Automating DAML-S Web Services Composition Using SHOP2

Dan Wu[1], Bijan Parsia[2], Evren Sirin[1], James Hendler[1], and Dana Nau[1]

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

Abstract. The DAML-S Process Model is designed to support the application of AI planning techniques to the automated composition of Web services. SHOP2 is an Hierarchical Task Network (HTN) planner well-suited for working with the Process Model. We have proven the correspondence between the semantics of SHOP2 and the situation calculus semantics of the Process Model. We have also implemented a system which soundly and completely plans over sets of DAML-S descriptions using a SHOP2 planner, and then executes the resulting plans over the Web. We discuss the challenges and difficulties of using SHOP2 in the information-rich and human-oriented context of Web services.

1 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) [1] or DAML+OIL[2], 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 [4], 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 Web services related activities including service discovery, composition, execution, and monitering.

For our work, we are motivated by issues related to automated Web services composition. One part of DAML-S, namely its process ontology, provides a standard language for describing the composition of Web services. Several metaphors have been used in developing this semantic markup of Web services including

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 195–210, 2003.
© Springer-Verlag Berlin Heidelberg 2003

viewing Web services as primitive and complex actions with preconditions and effects.

Given a representation of services as actions, we can exploit AI planning techniques for automatic service composition by treating service composition as a planning problem. Ideally, given a user's objective and a set of Web services, a planner would find a collection of Web services requests that achieves the objective. We believe that HTN planning is especially promising for this purpose, 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 how to use the SHOP2 HTN planning system[5,6] to do automatic composition of DAML-S Web services.

This paper is organized in the following manner. In Section 2, we describe a sample scenario 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 our approach. But our work is designed to be domain-independent and is not restricted to only this example.

2 Motivating Example

The example we describe here is based loosely on a scenario described in the Scientific American article about the Semantic Web [7]. 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, and that person drives only 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.

Assume that there are the requisite Web services for finding appointment times and making appointments at the relevant clinics, Bill and Joan could use those services to schedule their mother's appointments. It would be difficult for Bill and Joan to finish their task with an optimal plan 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 find an execution path based on this predefined task decompositions, then we can perform Bill and Joan's Web services composition task automatically.

3 Background

3.1 DAML-S

In the DAML-S process ontology, services are modelled as processes. There are three kinds of processes: *atomic* processes, *composite* processes and *simple* processes. In DAML-S, an *atomic* process is a model of a "single step" (from the point of view of the client) Web service that is directly executed to accomplish some task. Executing an *atomic* process consists of calling the corresponding Web-accessible program with its input parameters bound to particular values. A *composite* process represents a compound Web service, i.e., it can be decomposed into other *atomic* or *composite* processes. The decomposition of a *composite* process is specified through its control constructs. The set of control constructs includes: **Sequence**, **Unordered**, **Choice**, **If-Then-Else**, **Iterate**, **Repeat-Until**, **Repeat-While**, **Split** and **Split+Join**. A simple process is an abstraction of an atomic or composite process (or of a possibly empty set of these). It is not considered to be directly executable, but provides an abstract view of an action. Like atomic processes, simple processes are, themselves, single-step, but unlike atomic processes, it's possible to peek at the internal structure of a simple process (if available) or to replace the simple process with an expansion of it.

In the process ontology, each process has several properties, including, *(optional) inputs*, *preconditions*, (conditional)*outputs* and (conditional) *effects*. *preconditions* specify things that must be true of the world in order for an agent to execute a service. *effects* characterize the physical side-effects that execution of a Web-service has on the world. *inputs* and *outputs* correspond to knowledge preconditions and effects. That is, necessary states of our knowledge base before execution and modifications to our knowledge base as a result of the execution. Note that not all services have physical side-effects, in particular, services that are strictly information-providing do not. Here is part of the DAML-S definition of an atomic process called PharmacyLocator used in our treatment scheduling example:

```
<daml:Class rdf:ID="PharmacyLocator">
  <rdfs:subClassOf rdf:resource=
    "http://www.daml.org/services/daml-s/0.7/Process.daml
    #AtomicProcess"/>
</daml:Class>

<rdf:Property rdf:ID="LocationPreference">
  <rdfs:subPropertyOf rdf:resource=
    "http://www.daml.org/services/daml-s/0.7/Process.daml
    #input"/>
  <rdfs:domain rdf:resource="#PharmacyLocator"/>
  <rdfs:range rdf:resource= "http://www.mindswap.org/services/
    shop2/concepts.daml#LocationPreference"/>
  </rdf:Property>
```

The process model of a compound Web service includes the designation of the top-level composite process corresponding to that service plus a decomposition of that composite process into a structured collection of (perhaps further decomposed) subprocesses.[1] Web services composition is sometimes thought of as the process of generating a (potentially) complexly structured composite process description which is subsequently executed. On this model, composite processes are the *output* of composition. In this paper, we take composite processes as *input* to a planner, that is, as descriptions of *how* to compose a sequence of single step actions. Thus, for us, the goal of automated Web services composition is find a collection of atomic processes instances which form an execution path for some top-level composite process.

