Information Gathering During Planning for Web Service Composition

Ugur Kuter[†] and **Evren Sirin**[†] and **Dana Nau**[†] and **Bijan Parsia**[‡] and **James Hendler**[†]
[†]Department of Computer Science,
University of Maryland, College Park, MD 20742, USA
[‡]MIND Lab, University of Maryland,
8400 Baltimore Ave., College Park, MD 20742, USA

Abstract

Hierarchical Task Network based planning techniques have been applied to the problem of composing Web services, especially when described using the OWL-S service ontologies. However, many Web services either are exclusively information providing or crucially depend on information providing services. Thus, many interesting service compositions involved collecting information either during execution, or in the composition process itself. In this paper, we focus on the latter issue. In particular, we present the WSC-SHOP2, a HTN planning algorithm designed for planning domains in which the information about the initial state of the world may not be complete, but is discoverable through plan-time actions. We have shown that WSC-SHOP2 is sound and complete, and we have derived several mathematical relationships among the amount of available information, the likelihood of WSC-SHOP2 finding a plan, and the quality of the plan found. Our preliminary experimental tests of WSC-SHOP2 confirm the theoretical results.

Introduction

Web services are Web accessible, loosely coupled chunks of functionality with a interface described in a machine readable format, whether by the minimal descriptions supported by the Web Service Description Language (http://www.w3.org/TR/wsdl20/), or the more expressive descriptions supported by OWL-S (OWL Services Coalition 2003). Web services are designed to be *composed*, that is, combined in workflows of varying complexity to provide functionality that none of the component services could provide alone. AI planning techniques can be used to automate Web service composition by representing services as actions, and treating service composition as a planning problem. On this model, a service composition is a ground sequence of service invocations that accomplishes a goal or task.

Using AI planning techniques to automate the composition of Web services introduces some challenges to classical planning systems. Traditional generative planning systems assume that complete information about the planner's state of the world is available. This assumption provides a clear semantics for the planner's inference mechanism. However, many Web services either provide information about the state of the world, typically by exposing databases, or require prior use of information providing services. For example, an appointment making service might require the user to determine an available appointment time first. In many cases, it isn't feasible or practical to execute all the information gathering services up front to form a complete initial state of the world. In such cases, it makes sense to do information gathering during planning.

In this paper, we present a domain-independent, hierarchical, interactive and generative planning algorithm called WSC-SHOP2, which is based on the SHOP2 planning system (Nau *et al.* 2003). WSC-SHOP2 is a sound and complete HTN-planning algorithm for solving planning problems with incomplete information about the initial world state. Basically, WSC-SHOP2 issues queries to learn the truth values of certain atoms when there is not enough information in the knowledge base to determine their values. The planner postpones the decision for the truth-value of that atom until a response comes in while it continues examining alternative branches of the search space. By gathering extra information at plan time, WSC-SHOP2 is able to explore many more branches in the search space than the initial state ordinarily permits. Since external queries often dominate planning time, and, being distributed, are strongly parallelizable, WSC-SHOP2's non-blocking strategy is sometimes able to dramatically improve the time to find a plan.

In this paper, we describe WSC-SHOP2 and its syntax and semantics, and provide the sufficient conditions to ensure its soundness and completeness. We derive a recurrence relation for the probability of WSC-SHOP2 finding a plan, and prove theoretical results that give mathematical relationships among the amount of information available to WSC-SHOP2 and the probability of finding a plan. These relationships are confirmed by our experimental evaluations. We describe how the generic WSC-SHOP2 algorithm can be used to solve Web service composition problem and test the effiency of the algorithm on real Web service composition problems.

Motivation

HTN planning algorithms have proven promising for Web service composition. Many service oriented objectives can be naturally described with a hierarchical structure. HTN style domains fit in well with the loosely coupled nature of Web services: different decompositions of a task are independent so the designer of a method does not have to have close knowledge of how the further decompositions will go. Hierarchical modeling is the core of the OWL-S (OWL Services Coalition 2003) process model to the point where the OWL-S process model constructs can be directly mapped to HTN methods and operators(Wu *et al.* 2003). We have kept the basic SHOP2 language mapping intact and focused on extending the way the SHOP2 algorithm deals with plan-time information gathering.[1] We call the extended algorithm WSC-SHOP2.

