Automatic Web Services Composition Using SHOP2

Dan Wu[+] **Evren Sirin**[+] **James Hendler**[+] **Dana Nau**[+] **Bijan Parsia**[*]
Department of Computer Science
University of Maryland
College Park, MD 20742
[+]{dandan,evren,hendler,nau}@cs.umd.edu
[*]bparsia@isr.umd.edu

Abstract

Semantic markup of Web services will enable the automation of various kinds of tasks, including discovery, composition, and execution of Web services. We describe how an AI planning system (SHOP2) can be used with DAML-S Web service descriptions to automatically compose Web services.

Introduction

As Web services – that is, programs and devices accessible via standard Web protocols – proliferate, it becomes more difficult to find the specific service that can perform the task at hand. It becomes even more difficult when there is no single service capable of performing that task, but there are combinations of existing services that could. Sufficiently rich, machine readable descriptions of Web services would allow the creation of novel, compound Web services with little or no direct human intervention. Semantic Web languages, such as the Web Ontology Language (OWL) (Dean *et al.* 2002) or DAML+OIL(Horrocks *et al.* 2001), provide the foundations for such sufficiently rich descriptions.

In May 2001, the DARPA Agent Markup Language (DAML) Program released the first version of DAML-S (Ankolekar *et al.* 2002), a set of ontologies for describing the properties and capabilities of Web services. The purpose of DAML-S markup of Web services is to support effective automation of various kinds of tasks including Web service discovery, composition, execution, and monitoring.

For our work, we are motivated by issues related to automated Web service composition. One part of DAML-S, namely its process ontology, provides a standard language for describing the composition of Web services. Below, we describe how the SHOP2(Nau *et al.* 2001) planning system can be used with DAML-S Web service descriptions to automatically compose Web services.

This paper is organized in the following manner. In Section 2, we describe a motivating example for our research. In Section 3, we give the background knowledge about DAML-S process ontology and SHOP2. In Section 4, we present our approach for automatic Web services composition. In Section 5, we describe the implementation. In Section 6, we summarize some related work. And finally, in Section 7, we conclude our work and present some future research directions. Throughout this paper, we use the example we described in Section 2 to illustrate some concepts used in our approach. But our work is designed to be domain-independent and is not restricted to only this example.

Motivating Example

The example we describe here is based loosely on an example described in a Scientific American article(Berners-Lee, Hendler, & Lassila 2001). Suppose Bill and Joan's mother goes to her physician complaining of pain and tingling in her legs and the physician proposes the following sequence of activities:

- A prescription for Relafen, an anti-inflammatory drug;
- An MRI scan and an electromyography, both of these are diagnostic tests to try to determine possible causes for the symptoms;
- A follow-up appointment with the physician to discuss the results of the diagnostic tests.

Bill and Joan need to do the following things for their mother:

- fill the prescription at a pharmacy;
- make appointments to take their mother to the two treatments;
- make an appointment for the doctor's follow-up meeting.

For the three appointment times, there are the following preferences and constraints:

- For the two treatments:
  - Bill and Joan would prefer two appointment times that are close together scheduled at one or two nearby places, so that only one person needs to drive once.
  - Otherwise, they would prefer two appointment times on different days, so that each person needs to drive once.
- The appointment time for doctor's follow up check must be later that the appointment times for the two treatments.
- An appointment time must fit the schedule of the person that will drive to the appointment.

Consider a possible scenario in the near future, where Bill and Joan can use Web services to schedule their mother's appointments. It would be difficult for Bill and Joan to finish their task by consulting the Web services manually, because:

- They may have to try every available pair of close appointment times at any two nearby treatment centers in order to find one that fits their schedules.
- Furthermore, if they first choose an appointment time for one treatment and then find they have to use this same time for the other treatment, then they will have to reschedule the first appointment.

Instead, suppose we use the DAML-S process ontology to encode a description of how to compose Web services for tasks such as the one faced by Bill and Joan. If we have an agent technology which can implement this encoding, then we can perform Bill and Joan's Web services composition task automatically.