3.2 SHOP2

SHOP2 is a domain-independent HTN planning system, which won one of the top four awards out of the 14 planners that competed in the 2002 International

[1] Here, we assume that a compound Web service always has a complete decomposition bottoming out in atomic processes. Such a composite process is *exectuable*.

Planning Competition. 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.

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, including Web based ones.

In order to do planning in a given planning domain, SHOP2 needs to be given the knowledge about that domain. A SHOP2 knowledge base consists of operators and methods (plus, various non-action related facts and axioms). 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 method's preconditions
- *T* represents a partially ordered list of subtasks which consist the decomposition of $h(\overrightarrow{v})$.

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.

4 From DAML-S to SHOP2

The execution of an atomic process is a call to the corresponding web accessible program with its input parameters instantiated.[2] The execution of a composite process ultimately consists in the execution of a collection of specific atomic processes. Instead of directly executing the composite process as a program on a DAML-S interpreter, we can treat the composite process as specification for how to compose a sequence of atomic process executions. 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 services descriptions to automatically generate a composition of Web services calls.

4.1 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.* According to the situation calculus based semantics of DAML-S[8], *outputs* characterize knowledge effects of executing Web services and *effects* characterize physical effects for executing Web services. An atomic process with only outputs models an strictly information-providing Web service. And an atomic process with only effects models an world-altering Web service. In general, we don't want to actually affect the world during planning. However, we do want to gather certain information from information-providing Web services, which entails executing them at plan time. To enable information gathering from Web services at planning time, we require that the atomic processes to be either exclusively information-providing or exclusively world-altering.
- *There is no composite process in K with DAML-S's* ***Split*** *and* ***Split+Join*** *control constructs.* SHOP2 currently doesn't handle concurrency. Therefore in our translation, we only consider DAML-S process models that have no composite process using **Split** and **Split+Join** control construct. We also assume only a non-concurrent interpretation of **Unordered** (as permitted

[2] Here, we assume that before the execution of an atomic process the preconditions for executing the atomic process have been satisfied.

by DAML-S). We intend to address how to extend SHOP2 to handle concurrency in the future work.

We encode a collection of DAML-S process definitions K into a SHOP2 domain D as follows:

- For each atomic process with effects in K, we encode it as a SHOP2 operator that simulates the effects of the world-altering Web service.
- For each atomic process with output in K, we encode it as a SHOP2 operator whose precondition include a call to the information-providing Web service.
- For each simple or composite process in K, we encode it as one or more SHOP2 methods. These methods will tell how to decompose an HTN task that represents the simple or composite process.

The following algorithm shows how to translate a DAML-S definition of an atomic process with only effects into a SHOP2 operator.[3]

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. Such Web services will never be executed at planning time, for obvious reasons.

The following algorithm shows how to translate 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 $O = (A(\overrightarrow{v})\ Pre\ Del\ Add)$

[3] Conditional effects can be easily encoded into SHOP2 operators. Here, for simplicity, we assume that effects (and outputs) are not conditional.

The above algorithm translates each atomic DAML-S definition into a SHOP2 operator that will call the information-providing Web service in its precondition. In this way, the information-providing web service is executed during the planning process. The operator for these atomic processes are entirely "bookkeeping", thus none of these operators will appear in the final plan.

The following algorithm shows how to translate a DAML-S definition of a simple process into SHOP2 method(s).

TRANSLATE-SIMPLE-PROCESS(Q)
Input: a DAML-S definition Q of a simple process S.
Output: a collection of SHOP2 methods 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. $(b_1, \ldots, b_m)$ = the list of atomic and composite processes that realizes or collapse into S as defined in Q.
4. for i = 1, ..., m
   – $M_i = (S(\overrightarrow{v})\ Pre\ b_i)$
5. return $M = \{M_1, \ldots, M_m\}$

The following algorithm shows how to 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 algorithms for translating composite processes with other control constructs such as **Unordered**, **Choice**, **If-Then-Else**, **Iterate**, **Repeat-Until** and **Repeat-While** control constructs are similar to the one for **Sequence**, and so we omit the details here.

