Information Gathering During Planning for Web Service Composition

Ugur Kuter[1], Evren Sirin[1], Dana Nau[1], Bijan Parsia[2], and James Hendler[1]

[1] Department of Computer Science,
University of Maryland, College Park, MD 20742, USA,
{ukuter,evren,nau,hendler}@cs.umd.edu
[2] MIND Lab, University of Maryland,
8400 Baltimore Ave., College Park, MD 20742, USA
bparsia@isr.umd.edu

Abstract. Hierarchical Task-Network (HTN) based planning techniques have been applied to the problem of composing Web Services, especially when described using the OWL-S service ontologies. Many of the existing Web Services are either exclusively information providing or crucially depend on information-providing services. Thus, many interesting service compositions involve collecting information either during execution or during the composition process itself. In this paper, we focus on the latter issue. In particular, we present ENQUIRER, an HTN-planning algorithm designed for planning domains in which the information about the initial state of the world may not be complete, but it is discoverable through plan-time information-gathering queries. We have shown that ENQUIRER is sound and complete, and derived several mathematical relationships among the amount of available information, the likelihood of the planner finding a plan, and the quality of the plan found. We have performed experimental tests that confirmed our theoretical results and that demonstrated how ENQUIRER can be used in Web Service composition.

1 Introduction

Web Services are Web accessible, loosely coupled chunks of functionality with an interface described in a machine readable format. 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.

OWL-S [1] provides a set of ontologies to describe Web Services in a more expressive way than allowed by the Web Service Description Language (WSDL). In OWL-S, services can be described as complex or atomic processes with preconditions and effects. This view enables us to translate the process-model constructs

S.A. McIlraith et al. (Eds.): ISWC 2004, LNCS 3298, pp. 335–349, 2004.

directly to HTN methods and operators [2]. Thus, it is possible to use HTN planners to plan for composing Web Services that are described in OWL-S.

However, using AI planning techniques for Web Services composition introduces some challenges. Traditional planning systems assume that the planner begins with complete information about the world. However, in service composition problems, most of the information (if it is available at all) must be acquired from Web Services, which may work by exposing databases, or may require prior use of such information-providing services. For example, an appointment making service might require the planner to determine an available appointment time first. In many cases, it is not 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 gather information during planning.

In this paper, we describe ENQUIRER, an HTN-planning algorithm that can solve Web Service composition problems that require gathering information during the composition process. ENQUIRER is based on the SHOP2 planning system [3], and designed for planning domains in which the information about the initial world state may not be complete. In such cases, ENQUIRER issues queries to obtain the necessary information, it postpones all decisions related to that information until a response comes in, and it continues examining alternative branches of the search space. By gathering extra information at plan time, the planner 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, ENQUIRER's non-blocking strategy is sometimes able to dramatically improve the time to find a plan.

We also provide the sufficient conditions to ensure the soundness and completeness of ENQUIRER, derive a recurrence relation for the probability of ENQUIRER finding a plan, and prove theoretical results that give mathematical relationships among the amount of information available to ENQUIRER and the probability of finding a plan. These relationships are confirmed by our experimental evaluations. We describe how the generic ENQUIRER algorithm can be used to solve the problem of composing Web Services, and test the efficiency of the algorithm on real-world problems.

2 Motivations

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 [1] process model to the point where the OWL-S process model constructs can be directly mapped to HTN methods and operators. In our previous work [2], we have shown how such a translation can be done for SHOP2 [3]. In this work, we have kept the basic SHOP2-language mapping intact, and focused on extending the way SHOP2

deals with plan-time information gathering.[1] We call the extended algorithm ENQUIRER.

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 heterogeneous 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 planning.* While not all the information relevant to a problem may have already been gathered, it will often be the case that it 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 inaccessible 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.

ENQUIRER is designed to address all of the issues above. In the subsequent sections, we first give a brief background on the HTN Planning and the SHOP2 planning system, and then we present the ENQUIRER planning algorithm, as well as our theoretical and experimental evaluations of it.