Background

DAML-S

In the DAML-S process ontology, each service is modelled as a process. There are three kinds of processes: atomic processes, composite processes and simple processes. An atomic process is undecomposable and represents a directly executable Web service. The execution of an atomic process is the call of the corresponding web accessible program with its input parameters instances. A composite process can be decomposed into other atomic or composite processes and represents a compound Web service. The decomposition of a composite processes is specified through its control constructs. The current set of control constructs defined in the process ontology includes **Sequence**, **Unordered**, **Choice**, **If-Then-Else**, **Iterate**, **Repeat-Until**, **Repeat-While**, **Split** and **Split+Join**. A simple process is used as an element of an abstraction to provide a view of either some atomic process, or a simplified representation of some composite process.

In the process ontology, each process also has related properties, i.e., *(optional) inputs*, (conditional)*outputs*, *preconditions* and (conditional) *effects*. These properties are inputs, outputs, conditions and effects for executing corresponding Web services. The range restriction on each input and output parameter tells the type requirement of the parameter. Here is part of DAML-S definition of an atomic process called PharmacyLocator used in our treatment schedule example as described in Section 2.

```
<daml:Class rdf:ID=”PharmacyLocator”>
  <rdfs:subClassOf
    rdf:resource=”&process;#AtomicProcess”/>
</daml:Class>
<rdf:Property rdf:ID=”LocationPreference”>
  <rdfs:subPropertyOf rdf:resource=”&process;#input”/>
  <rdfs:domain rdf:resource=”PharmacyLocator”/>
  <rdfs:range
    rdf:resource=”&concepts;#LocationPreference”/>
</rdf:Property>
```

The process model of a compound Web service includes the definition of its representing top level process and all other processes that involved in this top level process's decomposition. We can view each Web service as generalization of a task that people want to achieve on the Web. An atomic process models a task that is directly achievable. DAML-S definition of the atomic process articulates all the information of a Web service that can be directly executed to accomplish this task. A composite process models a more complicated task. DAML-S definition of a composite process specifies all the information necessary to select and compose directly executable Web services to finish the task. The goal of automatic web service composition is to develop software to manipulate these DAML-S definitions, find a collection of atomic processes execution thus to achieve the task automatically.

Several metaphors have been used in developing semantic markup of Web services including viewing Web services as primitive and complex actions with preconditions and effects. By using an action metaphor, we can exploit AI technique for planning for automatic service composition. More specifically, we can build an agent that can plan a collection of Web service requests to achieve user's goal of task. Among all planning techniques, HTN (Hierarchical Task Network) planning seems very promising because the concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in DAML-S process ontology. In this paper, we explore a special HTN planning system SHOP2 (Simple Hierarchical Ordered Planner) to show how SHOP2 can be used with DAML-S Web service descriptions to automatically compose Web services.

SHOP2

SHOP2 is a domain-independent HTN planning system. HTN planning is an AI planning methodology that creates plan by task decomposition. This is a process in which the planning system decomposes tasks into smaller and smaller subtasks, until primitive tasks are found that can be performed directly. We can see that the concept of task decomposition in HTN is very similar to the concept of process decomposition in DAML-S. This will make HTN planning system a good candidate for automatic Web services composition task.

One difference between SHOP2 and most other HTN planning systems is that SHOP2 plans for tasks in the same order that they will later be executed. Planning for tasks in the order that those task will be performed makes it possible to know the current state of the world at each step in the planning process, which makes it possible for SHOP2's precondition-evaluation mechanism to incorporate significant inferencing and reasoning power, including the ability to call external programs. This makes SHOP2 ideal as a basis for integrating planning with external information sources as in the Web environment.

In order to do planning in a given planning domain, SHOP2 needs to be given the knowledge about that domain. SHOP2's knowledge based contains operators and methods. Each operator is a description of what needs to be done to accomplish some primitive task, and each method tells how to decompose some compound task into partially ordered subtasks.

Definition 1 (Operator) A SHOP2 operator is an expression of the form ($h(\overrightarrow{v})$ *Pre Del Add*) where

- $h(\overrightarrow{v})$ represents a primitive task with a list of input parameters $\overrightarrow{v}$
- *Pre* represents the operator's preconditions
- *Del* represents the operator's delete list which includes the list of things that will become false after operator's execution.
- *Add* represents the operator's add list which includes the list of things that will become true after operator's execution.