We have identified three key features of service oriented planning:

- The planner's initial information about the world is incomplete. When the size and nature of Web is considered, we cannot assume the planner will have gathered all the information needed to find a plan. As the set of operators and methods grows very large (i.e., as we start using large repositories of heterogenous services) it is likely that trying to complete the initial state will be wasteful at best and practically impossible in the common case.
- The planning system should gather information during the planning process. While not all the information relevant to a problem may have already been gathered, it will often be the case that that information is accessible to the system. The relevance of possible information can be determined by the possible plans the planner is considering, so it makes sense to gather that information while planning.
- Web services may not return needed information quickly, or at all. Executing Web services to get the information will typically take longer time than the planner would spend to generate plans. In some cases, it will not be known a priori which Web service gives the necessary information and it will be required to search a Web service repository to find such capable services. It may not be possible at all to find those services. Furthermore, in some cases the found service cannot be executed because the service requires some password that the user cannot provide or the service is inaccesible due to some network failure. The system should not cease planning while waiting for answers to its queries, but keep planning to look for other plans that do not depend on answering those specific queries.

[1]In this paper, we focus on information gathering as plan-time execution of Web services. Nothing in this work, however, is specific to information providing Web services and could be immediately adapted to any oracular query answering mechanism, e.g., a user could interactively supply answers to the system.

WSC-SHOP2 can start with an incomplete initial world state, will gather relevant information during planning, and will continue to explore alternative possible plans while waiting for information to come in.

Definitions and Notation

We used the same definitions for logical atoms, states, task symbols, tasks, task networks, operators, methods, and plans as in the SHOP2 planning system (Nau *et al.* 2003). We extended the SHOP2 planning framework in order to be able to reason about the incomplete information about the states of the world as follows.

A *query* is an expression of the form $q = (h\ a)$, where h is the unique label of the query q and a is a logical atom. Note that we do not require a to be ground. The intent of a query is to gather information during planning about the relation in the world that is represented by the atom a. An *answer* for a query q is an expression of the form $x = (h\ R)$ where h is the same unique label of the query q and R is either a (possibly empty) set of ground atoms such that, for each $a' \in R$, there exists a substitution σ such that $\sigma(a) = \sigma(a')$.

An *askable list* is a list of atoms that the planner is eligible for querying during the planning process. In many realistic application domains, the planner can only obtain certain kinds of information, regardless of whether that information is needed for planning. Intuitively, an *askable list* specifies the kinds of information that is guaranteed to be available to the planner during planning, although this information may not be given the planner at the start of the planning process. For example, in planning for service composition, some of the services that provide the necesary information for planning may not be available, or the planner may not have access to those services for unknown reasons. When a query $q = (h\ a)$ with a possibly partially ground atom a is asked, we locate a Web service who can give the answer to this query. A service whose output or postcondition specification matches with a is said to be a possible match.[2] For example, a query q=(find-airports airport(US,?x)) can be answered a service that has this specification s=(:input ?c - country, :output ?x - airport, :postcondition airport(?c, ?x).

Given a planning domain, a *complete-information planning problem* in that domain is a tuple $P = (S, T, O, M)$, where S is the initial state, T is a task network, and O and M are the set of operators and methods defined for that planning domain, respectively. An *incomplete-information planning problem* in that domain is a tuple $I = (K, A, T, O, M)$, where K is a set of atoms that are initially known, A is the askable-list, and T, O, and M the task network, the set of operators, and the set of methods, respectively, as given for the complete-information problem P.

[2]In this paper, we assumed the existence of matched services. The problem of discovery and location of a matched web service is beyond the scope of this paper.

Given a planning domain, a complete-information problem $P = (S, T, O, M)$ and an incomplete-information problem $I = (K, A, T, O, M)$, are said to be *consistent* if and only if S is consistent with $K \cup W$, where W is the set of all possible ground instances of atoms in the askable list A. Note that $K \cup W$ denotes the total amount of information that a planner can possibly obtain while solving the problem I.

The WSC-SHOP2 algorithm

The WSC-SHOP2 algorithm is shown in Figure 1. The input (K, A, T, O, M) is an incomplete-information planning problem as defined above. Λ is the *asked list*, which is the set of all queries that have been issued by WSC-SHOP2. Θ is the *answered list*, which is the set of all queries for which answers have been received. The $OPEN$ list is the set of 4-tuples of the form (K, T, μ, π), where K is a (possibly incomplete) state, T is a task list, μ is a set of either ground instances of operators or ground instances of methods, and π is a plan. The set μ is used to ensure the correctness of algorithm when it defers a decision on applying a method or an operator due to incomplete information. At the beginning of the planning process, each of these lists is the empty set, except the $OPEN$ list, which is initialized to $\{(K_0, T_0, \mu, \pi)\}$. Planning is performed as an iterative process that can be described as follows.

At any iteration of the planning process, we first check the $OPEN$ list. If it is empty, then there is no plan, so we report $failure$. Otherwise, we check if any answers have arrived for the queries that were already issued before. Suppose there is an answer for a query q, in which R is the set of ground instances of the atom in q that are true in the world. Then, we add the atoms in R into every state in the $OPEN$ list since our queries provide information about the initial state of the world; any information gathered by these queries must be included in the states that are contained by the leaf nodes (i.e., the tuples in $OPEN$) of each of the branches of the search tree as well as in the initial state K_0. Then, we insert the query that was answered in to Θ.