3 Background: HTN Planning and SHOP2

The purpose of an HTN planner is to produce a sequence of actions that perform some activity or *task*. The description of a planning domain includes a set of planning *operators* and *methods*, each of which is a prescription for how to decompose a task into its *subtasks* (smaller tasks). The description of a planning problem contains an initial state as in classical planning. Instead of a goal formula, however, there is a partially-ordered set of tasks to accomplish.

[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.

Planning proceeds by decomposing tasks recursively into smaller and smaller subtasks, until *primitive tasks*, which can be performed directly using the planning operators, are reached. For each task, the planner chooses an applicable method, instantiates it to decompose the task into subtasks, and then chooses and instantiates other methods to decompose the subtasks even further. If the constraints on the subtasks or the interactions among them prevent the plan from being feasible, the planning system will backtrack and try other methods.

SHOP2 is an HTN planner that generates actions in the order they will be executed in the world. Its backtracking search considers the methods applicable to the same task in the order they are specified in the knowledge base given to the planner. This feature of the planner allows for specifying *user preferences* among such methods, and therefore, among the solution that can be generated using those methods. For example, Figure 1(c) shows a possible user preference among the three methods for the task of delivering a box from UMD to MIT.

An example is in order. Consider a Delivery Domain, in which the task is to deliver a box from one location to another. Figure 1(a) shows two SHOP2 methods for this task: *delivering by car*, and *delivering by truck*. Delivering by car involves the subtasks of loading the box to the car, driving the car to the destination location, and unloading the box at the destination. Note that each method's preconditions are used to determine whether or not the method is applicable: thus in Figure 1(a), the *deliver by car* method is only applicable if the delivery is to be a fast one, and the *deliver by truck* method is only applicable if it is to be a slow one. Now, consider the task of delivering a box from the University of Maryland to MIT and suppose we do not care about a fast delivery. Then, the *deliver by car* method is not applicable, and we choose the *deliver by truck* method. As shown in Figure 1(b), this decomposes the task into the following subtasks: *(1)* reserve a truck from the delivery center at Laurel, Maryland to the center at Cambridge, Massachusetts, *(2)* deliver the box from the University of Maryland to Laurel, *(3)* drive the truck from Laurel to Cambridge, and *(4)* deliver the box from Cambridge to MIT. For the two delivery subtasks produced by this decomposition, we must again consider our delivery methods for further decomposing them until we do not have any other task to decompose.

During planning, the planner evaluates the preconditions of the operators and methods with respect to the world state it maintains locally. It is assumed that planner has all the required information in its local state in order to evaluate these preconditions. For example, in the delivery example, it is assumed that the planner knows all the distances between the any initial and final locations so that it can determine how long a truck will be reserved for a delivery task. Certainly, it is not realistic to assume that planner will have all this information before the planning process starts. Considering the amount of information available on the Web is huge, planner should gather the information as needed by the planning process. Since gathering information may take some considerable amount of time, it would be wise to continue planning while the queries are being processed. For example, the planner can continue planning with the *truck delivery* method until the answer to the query about train schedules has been received.

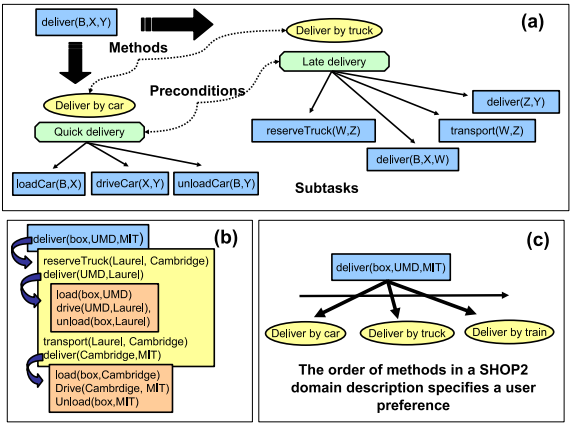


Fig. 1. Delivery planning example.