To keep the above pseudocode simple, we did not specify the recursive translations within a composite process. E.g., What happens if we have a **Sequence** of **If-Then-Else** or further nestings? Our way for handling this problem is to treat each control construct within a composite process as a composite process. For above example, in our translation, we will have a SHOP2 method for the composite process with **Sequence** construct construct and a method for each nested **If-Then-Else** control construct.

Also we did not explicitly describe how our algorithm handles 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 a SHOP2 domain's methods and operators. There is a, predicate $(O_{utput} I_{nstance} V_{alue})$, which indicates for each method or operator instance $I_{nstance}$ the value V_{alue} of the particular output O_{utput}.

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

A formal semantics has been given for DAML-S service description in terms of an action theory based on the situation calculus [8] [9]. The following definition of a DAML-S service composition problem follows naturally from this semantics.

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 1 (from 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 K 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 but also no atomic processes with outputs, T be a top level composite process defined in K, $\overrightarrow{c}$ be a list of input parameters

instances 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 have proven 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-providing 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)
- repeatable (with same result for the same call)

to ensure that the planner is sound and complete.

Given the dataflow model of DAML-S, no atomic process is, in fact, executed unless all its necessary inputs are bound, we know that when we call an information-providing service, all of its parameters must be grounded. If we can assume that information provided by all Web services will not change during SHOP2 planning time and all web services 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-providing 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 calls to the information-providing services are al-

ways terminable, information is always available whenever needed. Therefore, SHOP2 will have the same planning process for two problems.

5 Implementation

To realize our agent technology, we started with an implementation of a DAML-S to SHOP2 translator. This translator is a Java program that reads in a collection of DAML-S process definitions files and outputs a SHOP2 domain file. As shown in the translation algorithm in section 4.1, when planning for any problem in this domain, SHOP2 will actually call the information-providing Web services to collect information while maintaining the ability of backtrack by merely simulating the effect of world-altering Web services. The output of SHOP2 is a sequence of world-altering Web services calls that can be subsequently executed.

We built a monitor which handles SHOP2's calls to external information-providing Web services during planning. We wrote a DAML-S Web services executor which communicates with SOAP based Web services described by DAML-S groundings to WSDL descriptions of those Web services. Upon SHOP2's request, the monitor will call this DAML-S Web services executor to execute the corresponding Web service. Since the information-providing services are always defined as atomic processes, the service is executed by invoking the WSDL service in the grounding. The monitor also caches the responses of the information-providing 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, and establishes the repeatability condition required for proving SHOP2's soundness and completeness. Also information can only be added into the current state if it has not been changed by the planner. We assume that the cached information will not be changed by other agents during planning and we will generalize this in our future work.

We also built a SHOP2 to DAML-S plan converter, which will convert a the plan produced by SHOP2 to DAML-S format which can be directly executed by the DAML-S executor.

We ran our scenario from section 2 on this system. In doing so:

- Our system communicated with real Web services. Unfortunately, the current Web services available on the Web have only WSDL descriptions without any semantic markup. Therefore, we created DAML-S markup for the WSDL decriptions of these online services and for some services it was also necessary to create the WSDL descriptions since the service was only offered via filling a form on a web page such as CVS Online Pharmacy Store. It was not possible to use real services for some of the services either because none was available as Web services, e.g. a doctor's agent providing the patient's prescription, or it was infeasible to use a real Web service for the demo, e.g. making an appointment with a doctor each time the program is executed. For these services, we implemented Web services to simulate these functionalities.
- We built Web services that allow the access to user's personal information sources. For example, it is necessary to learn the user's schedule to be able to

generate a plan for the example task in our demo. It is possible to get this information from the sources available on the user's machine such as a Personal Information Manager like Microsoft's Outlook. We have implemented "local" SOAP based services that will retrieve this kind of information. WSDL and DAML-S descriptions are also generated for these local services so that they can be composed and executed in the same way as other remotely available services.
Finally, some information gathering services were implemented as direct Java calls from SHOP2 over a Java/SHOP2 bridge. For example, we have a service which asks the user for acceptable distances to the treatment center by popping up a window on the user's client to accept input. Changing the data entered at this point will possibly yield a different plan to be generated allowing the planner produce custom plans depending on personal preferences.