After processing the already-issued queries, we select a tuple (K, T, μ, π) from the $OPEN$ list, and remove it.[3] We then check if the current task network T is empty or not. If so, we have π as our plan since all of the goal tasks have been accomplished successfully. Otherwise, we nondeterministically choose a task t in T that has no predecessors in T. If t is primitive then we decompose it using the operators in O. Otherwise, the methods in M must be used.

Due to the space limitations, we only describe the case in which t is primitive. The case in which t is not primitive is very similar. If t is primitive, then we first find the set of ground instances of the operators in O such that there exists a substitution σ such that $\sigma(t) = head(o)$. For each ground operator instance o in this set, if each precondition p of o is satisfied in K, so o is *applicable* in K. Then, we generate the next state K' and the next task network T' by applying o in K and by removing t from T, respectively.

[3]Note that different mechanisms can be exploited for selecting a tuple from the $OPEN$ list. See the following section for examples and a detailed discussion.

```
procedure WSC-SHOP2($K_0, A, T_0, O, M$)
  $\Lambda \leftarrow \emptyset$; $\Theta \leftarrow \emptyset$; $\pi \leftarrow \emptyset$; $\mu \leftarrow \emptyset$
  $W \leftarrow \{\sigma(a) \mid a \in A$, and $\sigma$ is a variable substitution for $a\}$
  $OPEN \leftarrow \{(K_0, T_0, \mu, \pi)\}$
  loop
    if $OPEN = \emptyset$ then return($failure$)
    for each query $q = (h\ a)$ such that $q \notin \Theta$
      if there is an answer $x = (hR)$ for $q$ then
        if $R \neq \emptyset$ then
          insert the atoms in $R$ into every state in $OPEN$
          $K_0 \leftarrow K_0 \cup R$
        insert $q$ into $\Theta$
    select a tuple $(K, T, \mu, \pi)$ from $OPEN$ and remove it
    if $T = \emptyset$ then return($\pi$)
    $W \leftarrow \{t \mid t \in T$ and $t$ has no predecessors$\}$
    nondeterministically choose a $t \in W$
    if $t$ is a primitive task then
      $applicable \leftarrow$ FindApplicableOp($K, t, \mu, O$)
      for every $(o, \sigma) \in applicable$
        $K' \leftarrow$ ApplyOperator($K, o$)
        $T' \leftarrow \sigma(T \setminus \{t\})$
        $\pi' \leftarrow \pi \cup \{o\}$
        $OPEN \leftarrow OPEN \cup \{(K', T', \emptyset, \pi')\}$
      $deferrable \leftarrow$ FindDeferrableOp($K, t, \mu, O, \Lambda, \Theta$)
      $Q \leftarrow$ GenerateQueries($deferrable$)
    else
      $applicable \leftarrow$ FindApplicableMeth($K, t, \mu, O$)
      for every $(m, \sigma) \in applicable$
        $T' \leftarrow$ ApplyMethod($K, T, t, m, \sigma$)
        $OPEN \leftarrow OPEN \cup \{(K, T', \emptyset, \pi)\}$
      $deferrable \leftarrow$ FindDeferrableMeth($K, t, \mu, M, \Lambda, \Theta$)
      $Q \leftarrow$ GenerateQueries($deferrable$)
    issue all of the queries in $Q$
    $\Lambda \leftarrow \Lambda \cup Q$
    $OPEN \leftarrow OPEN \cup \{(K, T, deferrable, \pi)\}$
```

Figure 1: The WSC-SHOP2 algorithm.

If there exists a precondition p in o such that p cannot be satisfied in K, then we cannot always immediately declare that o is not applicable in K. We need to perform the following checks in an if-then-else manner:

- We have $p \in K_0 \setminus K$, where K_0 is the initial state. Then, this means that p has been deleted by an operator previously during planning. Therefore, p must be false in the world, and o cannot be applied in K.
- We have already issued a query for p before, but did not get an answer for it. In other words, we have $q \in \Lambda$ and $q \notin \Theta$, where q is a query of the form $(h\ p)$, Λ is the asked list and Θ is the answered list as described above. Then, we must *defer* our decision on the applicability of o in K since this decision depends on the answer we will supposedly get.
- We have already issued a query q for p before, and we have already got an answer — i.e., we have $q \in \Lambda$ and $q \in \Theta$. In this case, the operator o is not applicable in K because if the truth value of p were true, then

p would have been satisfied in K since we are adding every atom that is specified to be true in the answer to every state in the $OPEN$ list. However, since p was not satisfied in K, then this means that it was either false in the initial state, or it has been deleted by an operator previously. Therefore, p must be false in the world, and o cannot be applied in K.