4 Definitions and Notation

We use the same definitions for logical atoms, states, task symbols, tasks, task networks, actions, operators, methods, and plans as in SHOP2. The ENQUIRER planning procedure extends SHOP2 to be able to cope with incomplete information about the initial state of the world. The following definitions establish the framework for the ENQUIRER procedure.

An *askable list* is a set of logical atoms that are eligible for the planner query during the planning process. Note that we do not require the atoms in an askable list to be ground. 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.

A *query* is an expression of the form $(h\ p)$, where h is the unique label of the query and p is a logical atom. Note that we do not require p to be ground. The intent of a query is to gather information during planning about the relation p in the state of the world before planning started (i.e., in the initial state).

Let A be an askable list. We let $\delta(A)$ denote the set of all possible instantiations of the atoms in A. Then, a query $(h\ p)$ is said to be *askable with respect to* A if and only if p unifies with an atom in $\delta(A)$ – i.e., there exists a variable substitution Θ such that $\Theta(p) \in \delta(A)$. An *answer* for an already-asked query $(h\ p)$ is an expression of the form $(h\ R)$ such that $R \subseteq \delta(A)$, and for each $p' \in R$, there exists a substitution Θ such that $\Theta(p) = p'$.

A *complete-information planning problem* is a tuple $P^C = (S, T, D)$, where S is a complete initial state, T is a task network, and D is an HTN-domain description that consists of a set of planning operators O and methods M, respectively. An *incomplete-information planning problem* is a tuple $P^I = (J, A, T, D)$, where J is a set of ground atoms that are initially known, A is an askable list, T is a

task network, and D is an HTN-domain description. An incomplete-information planning problem P^I is *consistent* with a complete-information planning problem P^C if only if S is identical with $J \cup \delta(A)$. Note that $J \cup \delta(A)$ denotes the total amount of information that a planner can possibly obtain while solving P^I.

We define a *service-composition problem* to be a tuple of the form $W = (J, X, C, K)$, where J is a (possibly incomplete) initial state, X is a set of all possible information-providing Web Services that are available during the planning process, C is a (possibly) composite OWL-S process, and K is a collection of OWL-S process models such that we have $C \in K$. We assume that the information-gathering services in X return information only about the initial state, and they do not have any world-altering effects.

An explanation about the set of available services X is in order. When a query $(h\ p)$ for a possibly partially ground atom p is asked, Web Service that answer this query needs to be located. A service whose output or postcondition specification matches with p is said to be a possible match.[2] For example, a query (find-airports airport(US,?x)) can be answered by a service that is specified as (:input ?c - country, :output ?x - airport, :postcondition airport(?c, ?x)). Note that, such a service description can be written in OWL-S 1.1 using the recently added Result and Expression structures.

Let $W = (J, X, C, K)$ be a service-composition problem. We let $\tau(X)$ denote the total amount of information that can possibly be gathered from the Web Services in X – i.e., $\tau(X)$ denotes the set of all possible ground atoms that are specified by the outputs or the postconditions of the services in X. Then, we say that W is *equivalent* to an incomplete-information problem $P^I = (J, A, T, D)$ if and only if $\tau(X) = \delta(A)$, where T is the SHOP2 translation for the OWL-S process C, and $D = \mathsf{TRANSLATE}(K)$ is the HTN-domain description generated by the translation algorithm TRANSLATE of [2].

5 The ENQUIRER Algorithm

ENQUIRER starts with an incomplete initial world state, gathers relevant information during planning, and continues to explore alternative possible plans while waiting for information to come in. The algorithm is shown in Fig.2. The input is an incomplete-information planning problem (J, A, T, D) as defined above. ASKED is the set of all queries that have been issued by ENQUIRER and that have not been answered yet. ANS is the set of all queries for which answers have been received. The OPEN list is the set of triples of the form (J, T, π), where J is a (possibly) incomplete state, T is a task list, and π is a plan. Intuitively, the OPEN list holds the information on the leaf nodes of the search tree generated during the planning process. Initially, each of these lists is the empty set, except the OPEN list which is initialized to $\{(J, T, \pi)\}$.