Definition 2 (Method) A SHOP2 method is an expression of the form ($h(\overrightarrow{v})$ *Pre* T) where

- $h(\overrightarrow{v})$ represents a compound task with a list of input parameters $\overrightarrow{v}$
- *Pre* represents the operator's preconditions
- T represents a partially ordered list of subtasks which consist the decomposition of $h(\overrightarrow{v})$.

Additional preconditions and task lists can be appended to the method for SHOP2 to be used in an "if-then-else" manner

$$(h(\overrightarrow{v})\ \mathit{Pre}_1\ T_1\ \mathit{Pre}_2\ T_2 \ldots \mathit{Pre}_n\ T_n)$$

The idea here is that if Pre_1 is true then the method will produce T_1; otherwise if Pre_2 is true then the method will produce T_2 and so forth. This can be viewed as compact way for writing multiple alternative methods for a given task.

In addition to the usual logical atoms, preconditions of SHOP2 methods and operators may also contain calls to external programs and assignments to variables. These are useful for integrating planning with queries to information sources on the Web. For example, the following expression

$$(\text{assign } v \text{ (call } f\ t_1\ t_2 \ldots t_n))$$

will bind the variable symbol v with the result of calling external procedure f with arguments $t_1\ t_2 \ldots t_n$.

Definition 3 (Planning Problem) A planning problem for SHOP2 is a triple (S, T, D), where S is initial state, T is a task list, and D is a domain description. By taking (S, T, D) as input, SHOP2 will return a plan P = ($p_1 p_2 ... p_n$), a sequence of instantiated operators that will achieve T from S in D.

From DAML-S to SHOP2

The execution of an atomic process is a call to the corresponding web accessible program with its input parameters instances. The execution of a composite web service involves executions of a collection of atomic processes. The goal of automatic service composition is to find a collection of atomic processes executions. This sequence will consist a successful execution of a composite process based on its DAML-S definition. In this section, we will show how to encode a composite process composition problem as a SHOP2 planning problem, so SHOP2 can be used with DAML-S Web service descriptions to automatically compose Web services.

Encoding DAML-S Process Models as SHOP2 Domains

In this section, we introduce an algorithm for translating a collection of DAML-S process models K into a SHOP2 domain D. In our translation, we make the following assumption:

Assumption 1 Given a collection of DAML-S process models $K = \{K_1, K_2, \ldots, K_n\}$, we assume:

- *All atomic processes defined in K can either have effects or outputs, but not both.* An atomic process with only output models an information collecting web service. An atomic process with only effect models an world altering web service. If an atomic process models a web service which is both information collecting and world altering, we can always argue that this service can be divided into several web services that are either information collecting or world altering
- *There is no composite process in K with DAML-S's **Split** and **Split+Join** control constructs.* SHOP2 can't handle concurrency right now. Therefore in our translation, we only consider DAML-S process models that has no process with **Split** and **Split+Join** control construct. We intent to address how to extend SHOP2 to handle concurrency in the future work.
- *The effects of all processes in K are not conditional.* SHOP2 doesn't handle conditional effect now. But it is very straightforward to extend SHOP2 to handle conditional effects.

The following algorithm translates a DAML-S definition of an atomic process with only effects into a SHOP2 operator.

TRANSLATE-ATOMIC-PROCESS-EFFECT(Q)

Input: a DAML-S definition Q of an atomic process A with only effects.

Output: a SHOP2 operator O.

Procedure:

1. $\overrightarrow{v}$ = the list of input parameters defined for A in Q
2. *Pre* = conjunct of all preconditions of A, as defined in Q
3. *Add* = collection of all positive effects of A, as defined in Q
4. *Del* = collection of all negative effects of A, as defined in Q
5. Return O = ($A(\overrightarrow{v})$ *Pre Del Add*)

The above algorithm translates each atomic DAML-S definition into a SHOP2 operator that will simulate the effects of a world-altering web service by changing its local state via an operator. The reason why the Web service is not actually executed during the planning process is so SHOP2 can backtrack.

The following algorithm translates a DAML-S definition of an atomic process with only outputs into a SHOP2 operator.

TRANSLATE-ATOMIC-PROCESS-OUTPUT(Q)

Input: a DAML-S definition Q of an atomic process A with only outputs.

Output: a SHOP2 operator O.