– We also encoded a description of how to compose Web services for tasks such as the one faced by Bill and Joan in section 2 in DAML-S. The description is given as a DAML-S composite process that is composed of several other composite processes that are defined as sequence, choice or unordered processes. This DAML-S description constitutes the top level composite process described in Section 4.2 and is translated into a SHOP2 domain for planning. We encode most of the constraints mentioned in section 2 as Web service preconditions. Right now, there is no standard process modelling language for specifying Web service preconditions. Therefore, we directly encode the Web services preconditions in SHOP2 format.

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 time slots that 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. Fig. 1 shows a snapshot of some information collected during SHOP2 planning process and a plan produced by SHOP2.

6 Related Work

McIlarith and Son [10] proposed an approach to building agent technology based on the notion of generic procedures and customizing user constraints. They argue that an augmented version of the logic programming language Golog provides a natural formalism for programming Web services. These contributions are realized in their development of the ConGolog interpreter which communicates with Web services via the Open Agent Architecture (OAA) but the service and procedure ontologies are written in first-order logic. Our system more directly supports the use of existing Web services by being able to ground directly in the existing WSDL services rather than creating a separate execution system for semantically described services. Also, we suspect that their approach will not be as efficient as an HTN planner.

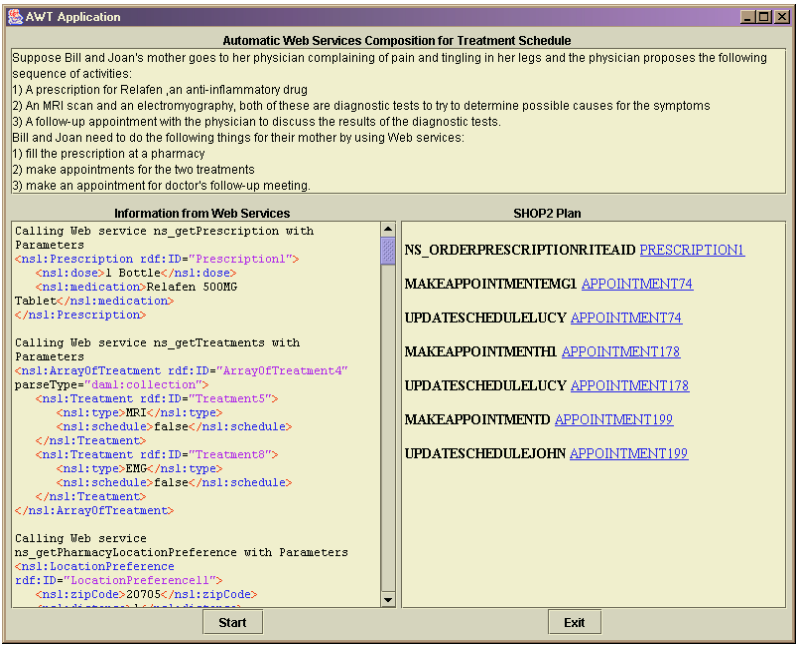

Fig. 1. Example Running Result

Matskin and Mao [11] applies software synthesis and composition methods to Web services composition. Their work is based on similarities between Web service composition and component-based system development in software engineering. They use DAML-S for service descriptions and adopt Structural Synthesis Program (SSP) method for automated service composition. Service composition is based on the input-output information of services components and requires little domain knowledge. This approach treats each service as an atomic entity without inspecting the internal process model and therefore lacks the ability to reason about different decompositions in a composite process.

RETSINA is an open multi-agent system that provides infrastructure for different types of deliberative, goal directed agents. RETSINA system includes a planner [13] based on the HTN planning paradigm. The RETSINA planner also extends HTN planning by adding interleaving of planning and execution which basically allows the acions execute before a plan is completely formed, similar to our approach. Paolucci et al. [13] also describes using RETSINA planner in the context of creating autonomous Web services that can automatically interact with each other. However, authors do not give details about how the HTN planning is employed in the system. It is not clear whether DAML-S Process Model was used or planning domain was given a priori to the planner agent. For this reason, we cannot make a comparison with our approach.