At each iteration of the planning process, we first check if the OPEN list is empty. If so, then we report *failure* since this means that every branch in the

[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.

```
procedure ENQUIRER(J, A, T, D = (O, M))
 ASKED ← ∅; ANS ← ∅; π ← ∅; OPEN ← {(J, T, π)}
 loop
  if OPEN = ∅ then return(failure)
  OPEN ← {(J', T, π) | (J, T, π) ∈ OPEN, (h p) ∈ ASKED, an answer (h R) has
                       been received for (h p), and J' ← ProcessAnswers(J, R)}
  for each query (h p) ∈ ASKED that has been answered recently
   remove (h p) from ASKED and insert it into ANS
  select a tuple (J, T, π) from OPEN and remove it
  if T = ∅ then return(π)
  nondeterministically choose a task t ∈ T that has no predecessors
  if t is a primitive task then
   U ← {(o, Θ) | o ∈ O, and ∃ a substitution Θ such that Θ(t) = Θ(head(o))}
   S ← {(J', Θ(T − {t}), π ∪ {Θ(o)}) | (o, Θ) ∈ X, Θ(o) is applicable in J, and
                                       J' ← ApplyOp(J, Θ(o))}
  else
   U ← {(m, Θ) | m ∈ M, and ∃ a substitution Θ such that Θ(t) = Θ(head(m))}
   S ← {(J, T', π)|(m, Θ) ∈ X, Θ(m) is applicable in J, and
                      T' ∈ ApplyMeth(J, T, t, m, Θ)}
  OPEN ← OPEN ∪ S
  Q ← {(h p) | (u, Θ) ∈ U, p is a precondition of u such that Θ(p) is askable w.r.t. A,
              Θ(u) is not applicable in J, and ∄ a query (h Θ(p)) ∈ ANS}
  if Q ≠ ∅ then
     ASKED ← ASKED ∪ {(h p) | (h p) ∈ Q, and (h p) ∉ ASKED}
     OPEN ← OPEN ∪ {(J, T, π)}
```

Fig. 2. The ENQUIRER algorithm.

search space is exhausted without success. Otherwise, we start processing the answers for our previously-issued queries that have arrived at this point. Let $(h\ R)$ be such an answer for the query $(h\ p)$. Then, for every triple (J, T, π) in OPEN, we insert a ground atom $p' \in R$ into J if $p' \notin J$ and there is no action $a \in \pi$ that makes p' false in the world. Note that this condition is necessary to ensure the correctness of our algorithm since ENQUIRER's queries provide information about the initial state of the world: if we insert an atom into J that is in the delete-list of an action in π, then J becomes inconsistent. In Fig.2, the subroutine ProcessAnswers is responsible for checking this condition and updating the states in OPEN accordingly.

After processing the answered queries, the next step is to select a tuple (J, T, π) from OPEN, and remove it. 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 select each operator o that matches to the task t — i.e., there exists a

variable substitution Θ such that $\Theta(t) = \Theta(head(o))$. The operator instance $\Theta(o)$ is applicable in J, if all of its preconditions are ground and satisfiable in J. Otherwise, it is not applicable. In the former case, we generate the next state J' and the next task network T' by applying $\Theta(o)$ in J and by removing t from T, respectively. We update the current partial plan π by adding $\Theta(o)$ to it, and update the OPEN list with the tuple $(J', T', \pi \cup \{\Theta(o)\})$.

If the operator instance $\Theta(o)$ is not applicable in J then this means that there is a precondition p of $\Theta(o)$ such that either p is not ground, or p is ground but cannot be satisfied in J. In either case, we cannot immediately come to a conclusion on whether $\Theta(o)$ is applicable in J or not. Instead, we must do the following. We first check if p is askable with respect to the askable list A; if it is not then there is no way we can obtain further information about it so we conclude that it is false. Otherwise, we check if there is a query for p in the ANS list. If so, then this means that p was queried before and an answer has been received for it. Then, since we still cannot infer any information about p from J, p must be false in the world and we cannot apply $\Theta(o)$ in J.