- We have not issued a query for p before. In this case, if p is askable — i.e., $p \in W$ for p, then we issue a new query for p. Therefore, we must again defer our decision on the applicability of o in K.
- Our final case is the one in which we have not issued a query for p before, but p is not askable either. Then it immediately follows that the operator cannot be applied in K.

Note that even if the precondition p is satisfied in K, there are may be additional information related to p that cannot be inferred in K. In such cases, we must query to the related web services about p regardless of what we inferred in K, in order to ensure the completeness of our approach.

The FindApplicableOp subroutine in Figure 1 returns the set of all applicable ground operator instances in K. We then update the partial plan π by inserting o into it. Finally, we add the tuple $(K', T', \emptyset, \pi')$ into the the $OPEN$ list. In doing so, note that we implicitly set $\mu = \emptyset$ for the following reason. In a later iteration, suppose the planner selected the tuple (K', T', μ, π') from the $OPEN$ list, where $\mu = \emptyset$. Then, it considers all of the operators applicable to the task t it chooses from T'. If $\mu \neq \emptyset$ is such a tuple, then this means that this tuple represents a set of defered decisions about the application of a set of operators so it should only consider the operators in μ at this iteration. Otherwise, the planner may try to apply the operators it already considered for this tuple. This yields an incorrect behavior since it possibly ends up with an infinite loop, violating the completeness of the algorithm.

The FindDeferrableOp subroutine in Figure 1 performs the checks explained above, and returns the set of all ground operator instances that are not applicable in K at this point, but we may issue queries about their preconditions; therefore, we *defer* our decision on them. We then create the set of queries that needs to be issued for these operators – in the pseudocode, the subroutine GenerateQueries generates a new query for the preconditions of each of the deferrable operators.

The WSC-SHOP2 algorithm also uses two subroutines called ApplyOperator and ApplyMethod. The former subroutine takes as input the current state and a ground operator instance, and it outputs the successor state that arises from applying the operator instance in the current state. The latter subroutine takes as input the current state, the current task, the current task network, a ground method instance for the current task, and a variable substitution. It outputs the successor task network that arises from applying the method instance to the current task in the current state.

Search Strategies in WSC-SHOP2

WSC-SHOP2 allows for exploiting different sorts of search strategies by using different insertion and selection mechanisms for manipulating its OPEN list. As we will demonstrate in the following discussion, each such search strategy corresponds to a different way for gathering information during planning.

As an example, WSC-SHOP2 can exploit a *First-In-Last-Out (FILO)* search strategy as follows. The planner always selects the first tuple in the OPEN list. If it can decompose a task without issuing any queries — i.e., when it does not defer the decision whether a task can be decomposed or not —, then it inserts such tuples at the front of the OPEN list. Otherwise, it inserts the tuples it defers at the end of the OPEN list. Note that this strategy corresponds to the following information-gathering methodology: when the planner issues a query, it does not block the planning process by waiting for an answer; instead, searches the other parts of the search space in a depth-first manner for other possible solutions in the search space.

As another example, suppose we want planner to block the search until a query is answered. Then, the planner can follow this methodology by manipulating the OPEN list as follows. We constrain the number of tuples in the OPEN list that correspond to a query to be at most 1 at any time during planning. In this case, if the planner always inserts a tuple in the front to OPEN, and it always selects the first tuple in OPEN. If the first tuple corresponds to a query that has not been answered before, then it waits until it is answered without considering the other tuples in OPEN – if there are any. We call this strategy the ISSUE-WAIT-CONTINUE (IWC) strategy.

Note that, in general, we can set the number of tuples in the OPEN list that correspond to a query to a specified constant before the planning process. The rationale behind this strategy would be that since every query requires a memory space and a network channel must be opened for the web service corresponding the queries issued by the planner, the planner may not have enough space to keep all of the tuples in the OPEN list that correspond to the queries in the ASKED list. In such cases, if the number of queries issued by the planner reaches the specified limit, and the planner needs to issue a new query, it simply has to wait until one the previously-issued queries gets answered.

As a final example, WSC-SHOP2 can also exploit a *First-In-First-Out (FIFO)* search strategy as follows. The planner always inserts a tuple in the front of the OPEN list. It always selects the first tuple from the front of the OPEN list, for which a query has been issued before and an answer has been received for that query at this point. For example, if there was a query corresponding the first tuple in the OPEN list, and an answer is available at this point, then the planner selects this tuple and removes it from the OPEN list for proceeding with planning along the branch that this tuple belongs in the search space. On the other hand, if there are still no answers for that query, then the planner leaves this tuple at the place where it is in the OPEN list and checks the next tuple. It proceeds in this manner until either it finds a tuple (K, T, μ, π) in the OPEN list for which there are no queries (i.e., the planning can proceed from this point without any extra information than we have already in K), or it finds a tuple for which a query has been issued and answered. If it reaches to the end of OPEN list without finding such a tuple, it re-starts from the front of the list in the same way. Note that since all of the queries require a finite time to get answers, this process always terminates.