Procedure:

1. $\overrightarrow{v}$ = the list of input parameters defined for Aas in Q
2. *Pre* = a conjunct of all the preconditions of A, as defined in Q, plus one more precondition of the form (assign y (call Monitor $A\ \overrightarrow{v}$)), where Monitor is a procedure which will handle SHOP2's call to Web services
3. $Add = y$
4. $Del = \emptyset$
5. Return O = ($A(\overrightarrow{v})$ *Pre Del Add*)

The above algorithm translate each atomic DAML-S definition into a SHOP2 operator that will call the information collecting Web service in its precondition. In this way, the information collecting web service is executed during the planning process.

The following algorithm translates a DAML-S definition of a simple process into a SHOP2 method.

TRANSLATE-SIMPLE-PROCESS(Q)

Input: a DAML-S definition Q of a simple process S.

Output: a SHOP2 method M.

Procedure:

1. $\overrightarrow{v}$ = the list of input parameters defined for Sas in Q
2. *Pre* = conjunct of all preconditions of S as defined in Q
3. T = the atomic process that realizes S or the composite process that collapse into S as defined in Q.
4. Return M = ($S(\overrightarrow{v})$ *Pre* T)

The following algorithm translates a DAML-S definition of a composite process with **Sequence** control construct into a SHOP2 method.

TRANSLATE-Sequence-PROCESS(Q)

Input: a DAML-S definition Q of a composite process C with **Sequence** control construct.

Output: a SHOP2 method M.

Procedure:

1. $\overrightarrow{v}$ = the list of input parameters defined for Cas in Q
2. *Pre* = conjunct of all preconditions of C as defined in Q
3. B = **Sequence** control construct of C as defined in Q
4. $(b_1, \ldots, b_m)$ = the sequence of component processes of B as defined in Q
5. T = ordered task list of $(b_1, \ldots b_m)$
6. Return M = ($C(\overrightarrow{v})$ *Pre* T)

The following algorithm translates a DAML-S definition of a composite process with **If-Then-Else** control construct into a SHOP2 method.

TRANSLATE-If-Then-Else-PROCESS(Q)

Input: a DAML-S definition Q of a composite process C with **If-Then-Else** control construct.

Output: a SHOP2 method M.

Procedure:

1. $\overrightarrow{v}$ = the list of input parameters defined for Cas in Q
2. π_{if} = conditions for **If** as defined in Q
3. P_{re1} = conjunct of all preconditions of C as defined in Q and π_{if}
4. P_{re2} is conjunct of all preconditions of C as defined in Q
5. b_1 = process for **Then** as defined in Q
6. b_2 = process for **Else** as defined in Q
7. Return M = ($C(\overrightarrow{v})$ P_{re1} b_1 P_{re2} b_2)

The following algorithm translates a DAML-S definition of a composite process with **Repeat-While** control construct into SHOP2 methods.

TRANSLATE-Repeat-While-PROCESS(Q)

Input: a DAML-S definition Q of a composite process C with **Repeat-While** control construct.

Output: a collection of SHOP2 methods M.

Procedure:

1. $\overrightarrow{v}$ = the list of input parameters defined for Cas in Q
2. π_{while} = conditions for **While** as defined in Q
3. *Pre* = conjunct of all preconditions of C as defined in Q
4. b_1 = process for **Repeat** as defined in Q
5. M_1 = ($C(\overrightarrow{v})$ *Pre* $C_1(\overrightarrow{v})$)
6. M_2 = ($C_1(\overrightarrow{v})$ π_{While} b_1 $\emptyset$ $\emptyset$)
7. Return $M = \{M_1, M_2\}$

The following algorithm translates a DAML-S definition of a composite process with **Repeat-Until** control construct into SHOP2 methods.

TRANSLATE-Repeat-Util-PROCESS(Q)

Input: a DAML-S definition Q of a composite process C with **Repeat-Until** control construct.

Output: a collection of SHOP2 methods M.

Procedure:

1. $\overrightarrow{v}$ = the list of input parameters defined for Cas in Q
2. π_{Until} = conditions for **Until** as defined in Q
3. *Pre* = conjunct of all preconditions of C as defined in Q

4. b_1 = process for **Repeat** as defined in Q
5. M_1 = ($C(\overrightarrow{v})$ *Pre* $C_1(\overrightarrow{v})$)
6. M_2 = ($C_1(\overrightarrow{v})$ $(not(\pi_{Until}))$ b_1 $\emptyset$ $\emptyset$)
7. Return $M = \{M_1, M_2\}$

The following algorithm translates a DAML-S definition of a composite process with **Choice** control construct into a collection of SHOP2 methods.