If there is no query related to p in ANS, then this means one of the following conditions holds: we have queried p before but no answer has come yet, or this is the first time we need to query it. To determine which case we are in, we check the ASKED list if there is a query for p in it. If so, then the former is the case and we defer our decision on the applicability of $\Theta(o)$ in J. If there is no query regarding to p in ASKED, then we create a new query for p and insert it into ASKED. In this case, we again defer our decision on $\Theta(o)$.

Note that if the precondition p is unground in $\Theta(o)$, there may be additional information related to p that cannot be inferred in J, even if p is satisfiable in that state. For example, in the delivery domain, an unground precondition may be used to get the schedule of all trains for a particular route. In the current state, there may already be some related information, e.g. results of a previous queries returned by one Web Service invocation. However, there may still be other Web Services that will return additional information that could be crucial for finding a plan. Therefore, in order to ensure completeness, ENQUIRER queries all the related Web Services about p regardless of what is inferred in J.

The ENQUIRER algorithm also uses two subroutines called ApplyOp and ApplyMeth. ApplyOp takes as input the current state J and the action to be applied in J, and outputs the successor state that arises from applying that action in J. The definition for ApplyMeth is more complicated; intuitively, it takes a method m and a non-primitive task t to be decomposed by m, and it modifies the current task network T by removing t and adding its subtasks to T (for the full definition of this subroutine, see [3]).

6 Formal Properties of ENQUIRER

Due to space limitations, we omit the proofs of our theorems here. They can be found at [4]. We first establish the soundness and completeness of our algorithm.

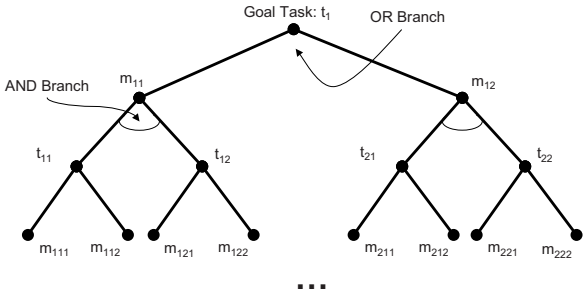


Fig. 3. The solution tree that ENQUIRER generates for an incomplete-information planning problem.

Theorem 1. *Let $W = (J, X, C, K)$ be a service-composition problem, and $P^I = (J, A, T, D)$ be an incomplete-information planning problem that is equivalent to W. If ENQUIRER returns a plan for P^I, then that plan is a solution for every complete-information planning problem that is consistent with P^I. If ENQUIRER does not return a plan, then there exists at least one planning problem P^C that is consistent with P^I, and P^C is unsolvable (i.e., no plans for P^C exist).*

The following theorem establishes the correctness of our approach.

Theorem 2. *Let $W = (J, X, C, K)$ be a service-composition problem, and $P^I = (J, A, T, D)$ be an incomplete-information planning problem that is equivalent to W. ENQUIRER returns a plan π for P^I if and only if π is a composition for W.*

Let $\chi(P^I)$ be the set of all solutions returned by any of the non-deterministic traces of ENQUIRER on an incomplete-information problem P^I. Furthermore, we let π_{P^I} be the shortest solution in $\chi(P^I)$, and let $|\pi_{P^I}|$ denote the length of that solution (i.e., plan). We now establish our first informedness theorem:

Theorem 3. *Let $P_1^I = (J_1, A_1, T, D)$ and $P_2^I = (J_2, A_2, T, D)$ be two incomplete-information planning problems. Then $\chi(P_1^I) \subseteq \chi(P_2^I)$, if $J_1 \cup \delta(A_1) \subseteq J_2 \cup \delta(A_2)$.*

