on sopranos from Wikipedia etc. Depending on what the user meant, the user would find one of the two categories of search results to be more relevant than the others. However, when composing Web services using a planner, the notion of relevancy needs to be interpreted slightly differently. Since planners are goal directed, and the semantic matches are often driven by closely related terms in the domain-independent and domain-dependent ontologies, all the results obtained were found to be relevant. Therefore, we redefined relevancy as the direct matches SEMAPLAN is able to find with minimal length (fewer number of services composed to meet the request) without redundancies in our experiments. For example, in the text analysis example, one of the direct plans is the sequence of: *Tokenizer, Lexical Analyzer* and *NamedEntityRecognizer* services. We have noticed that depending on the number of states we allow the SEMAPLAN system to run, it finds compositions that include sequences such as: *Tokenizer, LexicalAnalyzer, Tokenizer*, and *NamedEntityRecognizer*. In this plan, the second *Tokenizer* is redundant. The total number of direct plans in the database was computed by manually performing an exhaustive search and counting all plans. 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.

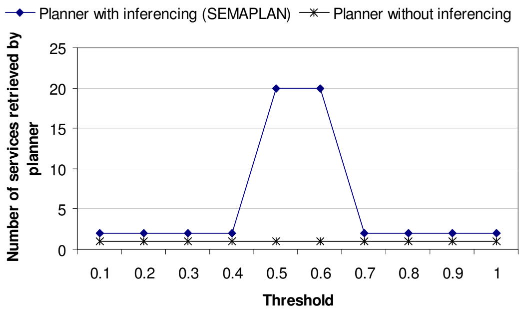

Figure 3: Comparison of Metric Planner Vs. SEMAPLAN

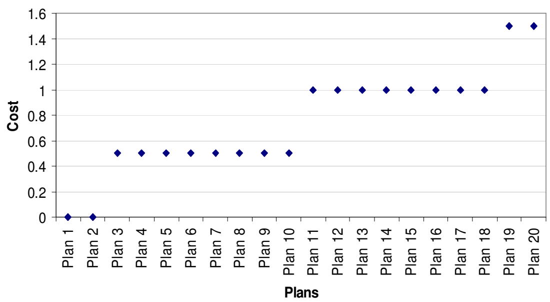

Figure 4. Costs of all plans retrieved by planner (threshold=0.6)

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 *lexemeAttrib* 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. The costs of all plans retrieved by SEMAPLAN are shown in Figure 4. The increased number of solutions is more prevalent with certain semantic thresholds. This behavior is explained in the context of next experiment.

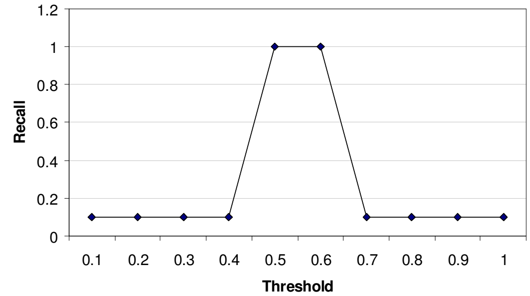

Figure 5: SEMAPLAN performance when varying threshold

SEMAPLAN: *f(ST)* where CF, and #SS are constants: In this experiment, we varied the semantic threshold for a given number of state spaces to be explored (1000) and a given cost function for each domain (w=0.5). As the semantic threshold increases, only those concepts that are above the threshold would be considered matches; therefore, we expected that the number of results produced by the planner would decrease and vice versa. While this is confirmed by our results in figure 5, we noticed an interesting phenomenon. 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. Therefore both in figure 3 and figure 4, we can notice a drop in the number of plans retrieved by the planner at higher threshold. Our observation from this experiment is 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. This led us to run the third experiment to see if SEMAPLAN can discover the missing good plans if the number of state spaces explored are increased.