TRANSLATE-Choice-PROCESS(Q)

Input: a DAML-S definition Q of a composite process C with **Choice** control construct.

Output: a collection of SHOP2 methods M.

Procedure:

1. $\overrightarrow{v}$ = the list of input parameters defined for Cas in Q
2. *Pre* = conjunct of all preconditions of C as defined in Q
3. B = **Choice** control construct of C as defined in Q
4. $(b_1, \ldots, b_m)$ = the bag of component processes of B as defined in Q
5. for i = 1, …, m
   M_i = ($C(\overrightarrow{v})$ *Pre* b_i)
6. return M =$\{M_1, \ldots, M_m\}$

The following algorithm translates a DAML-S definition of a composite process with **Unordered** control construct into a SHOP2 method.

TRANSLATE-Unordered-PROCESS(Q)

Input: a DAML-S definition Q of a composite process C with **Unordered** control construct.

Output: a SHOP2 method M.

Procedure:

1. $\overrightarrow{v}$ = the list of input parameters defined for Cas in Q
2. *Pre* = conjunct of all preconditions of C as defined in Q
3. B = **Unordered** control construct of C as defined in Q
4. $(b_1, \ldots, b_m)$ = the bag of component processes of B as defined in Q
5. T = unordered task list of $(b_1, \ldots b_m)$
6. Return M = ($C(\overrightarrow{v})$ *Pre* T)

The following algorithm translates a collection of DAML-S process models into a SHOP2 domain.

TRANSLATE-PROCESS-MODEL(K)

Input: a collection of DAML-S process models K.

Output: a SHOP2 domain D.

Procedure:

1. $D = \emptyset$
2. For each atomic process definition Q in K
   If this atomic process has only outputs
   O = TRANSLATE-ATOMIC-PROCESS-OUTPUT(Q)
   If this atomic process has only effects
   O = TRANSLATE-ATOMIC-PROCESS-EFFECT(Q)
   add O to D
3. For each simple process definition Q in K
   M = TRANSLATE-SIMPLE-PROCESS(Q)
   add M to D
4. For each composite process definition Q in K
   If the process has a **Sequence** control construct
   M = TRANSLATE-Sequence-PROCESS(Q)
   If the process has a **If-Then-Else** control construct
   M = TRANSLATE-If-Then-Else-PROCESS(Q)
   If the process has a **Choice** control construct
   M = TRANSLATE-Choice-PROCESS(Q)
   If the process has a **Repeat-While** control construct
   M = TRANSLATE-Repeat-While-PROCESS(Q)
   If the process has a **Repeat-Until** control construct
   M = TRANSLATE-Repeat-Until-PROCESS(Q)
   If the process has a **Unordered** control construct
   M = TRANSLATE-Unordered-PROCESS(Q)
   add M to D
5. Return D

To keep the above pseudo-code simple, we did not explicitly describe how our algorithm handles the composite processes with outputs. In DAML-S, one can specify that an output of a composite process is equal to an output of one of its subprocesses whenever the composite process is instantiated. Also, for a composite process with a **sequence** control construct, one can specify that the output of one subprocess is an input to another subprocesses. SHOP2 does not have the concept of an output, but we handle this problem by assigning a unique number to each instance of SHOP2's methods and operators, and using a predicate $(O_{utput} I_{nstance} V_{alue})$ to indicate that for a method or operator instance $I_{nstance}$, its output named O_{utput} has value V_{alue}.

Encoding DAML-S Web Services Composition Problem as SHOP2 Planning Problem

The formal semantics have been defined for DAML-S service description in action theory based on situation calculus (Narayanan & McIlraith 2002) (Reiter 2001). The following definition of a DAML-S service composition problem follows naturally from this semantics definition.