A corollary immediately follows:

Corollary 1. *Let $P_1^I = (J_1, A_1, T, D)$ and $P_2^I = (J_2, A_2, T, D)$ be two incomplete-information planning problems. Then $|\pi_{P_2^I}| \leq |\pi_{P_1^I}|$, if $J_1 \cup \delta(A_1) \subseteq J_2 \cup \delta(A_2)$.*

Let $P^I = (J, A, T, D)$ 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 a ground atom a being true, q is the probability of the truth-value of an atom a being known to ENQUIRER, 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 ENQUIRER.

The solution tree produced by ENQUIRER is an AND-OR tree, in which the AND branches represent 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 ENQUIRER is a complete AND-OR tree as shown in Fig.3. 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 $P^I = (J, A, T, D)$ that satisfies the assumption given above, the probability ρ of ENQUIRER finding a solution for P^I is*

$$\rho_0 = 1; \text{ and}$$
$$\rho_d = (1 - c) * (p.q)^k + c * [1 - (1 - \gamma_d)^m],$$

where $\gamma_d = (p.q)^k \times (\rho_{d-1})^b$.

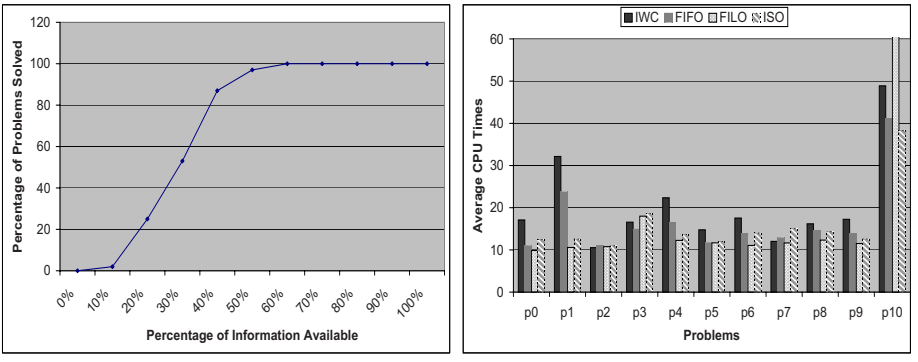
The following theorem establishes our second informedness result.

Theorem 4. *Let $P_1^I = (J_1, A_1, T, D)$ and $P_2^I = (J_2, A_2, T, D)$ be two incomplete-information planning problems satisfying the assumption given earlier. Furthermore, let ρ_1 and ρ_2 be the probabilities of ENQUIRER finding solutions for P_1^I and P_2^I, respectively. Then, $\rho_1 \leq \rho_2$, if $J_1 \cup \delta(A_1) \subseteq J_2 \cup \delta(A_2)$.*

7 Experimental Evaluation

For our experiments, we wanted to investigate (1) how well ENQUIRER would perform in service-composition problems where some of the information was completely unavailable, and (2) the trade-off between the time performance and the desirability of the solutions generated by the planner by using different query-processing strategies. We built a prototype implementation of the ENQUIRER algorithm that could be run with different query strategies. We also built a simulation program that would generate random response times for the queries issued by ENQUIRER. We ran our experiments on a Sun SPARC Ultra1 machine with 192MB memory running Solaris 8.

Experimental Case 1. In this case, we have investigated how the number of solutions (i.e., plans) found by the ENQUIRER algorithm is affected by the amount of the information available during planning. In these experiments, we used a set of Web Service composition problems on the Delivery domain described in Section 3. In these problems, a delivery company is trying to arrange the shipment of a number of packages by coordinating its several local branches. The company needs to gather information from the branches about the availability of vehicles (i.e., trucks and planes) and the status of packages. Such information is provided by Web Services, and the goal is to generate a sequence of confirmation messages that tells the company that every package is delivered to its destination.