Formal Properties of WSC-SHOP2

We first establish the correctness of our algorithm.

Theorem 1 (Soundness and Completeness) *Let $I = (K, A, T, O, M)$ be an incomplete-information planning problem and let W be the set of all ground instances of the atoms in A. If WSC-SHOP2 returns a plan, then that plan is a solution for every complete-information planning problem that is consistent with I. If WSC-SHOP2 does not return a plan, then there exists at least one complete-information planning problem P that is consistent with I, and P is unsolvable (i.e., no plans for P exist).*

We let $X(I)$ be the set of all solutions returned by any of the non-deterministic traces of WSC-SHOP2 on the incomplete-information planning problem I. Furthermore, we let $X^*(I)$ be the shortest solution in $X(I)$. We now establish our first informedness theorem:

Theorem 2 (Informedness Theorem 1) *Let $I_1 = (K_1, A_1, T, O, M)$ and $I_2 = (K_2, A_2, T, O, M)$ be two incomplete-information planning problems and let W_1 and W_2 be the sets of all ground instances of the atoms in A_1 and A_2, respectively. Then we have $X(I_1) \subseteq X(I_2)$, if $K_1 \cup W_1 \subseteq K_2 \cup W_2$.*

Corollary 1 (Convergence toward Optimality) *Let $I_1 = (K_1, A_1, T, O, M)$ and $I_2 = (K_2, A_2, T, O, M)$ be two incomplete-information planning problems and let W_1 and W_2 be the sets of all ground instances of the atoms in A_1 and A_2, respectively. Then, the length of the plan $X^*(I_2)$ is less than or equal to the length of the plan $X^*(I_1)$, if $K_1 \cup W_1 \subseteq K_2 \cup W_2$.*

Let $I = (K, A, T, O, M)$ be an incomplete-information planning problem. For the rest of this section, we will assume that there are constants c, p, q, m, k, b, and d such that c is the probability of a task being composite, p is the probability of an atom a being true, q is the probability of the truth-value of an atom a being known to WSC-SHOP2, m is the average number of methods that are applicable to a composite task, k is the average number of distinct preconditions for the methods in D, b is the number of successor subtasks, and d is the depth of the solution tree produced by WSC-SHOP2.

The solution tree produced by WSC-SHOP2 is an AND-OR tree, in which the AND branches represent

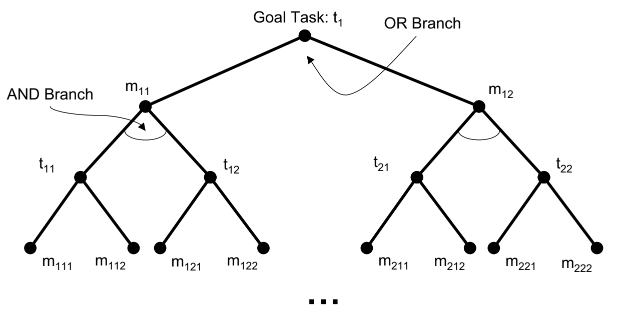

Figure 2: The solution tree that WSC-SHOP2 generates given an incomplete-information planning problem.

the task decompositions whereas the OR branches represent different possible methods whose heads match with a particular task. Without loss of generality, we assume that the solution tree of WSC-SHOP2 is a complete AND-OR tree as shown in Figure 2. Furthermore, we suppose that T contains only one task to be accomplished and we have no negated atoms in the preconditions of the methods in D.

Lemma 1 *Given an incomplete-information planning problem $I = (K, A, T, O, M)$ that satisfies the assumption given above, the probability ρ of WSC-SHOP2 finding a solution for I is*

$$\begin{aligned} \rho_0 &= 1; \textit{ and} \\ \rho_d &= (1-c) + c * [1 - (1-\gamma_d)^b], \end{aligned}$$

where $\gamma_d = (p.q)^k \times (\rho_d - 1)^m$.

As an example, in Figure 5 we set the following values for the independent variables in the equation of Lemma 1, and plotted the amount of information given vs. the convergence probability curves for WSC-SHOP2: $m = 2$, $b = 2$, $p = 0.8$, $d = 10$, $c = 0.8$, and for $k = 1.5, 2, 3, 5$, and 10. The probability of an atom being known, q, which is the x-axis of Figure 5, varied between 0.0 and 1.0. As shown in the figure, the probability of WSC-SHOP2 finding a solution increases with the increasing amount of information available.