Definition 3 (DAML-S Service Composition) Let $K = \{K_1, K_2, \ldots, K_m\}$ be a collection of DAML-S process models satisfying assumption in Section 4.1, T be a top level composite process defined in K and $\overrightarrow{c}$ be a list of input parameters instance for T, S_0 be the initial state,and $P = (p_1, p_2, \ldots, p_n)$ be a sequence of atomic processes defined in M with input parameters instance $\overrightarrow{c_1}, \overrightarrow{c_2}, \ldots, \overrightarrow{c_n}$ respectively. Then P is a composition for $T(\overrightarrow{c})$ with respect to K in S_0 iff in action theory, we can prove:

$$\Sigma \vdash (\exists s)(Do(T(\overrightarrow{c}), S_0, s)$$

with $p_1(\overrightarrow{c_1}), p_2(\overrightarrow{c_2}), \ldots, p_n(\overrightarrow{c_n})$ as an instance of s. Here

- Σ is the axiomatization of K and S_0 as defined in action theory.
- $T(\overrightarrow{c})$ is the complex action defined for T as in action theory with input parameters instance $\overrightarrow{c}$
- $p_1(\overrightarrow{c_1}), p_2(\overrightarrow{c_2}), \ldots, p_n(\overrightarrow{c_n})$ are the primitive actions defined for atomic processes $p_1, p_2, \ldots, p_n$ as in action theory with input parameters instances $\overrightarrow{c_1}, \overrightarrow{c_2}, \ldots, \overrightarrow{c_n}$.
- Do is an additional extralogical symbol defined in situation calculus and action theory. Intuitively, $Do(\delta, s, s')$ will macro-expand into a situation calculus formula that says that it is possible to reach situation s' from situation s by executing a sequence of actions specified by δ.

We first state a theorem about a special case.

Theorem 1 Let $K = \{K_1, K_2, \ldots, K_n\}$ be a collection of DAML-S process models satisfying assumption 1 with no atomic processes with outputs, T be a top level composite process defined in K, $\overrightarrow{c}$ be a list of input parameters instance for T, and S_0 be the initial state. Let $P = (p_1, p_2, \ldots, p_n)$ be a sequence of atomic processes defined in K with input parameters instance $\overrightarrow{c_1}, \overrightarrow{c_2}, \ldots, \overrightarrow{c_n}$ respectively. Let D = **TRANSLATE-PROCESS-MODEL(K)**. Then P is a composition for $T(\overrightarrow{c})$ with respect to K in S_0 iff P is a plan for planning problem (S_0, T($\overrightarrow{c}$), D).

Outline of Proof. We can prove that a service composition problem and its corresponding SHOP2 planning problem map to the same theorem proving problem in action theory.

We now generalize the above theorem to remove the restriction of no atomic processes with the outputs.

As shown in the **TRANSLATE-ATOMIC-PROCESS-OUTPUT** procedure earlier, the precondition for the operator translated from an atomic process with output, SHOP2 will call a Monitor procedure to handle SHOP2's call to external information collecting Web services. This Monitor will monitor the current state of SHOP2, so that information can only be added into the current state if it has not been changed by the planner. We assume here that information will not be changed by other agents during SHOP2 planning time and we will address this problem in the future work.

Soundness and completeness proofs of classical planners assume that the preconditions can be evaluated relative to the current state. However, if a precondition involves call to the external program, this is no longer the case. We have to guarantee that all programs calls to be

- executable (with all parameters grounded)
- terminable (with finite computation)

to ensure that the precondition is finitely evaluable.

For every composite process P defined in DAML-S, we know that an input parameter of a subprocess A of P is either bound to an input parameter of P or an output parameter of other subprocess B within P which must be executed before A. Therefore, when we call an information collecting service, all of its parameters must be grounded. If we can assume that all web service invocations are terminable, then we can establish the soundness and completeness proof of SHOP2.

Theorem 2 Let $K = \{K_1, K_2, \ldots, K_n\}$ be a collection of DAML-S process models satisfying assumption in Section 4.1, T be a top level composite process defined in K and $\overrightarrow{c}$ be a list of input parameters instance for T, S_0 be the initial state. K_a = K - {atomic processes with outputs in K} and P be a sequence of atomic processes defined in K with input parameters instance $\overrightarrow{c_1}, \overrightarrow{c_2}, \ldots, \overrightarrow{c_n}$ respectively. Let D = **TRANSLATE-PROCESS-MODEL(K)**. D_a = **TRANSLATE-PROCESS-MODEL(K_a)**. If every execution of the information collecting Web services defined in K is guaranteed to terminate, then P is a plan for planning problem ($\emptyset$, $T(\overrightarrow{c})$, D) iff P is a plan for planning problem (S_0, $T(\overrightarrow{c})$, D_a).