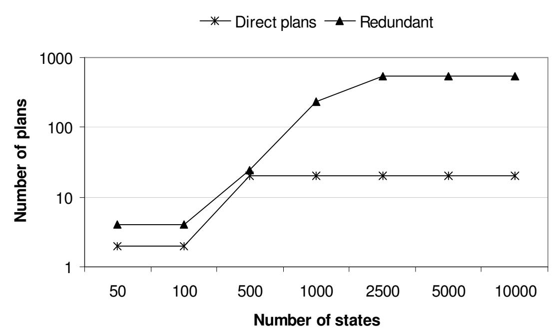

Figure 6: SEMAPLAN performance when increasing number of state spaces searched

SEMAPLAN: *f(#SS)* where CF, and ST are constants: Based on the insights from the second experiment, we varied the number of state spaces by keeping the weight *w* in cost function and semantic threshold at the optimal levels (w= 0.5, and ST=0.6). The results of this experiment, as shown in Figure 6, revealed that as the number of state spaces explored increases, SEMAPLAN finds more plans in general and more direct relevant plans than it could at the same ST and w. This is consistent with our expectations.

weight	direct	redundant	recall
0	20	234	1
0.5	20	374	1
1	20	423	1

Table 2: SEMAPLAN performance when changing the cost function

SEMAPLAN: *f(#CF)* where ST, #SS are constants: Finally, we varied the weight in the cost function to see how the quality of the plans generated get impacted by this. As weight approaches 1, the cost function gives less preference to length, therefore we expect to see more number of longer plans (sometimes with redundancies) than those expected at lower weights and vice versa. The results illustrated in Table 2 confirm this intuition.

Related Work

The literature on Web services matching and composition has focused on two main directions. One body of work explored the application of AI planning algorithms to achieve composition while the other investigated the application of information retrieval techniques for searching and composing of suitable services in the presence of semantic ambiguity from large repositories. In this section we contrast our approach with those taken by these two bodies of work.

First, we consider work that is done on composing Web services using planning based on some notion of annotations. A general survey of planning based approaches for web services composition can be found in (Peer 2005). SWORD (Ponnekanti, Fox 2002) was one of the initial attempts to use planning to compose web services. It does not model service capabilities in an ontology but uses rule chaining to composes web services. In (McIlraith 2001), a method is presented to compose Web services by applying logical inferencing techniques on pre-defined plan templates. The service capabilities are annotated in DAML-S/RDF and then manually translated into Prolog. Now, given a goal description, the logic programming language of Golog (which is implemented over Prolog) is used to instantiate the appropriate plan for composing the Web services. In (Traverso, Pistore 2004), executable BPELs are automatically composed from goal specification by casting the planning problem as a model checking problem on the message specification of partners. The approach is promising but presently restricted to logical goals and small number of partner services. Sirin et al. (Sirin et al 2003) use contextual information to find matching services at each step of service composition. They further filter the set of matching services by using ontological reasoning on the semantic description of the services as well as by using user input. Web Services Modeling Ontology is a recent effort for modeling semantic web services in a markup language (WSML) and also defining a web service execution environment (WSMX) for it. Our logical composition approach is not specific to any particular modeling language and can adapt to newer languages. Synthy (Agarwal et al 2005) takes an end-to-end view of composition of Web Services and combines semantic matching with domain ontologies with planning. While these bodies of work use the notion of domain annotations and semantic reasoning with planning, none of them use domain-independent cues such as thesaurus. Moreover, they do not consider text analysis techniques such as tokenization, abbreviation expansions, and stop word filtering etc. in drawing the semantic relationships among the terms referenced in Web services.

The second body of work looked at composition of Web services using domain independent cues alone. Syeda-Mahmood (Syeda-Mahmood 2004) models Web service composition as bipartite graph and solves a maximum matching problem while resolving the semantic ambiguities using domain-independent ontologies and text analysis approaches. This work takes its roots in schema matching. However, this work does not use domain-dependent ontologies which are crucial to resolving the differences between domain-specific contextual terms.