The following theorem establishes our second informedness result.

Theorem 3 (Informedness Theorem 2) *Let $I_1 = (K_1, A_1, T, O, M)$ and $I_2 = (K_2, A_2, T, O, M)$ be two incomplete-information planning problems satisfying the assumption given earlier, and let W_1 and W_2 be the sets of all ground instances of the atoms in A_1 and A_2, respectively. Furthermore, let ρ_1 and ρ_2 be the probabilities of WSC-SHOP2 finding solutions for I_1 and I_2, respectively. Then, $\rho_1 \leq \rho_2$, if $K_1 \cup W_1 \subseteq K_2 \cup W_2$.*

Preliminary Experimental Evalution

For our experiments, we wanted to find out how useful our framework would be for planning with incomplete information about the initial state. In this respect, we have implemented a prototype of the WSC-SHOP2

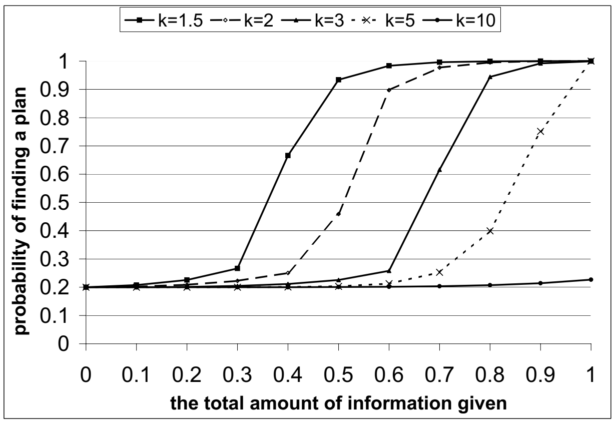

Figure 3: The probability of WSC-SHOP2 finding a plan with the assumptions of the lemma, as a function of the percentage of the atoms made known to it.

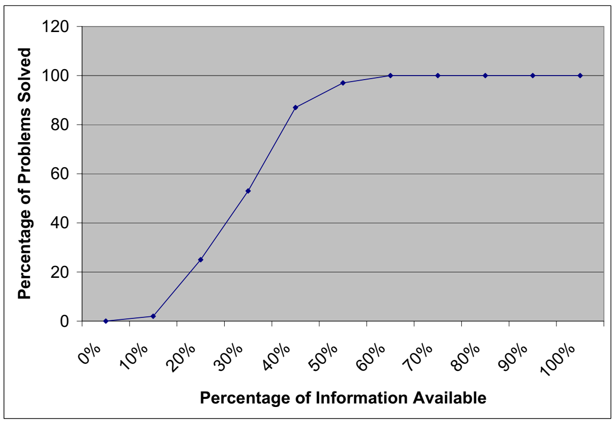

Figure 4: The percentage of times WSC-SHOP2 could find plans for the Logistics problems, as a function of the amount of information available during planning.

algorithm in the SHOP2 planning system. We have designed the following two experiments with this prototype: (1) testing whether WSC-SHOP2 would have a higher probability of solving incomplete-information planning problems with the increasing amount of information available, and (2) comparing two different search strategies that we implemented in WSC-SHOP2.

We ran our experiments on a Sun SPARC Ultra1 machine with 192MB memory running Solaris 8. For our experiments, we have implemented a simulation program for our prototype in order to generate the random response times for the queries issued by WSC-SHOP2.

Experimental Case 1. In this case, we have investigated how the number of solutions (i.e., plans) found by the WSC-SHOP2 algorithm is affected by the amount of the information availble. In these experiments, we used a variant of the Logistics planning domain, which is one of the well-known benchmarks in AI planning. In this domain, there are a number of cities and in each city a number of places. The goal is to move a number of packages from their initial places to their respective destinations using vehicles of different kinds such as trucks and airplanes. We have assumed the existence of Web services that provide information about the vehicles and the packages (e.g., information about their locations, their availability, etc.).

In these experiments, we have implemented the FILO strategy of the previous section in WSC-SHOP2, and run the planner on 100 randomly-generated problem instances in the Logistics domain. For each problem instance, we ran WSC-SHOP2 several times as described below, varying the amount of information available about the initial state. We did this by varying the following quantities: $|K|$, the number of atoms of S that were given initially; and $|A|$, the amount of atoms of S that were made available in the askable list, where S is the set of all possible ground atoms in the domain.