Outline of Proof. Because call to the information collecting services are always terminable, information is always available whenever needed. Therefore, SHOP2 will have the same planning process for two problems.

Implementation

Our implementation includes:

- A DAML-S to SHOP2 translator which translates a collection of DAML-S process definitions into a SHOP2 domain.
- An interface which lets users specify the request for a service.
- A monitor which handles SHOP2's calls to external information collecting Web services during planning. This monitor system will cache the responses of the information collecting Web services to avoid invoking a Web service with same parameters more than once during planning. This will save the network communication times and improve planning efficiency. We assume that the cached information will not be changed by other agents during planning; we will generalize this in our future work
- A SHOP2 to DAML-S plan converter, which will convert a plan to DAML-S format which can be directly executed by a DAML-S executor.

To test the effectiveness of our approach, we have run SHOP2 on several instances of the problem described in Section 2. These problem instances varied from cases where it was easy to schedule satisfactory appointments to a case in which no nearby treatment centers had treatment times slot were close together, so that Bill and Joan would both have to drive Mom for treatments on separate days. In all of these cases, SHOP2 was easily able to find the best possible solution.

Related Work

Our technology here use the SHOP2 planning system which won an award for distinguished performance in the 2002 International Planning Competition.

We use DAML-S for semantic markup of Web services. The current version 0.7 of DAML-S is still an incomplete draft version, and many researchers in DAML program are working actively on it. Our work will evolve with the work on DAML-S.

Another approach for automatic Web service composition is the work of McIlraith and others in Stanford University on adapting Golog for programming the semantic Web (McIlraith & Son 2002). Golog is a logical programming language built on top of situation calculus. Golog forms a natural formalism for automatic service composition tasks by providing some extra logical constructs for assembling primitive actions. However, we suspect that the approach will not be as efficient as an HTN planner.

Conclusion

In this paper, we have proposed a way to do automatic Web service composition by exploiting AI techniques for planning. We believe that SHOP2 provides a natural formalism for this task. We have described an approach for translating process models of Web services into sets of SHOP2 methods and operators, so that SHOP2 can be used with DAML-S Web service descriptions to automatically compose Web services. Our future work will include the following:

- We need to enhance SHOP2 to handle those control constructs related to concurrency in DAML-S.
- In our current approach, we assume that the information we get during the planning will not be changed. For example, if we locate a book in a bookstore, and it is in stock, then we assume the book is available when we actually try to buy the book—but in a changing world, this will not always be true, we intend to extend our approach to take into account the ways in which information may change during planning.

Acknowledgments

This work was supported in part by Air Force Research Laboratory grant F30602-00-2-0505.

References

Ankolekar, A.; Burstein, M.; Hobbs, J.; Lassila, O.; Martin, D.; McDermott, D.; McIlraith, S.; Narayanan, S.; Paolucci, M.; Payne, T.; and Sycara, K. 2002. DAML-S: Web service description for the semantic web. In *Proceedings of the First International Semantic Web Conference*.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The semantic web. *Scientific American*.

Dean, M.; Connolly, D.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; and Stein, L. A. 2002. Web ontology language (OWL) reference version 1.0. W3C Working Draft 12 November 2002, http://www.w3.org/TR/2002/WD-owl-ref-20021112/.

Horrocks, I.; van Harmelen, F.; Patel-Schneider, P.; Berners-Lee, T.; Brickley, D.; Connoly, D.; Dean, M.; Decker, S.; Fensel, D.; Hayes, P.; Heflin, J.; Hendler, J.; Lassila, O.; McGuinness, D.; and Stein, L. A. 2001. DAML+OIL. http://www.daml.org/2001/03/daml+oil-index.html.

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*.

Narayanan, S., and McIlraith, S. 2002. Simulation, verification and automated composition of web services. In *Proceedings of the Eleventh International World Wide Web Conference*.

Nau, D.; Munoz-Avila, H.; Cao, Y.; Lotem, A.; and Mitchell, S. 2001. Total-order planning with partially ordered subtasks. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*.

Reiter, R. 2001. *Knowledge In Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.