(a) **(b)**

Fig. 4. Chart (a) shows the percentage of times ENQUIRER could find plans for the Delivery problems, as a function of the amount of information available during planning. Chart (b) shows the comparison of the average CPU times (in secs.) of the four query-processing strategies in ENQUIRER.

In these experiments, we have ENQUIRER run the planner on 100 randomly-generated problem instances. For each problem instance, we ran ENQUIRER several times as described below, varying the amount of information available about the initial state by varying the following quantities: $|J|$, 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 ENQUIRER could find plans, as a function of the quantity $\frac{|J \cup A|}{|S|}$. The fraction $\frac{|J \cup A|}{|S|}$ is the fraction of atoms about the initial state that are available during planning. We varied $\frac{|J \cup A|}{|S|}$ from $f = 0\%$ to 100% in steps of 10% as follows: we first randomly chose a set of atoms for $|S|$ such the size of this set is equal to the fraction specified by the particular f value. Then, for each atom in this set, we randomly decided whether the atom should go in J – the incomplete initial state –, or in A – the askable list. Using this setup, we performed 100 runs of ENQUIRER for each value of $\frac{|J \cup A|}{|S|}$. The results, shown in Fig.4(a), show that the success rate for ENQUIRER increased as we increased $\frac{|J \cup A|}{|S|}$. ENQUIRER was able to solve 100% of the problem instances even when $\frac{|J \cup A|}{|S|}$ was as low as 60%.

Experimental Case 2. We also aimed to investigate the comparison between the time performance of the non-blocking nature of the search performed by the ENQUIRER algorithm and the desirability of the plans generated — i.e., how much the generated plans conforms to the user preferences encoded in the domain descriptions given to the algorithm in terms of the ordering among the methods applicable to the same task.

In this respect, we have implemented three variations of the ENQUIRER algorithm. All of these variations are based on how the planning algorithm maintains

its *OPEN* list. The original algorithm, shown in Figure 2, always searches for solutions that can be found regardless of the incompleteness of the information about the initial state: when ENQUIRER issues a query, it does not wait for an answer; instead it continues its search for solutions that can be found without issuing any queries at all. We call this strategy as the Issue-Search-Other (ISO) strategy. Similar to ISO, all the other three strategies are also based on different implementations of the OPEN list. The Issue-Wait-Continue (IWC) strategy involves blocking the search until a query, which is issued recently, gets answered. In the First-In-Last-Out (FILO) strategy, if the planner has asked many queries during its search and several of them have been answered, then it always returns to the point in the search space that corresponds to the most recently-issued query that has been answered. Finally, in FIFO which is a complete symmetric version of FILO, the planner always considers the answers corresponding to the least-recent query it has asked. Due to space limitations, we do not give the details of these strategies; for further information see [4].

In our experiments with these variations of ENQUIRER, we compared their time performances on the domain used in [2], which is partly based on the scenario described in the Scientific American article about the Semantic Web [5]. 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 preferences.

We have created 11 random problems in this domain and we have run all strategies 10 times in each problem. We have set the maximum response time for a query asked by ENQUIRER to be 1 seconds. Fig.4(b) reports the average CPU times required by the four strategies described above.

These results provide several insights regarding the trade-off between time performance of ENQUIRER and the desirability of the solutions it generates under incomplete information. From a performance-oriented point of view, they showed that it is very important for the planner not to wait until its queries get responded, especially when it takes a very long time to get responses for those queries. In our experiments, the IWC strategy performed worst than the others in most of the problems. However, IWC was able to generate the most-desirable solutions for our problems, since it always considers the methods in ENQUIRER knowledge base in the order they were specified.

In most of the problems, the FILO strategy was able to find solutions more quickly than the other strategies; in particular, it was consistently more efficient than IWC. However, these solutions were usually less-desirable ones with respect to the user preferences encoded as orderings among the methods in ENQUIRER knowledge base, since FILO always tends to consider the answers for the most-recently issued queries, which usually correspond to less-desirable preferences because of the queries waiting for an answer.