SEMAPLAN, to the best of our knowledge, is the first attempt at combining semantic matching consisting of domain-dependent and domain-independent ontologies with AI planning techniques to achieve Web services composition.

Conclusions and Future Work

In this paper, we have presented 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. 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. The experimental results confirmed our intuitions: (1) the number of direct plans improved by combining semantic matching with planning algorithms; thus, SEMAPLAN achieved better recall than the metric planner alone, and (2) there is a tradeoff between the semantic threshold, the limit on the state search space, the cost function, and the quality of results obtained. We noticed that there is an optimal threshold, which when set, helps focus the planner and at the same time provides enough semantic variety to improve recall.

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, Farrell J, Miller, J. Nagarajan M, Sheth A, Verma K. 2005 Web Services Semantics - WSDL-S. A W3C submission.

Ankolekar A., Burstein M., Hobbs J. J., *et al*. 2001. DAML-S/OWL-S: Semantic Markup for Web Services. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*

Agarwal V. Chafle G, Dasgupta K. et al 2004. Synthy. A System for End to End Composition of Web Services. http://www.research.ibm.com/people/b/biplav/areas.html

Bigus J., and Schlosnagle D. 2001. Agent Building and Learning Environment Project: ABLE. http://www.research.ibm.com/able/

Biplav S. 2004. A Framework for Building Planners. In the Proc. Knowledge Based Computer Systems

Christenson E., Curbera F., Meredith G., and Weerawarana S.. "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.

DQL Technical Committee 2003. DAML Query Language (DQL) http://www.daml.org/dql

Lee J., Goodwin R. T., Akkiraju R., Doshi P., Ye Y. 2003. SNoBASE: A Semantic Network-based Ontology Ontology Management. http://alphaWorks.ibm.com/tech/snobase.

Mack. R, Mukherjea S., Soffer A., Uramoto N. et. al. 2004. Text Analytics for Life sciences using the Unstructured Information Management Architecture (UIMA). *IBM Systems Journal. Volume 43. Number 3*.

Mandel, D., McIlraith S., 2003 Adapting PBEL4WS for the semantic web: The bottom up approach to web service interoperation *Second International Semantic Web Conference (ISWC2003),* Sanibel Island, Florida.

McIlraith S. and Son T.C and Zeng H. 2001. Semantic Web Services. IEEE Intelligent Systems, Special Issue on the Semantic Web. March/April. Number 2, Pages 46-53 Volume 16.

Melnik S. et al, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching," in Proc. ICDE, 2002.

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

Nicholas Kushmerick and Steve Hanks and Daniel S. Weld 1995. An Algorithm for Probabilistic Planning, Artificial Intelligence, volume 76 number 1-2, pages 239-286.

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

Peer J. 2005. Web Service Composition as a Planning Problem – a survey. http://elektra.mcm.unisg.ch/pbwsc/docs/pfwsc.pdf

Ponnekanti S. Fox A. 2002. SWORD: A Developer Toolkit for Web Service Composition. Proc. Of the 11th International World Wide Web Conference.

Sirin E., Hendler J., and Parsia B. 2003. Semi-automatic composition of web services using semantic descriptions. Web Services: *Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003*, April 2003.

Shivashanmugam K., Verma K., Sheth A., Miller J.. Adding Semantics to Web Services Standards. In the proceedings of *The 2003 International Conference on Web Services (ICWS'03),* Las Vegas, NV, June 23 - 26, 2003,, pp. 395-401.

Syeda-Mahmood T., Shah G., Akkiraju R., Ivan A., and Goodwin R.. 2005 Searching Service Repositories by Combining Semantic and Ontological Matching. *Third International Conference on Web Services (ICWS), Florida.*

Syeda-Mahmood 2004. Minelink: Automatic Composition of Web Services through Schema Matching. Poster paper World Wide Web Conference. 2004.

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

Weld D. 1999. Recent Advances in AI Planning. AI Magazine, 20:2.

WSMO Technical Committee Web Services Modeling Ontology 2003, http://www.wsmo.org