We measured the percentage of times that WSC-SHOP2 could find plans, as a function of the quantity $\frac{|K \cup W|}{|S|}$. The fraction $\frac{|K \cup W|}{|S|}$ is the fraction of atoms about the initial state that are available during planning. We varied $\frac{|K \cup W|}{|S|}$ from 0% to 100% in steps of 10%, and we performed 100 runs of WSC-SHOP2 for each value of $\frac{|K \cup W|}{|S|}$. The results, shown in Figure 4, show that the success rate for WSC-SHOP2 increased as we increased $\frac{|K \cup W|}{|S|}$, showing curves similar to the ones developed in our theoretical study (see Figure 3. WSC-SHOP2 was able to solve 100% of the problem instances even when $\frac{|K \cup W|}{|S|}$ was as low as 60%.

Experimental Case 2. In our experiments, we also aimed to compare how the planning time is affected by different search strategies that can be implemented in WSC-SHOP2. More specifically, we wanted to test how planning time would change when the planner searches different branches of the search space while queries are running, compared to the case where the planner waits for the answers for those queries. In this respect, in addition to the FILO strategy, we have implemented the IWC strategy of the previous section in WSC-SHOP2.

We have tested WSC-SHOP2 with both strategies on the domain used in (Wu *et al.* 2003), which is based on the scenario described in the Scientific American article about the Semantic Web (Berners-Lee, Hendler, & Lassila 2001). This scenario describes two people who are trying to take their mother to a physician for a series of treatments and follow-up meetings. The planning problem is to come up with a sequence of appointments that will fit in to everyone's schedules, and at the same time, to satisfy everybody's constraints and preferences.

We have created 11 random problems in this domain and we have run both strategies 10 times in each problem. We have set the time limit for planning as 15 minutes. Table 1 reports the average CPU times that both search strategies required in these experiments. Two of the runs with WSC-SHOP2/FILO required more than 15 minutes to find a solution; therefore, we did

Table 1: Comparison of the average CPU times (in secs.) of the two search strategies in WSC-SHOP2.

Problem	WSC-SHOP2/IWC	WSC-SHOP2/FILO
S0	24.85	11.02
S1	131.74	13.75
S2	9.94	12.17
S3	9.99	32.57
S4	48.32	13.13
S5	10.19	11.50
S6	18.58	11.50
S7	10.54	14.36
S8	12.90	14.63
S9	16.33	10.58
S10	210.83	244.52

not include those runs in these results.

These preliminary experiments suggested that both search strategies have their advantages and disadvantages. For example, when it takes a very long time to get the responses for the queries, our experiments showed that it is very important for the planner not to wait until they get responded; instead, the FILO was able to find other solutions in the search space more quickly.

However, our experiments showed that there are cases where the FILO strategy might be inefficient compared to IWC. In particular, when FILO issues a query q, it continues searching for a plan, and does not check if q got answered during this search. The only way it considers the answer for q is when it cannot find a solution in the other parts of the search space – either because there are no solutions, or because it issues other queries there. This has some drawbacks. First, if q gets answered really fast and the answer yields to a solution, WSC-SHOP2 unnecessarily searches for other solutions. Second, if there are only a few solutions in the search space, then the planner can take a very long time to find a solution since it does not consider the answers for the relevant queries immediately when they are available. For example, in two of our experimental problems, WSC-SHOP2 could not find a solution for within the alloted time since there were only a few solutions to these problems, and WSC-SHOP2 was searching almost the entire space to find one of them.

Related Work

(McIlraith & Son 2002) describes an approach to building agent technology based on the notion of generic procedures and customizing user constraints. This work extends the Golog language to enable programs that are generic, customizable and usable in the context of the Web. Also, the approach augments a ConGolog interpreter that combines online execution of information-providing services with offline simulation of world-altering services. Although this approach is similar to our work, we suspect that, in general, a logic-based approach will not be as efficient as a planning approach. We intend to test this hypothesis emprically in our future work.

In the AI planning literature, there are various approaches to planning with incomplete-information. These approaches are generally developed for gathering information during execution, by inserting sensing actions in the plan during planning time and by generating conditional plans, conditioned on the possible pieces of information that can be gathered by those actions. (Bertoli *et al.* 2001) presents a planning system that implements these ideas. (Etzioni *et al.* 1992) presented a planning language called UWL, which is an extension of the STRIPS language, in order to distinguish between (1) the world-altering and the observational effects of the actions, and (2) the goals of satisfaction and the goals of information. (Golden, Etzioni, & Weld. 1996) describes an algorithm for planning with both complete and incomplete information, called the XII planner, which is an extension of UCPOP (Penberthy & Weld 1992). This planner uses a 3-valued logic for representing and reasoning about the incomplete world-state information, and it plans for sensing actions for information gathering during execution.