Our experiments also suggest that the ISO strategy can be particularly useful on planning problems that have few solutions. In particular, the problem $p10$ of our experiments is one such problem, and ISO was able to find solutions for

this problem more quickly than others. The reason for this behavior is that ISO performs a less organized and less structured search compared to other strategies, enabling the planner to consider different places in the search space. On the problem $p10$ which has only 48 solutions compared to the 4248 solutions of the problem $p9$, this characteristic enabled ISO to perform better than the other strategies. However, in most of the problems, the solutions generated by the planner using ISO were not the desirable ones.

8 Related Work

[6] 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 a logic-based approach will not be as efficient as a planning approach. We intend to test this hypothesis empirically in the near future.

In the AI planning literature, there are various approaches to planning with incomplete-information, designed 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. [7] presents a planning system that implements these ideas. [8] 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. [9] describes he XII planner for planning with both complete and incomplete information, which is an extension of UCPOP [10], and is able to generate sensing actions for information gathering during execution.

ENQUIRER 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. As a result, ENQUIRER can generate simple plans since observational actions are not included in the plan, and it can 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.

[11] 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. ENQUIRER 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.

9 Conclusions and Future Work

In our previous work [2], 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 approach 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 ENQUIRER algorithm, when the queries are explicitly specified as special primitive tasks that correspond to atomic service executions.

ENQUIRER overcomes 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 appropriate 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 an entirely different planner could be used to plan for 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.

In this paper, we have assumed that information-providing services cannot have world-altering effects. Otherwise, the correctness of the plans generated cannot be guaranteed since the changes done by the information-providing service may invalidate some of the steps planner already committed to. However, this restriction is not necessary when the effects of the information-gathering services do not interact with the plan being sought for. As an example, consider a service that charges a small amount of fee for the service. If we are 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 problem and information-gathering problem correspond to two disconnected task networks that can be accomplished without any kind of interaction. Verifying that there is no interaction between two problems is a challenging task that we will address in our future work.

ENQUIRER is designed for information gathering at plan time; however, it is important to do so in execution time as well. We hypothesize that it should possible to extend our framework for this purpose as follows. ENQUIRER'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 ramifications on planning.

Acknowledgments. 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, 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

1. OWL Services Coalition: OWL-S: Semantic markup for web services (2003) OWL-S White Paper http://www.daml.org/services/owl-s/0.9/owl-s.pdf.
2. Wu, D., Parsia, B., Sirin, E., Hendler, J., Nau, D.: Automating daml-s web services composition using SHOP2. In: Proceedings of ISWC-2003, Sanibel Island, Florida (2003)
3. Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. Journal of Artificial Intelligence Research **20** (2003)
4. Kuter, U., Sirin, E., Nau, D., Parsia, B., Hendler, J.: Information gathering during planning for web service composition. Technical Report 4616-2004-58, Department of Computer Science, University of Maryland, College Park (2004)
5. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284** (2001) 34–43
6. McIlraith, S., Son, T.: Adapting Golog for composition of semantic web services. In: Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning, Toulouse, France (2002)
7. Bertoli, P., Cimatti, A., Roveri, M., Traverso, P.: Planning in nondeterministic domains under partial observability via symbolic model checking. In: IJCAI 2001. (2001) 473–478
8. Etzioni, O., Weld, D., Draper, D., Lesh, N., Williamson, M.: An approach to planning with incomplete information. In: Proceedings of KR-92. (1992)
9. Golden, K., Etzioni, O., Weld., D.: Planning with execution and incomplete information. Technical Report TR96-01-09, Department of Computer Science, University of Washington (1996)
10. Penberthy, J.S., Weld, D.: UCPOP: A Sound, Complete, Partial Order Planner for ADL. In: Proceedings of KR-92. (1992)
11. Barish, G., Knoblock, C.A.: Planning, executing, sensing, and replanning for information gathering. In: Proceedings of AIPS-2002, Menlo Park, CA, AAAI Press (2002) 184–193