WSC-SHOP2 differs from these approaches in that it does not explicitly plan for sensing actions to obtain information at execution time. Instead, it is a planning technique for gathering the necessary information during planning time. This property enables WSC-SHOP2 to (1) generate simple plans since observational actions are not included in the plan, and (2) interact with external information sources and clear out the "unknown"s during planning time as much as possible. In this respect, it is very suitable for the problem of composing Web Services, as discussed in this paper.

(Barish & Knoblock 2002) describes a speculative execution method for generating information-gathering plans in order to retrieve, combine, and manipulate data located in remote sources. This technique exploits certain hints received during plan execution to generate speculative information for reasoning about the dependencies between operators and queries later in the execution. WSC-SHOP2 differs from this method in two aspects: (1) it searches different branches of the search space when one branch is blocked with a query execution, and (2) it runs the queries in planning time, whereas speculative execution was shown to be useful for executing plans. Combining speculative execution with our approach would enable us to run queries down in a blocked branch; however, since the time spent for a query is lost when speculations about that query are not valid, it is not clear if combining the two approaches will lead to significant results.

Conclusions and Future Work

In our previous work (Wu *et al.* 2003), we have shown how a set of OWL-S service descriptions can be translated to a planning domain description that can be used by SHOP2. The corresponding planning problem for a service composition problem is to find a plan for the task that is the translation of a composite process.

This translation methodology differentiates between the information-gathering services, i.e. services that have only output but no effects, and the world-altering services, i.e. services that have effects but no outputs. The preconditions of information-gathering services include an external function to call to execute the service during planning and add the information to the current state. This can be seen as a specialized application of the WSC-SHOP2 framework presented in this paper. The queries to the information sources correspond to atomic service executions, and they are explicitly specified as special primitive tasks in a domain description.

WSC-SHOP2 helps us to overcome the following limitations in our previous work:

- The information providing services do not need to be explicitly specified in the initial description. The query mechanism can be used to select the approriate Web service on the fly when the information is needed. Note that matching service description does not need to be an atomic service. A composite service description matching the request could be recursively fed to the planner, or a different planning algorithm altogether could be used to plan the information gathering.
- The planning process does not need to wait for the information gathering to finish and can continue planning while the service is still executing.

The restriction that information-gathering services cannot have world-altering effects can be lifted easily as follows. Note that without this restriction, if an external process changes the state of the world, the generated plans may not be correct anymore. However, this restriction is not necessary when the effects of the information-gathering service do not interact with the plan being sought for. For example, a world-altering effect of the information providing service could be charging a small amount of fee for the service. If the planner is looking for a plan that has nothing to do with money, then it would be safe to execute this service and change the state of the world. In general, this safety can be established when the original planning and information-gathering problems correspond to two disconnected task networks that can be decomposed into primitive tasks without any kind of interaction.

Although, we have concentrated on only information gathering during planning so far, it is also important to gather information during the execution. We hypothesize that it should possible to extend the WSC-SHOP2 formalism for this purpose as follows. WSC-SHOP2's queries are about the initial state of a planning problem, to ensure that the planner is sound and complete. However, in principle, we should be able to issue queries about any state during planning. This would allow us to insert queries, similar to sensing actions, in the plan generated by the planner, leading conditional plans to be generated by the planner based on the possible answers to such queries. We are currently exploring this possibility and its effects on the soundness and completeness of the planning process.

Acknowledgments

We would like to thank our reviewers for their very helpful comments on this paper. This work was supported in part by the following grants, contracts, and awards: Air Force Research Laboratory F30602-00-2-0505, Army Research Laboratory DAAL0197K0135, Naval Research Laboratory N00173021G005, the Defense Advanced Research Projects Agency (DARPA), Fujitsu Laboratory of America at College Park, Lockheed Martin Advanced Technology Laboratories, the National Science Foundation (NSF), the National Institute of Standards and Technology (NIST), and NTT Corp. The opinions expressed in this paper are those of authors and do not necessarily reflect the opinions of the funders.

References

Barish, G., and Knoblock, C. A. 2002. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 184–193. Menlo Park, CA: AAAI Press.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The Semantic Web. *Scientific American* 284(5):34–43.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI 2001*, 473–478.

Etzioni, O.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proceedings of KR-92.*

Golden, K.; Etzioni, O.; and Weld., D. 1996. Planning with execution and incomplete information. Technical Report TR96-01-09, Department of Computer Science, University of Washington.

McIlraith, S., and Son, T. 2002. Adapting Golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning.*

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20.

OWL Services Coalition. 2003. OWL-S: Semantic markup for web services. OWL-S White Paper http://www.daml.org/services/owl-s/0.9/owl-s.pdf.

Penberthy, J. S., and Weld, D. 1992. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Proc. KR-92.*

Wu, D.; Parsia, B.; Sirin, E.; Hendler, J.; and Nau, D. 2003. Automating daml-s web services composition using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003).*