7 Conclusion

In this paper, we have defined a translation from DAML-S process models to the SHOP2 domains, and from DAML-S composition tasks to SHOP2 planning problems. We have described our implemented system which performs this translation, uses an extended SHOP2 implementation to plan with and over the translated domain, and then executes the resulting plans. In the process of defining the translation and building the system, we observed that:

- In our current approach, the planner always executes output producing actions as it plans. While this is fine for many situations, it may not always be appropriate. For example, the execution of some Web services may take a very long time. It would be better if the planner could continue to plan while waiting for this information.
- In our paper, we assume that all effects are physical. In complex situations, there may be other changes, such as in the mental states of the agents involved, that are not modelled. We will explore this problem in our future work.
- There is a fundamental difference between exclusively information providing and possibly world-altering atomic processes. We typically want to execute information providing atomic processes at various points in the planning process, while we never want to execute world-altering ones. Contrariwise, at composition execution time, the primary interest is in the execution of world-altering processes. Indeed, in our system we do not include any information providing processes in compositions. Furthermore, currently we do not permit world-altering processes to be information providing, at least in the sense that they must have no outputs. This simplification made the system fairly easy to implement without substantial modification of SHOP2. Mapping information-gathering processes to so-called "book-keeping" operators may be somewhat unaesthetic. It seems possible to encode them directly as external code calls in preconditions for the processes that receive the information as input, or more generally as part of the axiomatic inferential structure of the planner's state, though this would probably require some substantial dataflow analysis.

  Information providing (whether exclusively so, or not) is and is likely to be a significant fraction of the available and salient Web services. Many Web contexts seem to be *information rich* but *action poor*. In such environments, we'd want to reconsider the strict partition of services into exclusively information providing and output free. For example, world-altering services with outputs might supply information needed to decide subsequent courses of action. Clearly, such a service shouldn't be executed at planning time, which suggests that we will need to investigate generating conditional plans by SHOP2 style HTN planning.

  Conditional plans will also help eliminate the constraint on information change during planning. Currently, both for theoretical and practical reasons, we only execute an information providing process once during planning for any given input, and subsequently retrieve a cached result. Given

SHOPs speed, this is not that unreasonable a restriction for many cases, but conditional plans would permit planning for various contingencies.

These considerations raise a host of issues regarding plan time vs. composition execution time execution of information providing processes, including those of deciding which such processes to execute only during planning, only during plan execution, and during both. Furthermore, in complex, long running planning sessions, it might make sense to refresh the monitors cache for certain services at intervals. Presumably, DAML-S descriptions will be enriched to help support the requisite analysis. We intend to explore these issues in future work.

- Compared the complexities raised above, composite processes raise no additional or special problems — encoding them as SHOP2 methods seems correct and straightforward, modulo the need to extend SHOP2 to handle concurrency.
- Simple processes are the odd duck of the lot. Though various members of the DAML-S coalition have suggested, in conversation, that simple processes were intended to support HTN planning, we found them niether necessary, nor convenient, nor useful. In part, their lack of a clear semantics, particularly with regard to the relationship of their inputs, outpus, preconditions, and effects to those of their corresponding atomic or composite processes. Furthermore, while the language of the technical overview[3] suggests that a given simple process can be a view of one atomic process or one composite process, but not both, neither the langauge nor the ontology actually require this restriction. We speculated that this would make simple processes useful for specifying a range of alternative composition paths, but it wasn't clear that this was really more convenient (for our purposes) than using the **Choice** control construct.

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

References

1. Dean, M., Connolly, D., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P. F. and Stein, L. A.: Web Ontology Language (OWL) Reference Version 1.0. Recent Trends and Developments. W3C Working Draft 12 November 2002, http://www.w3.org/TR/2002/WD-owl-ref-20021112/
2. 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.: DAML+OIL. http://www.daml.org/2001/03/daml+oil-index.html (2001)
3. The DAML Services Coalition: DAML-S:Semantic Markup for Web Services. Technical Overview. http://www.daml.org/services/daml-s/0.7/daml-s.html (2002)
4. The DAML Services Coalition(Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D. , McDermott, D., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K.):DAML-S: Web Service Description for the Semantic Web. Proceedings of the First International Semantic Web Conference (2002)

5. Nau, D., Munoz-Avila, H., Cao, Y., Lotem, A., Mitchell, S.:Total-Order Planning with Partially Ordered Subtasks. Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (2001)
6. Nau, D., Au, T., Ilghami,O., Kuter, U., Murdock, J. W., Wu, D., Yaman, F.: SHOP2: An HTN Planning Environment. To appear in Journal of Artificial Intelligence Research (2003)
7. Berners-Lee, T., Hendler, J., Lassila, O.:The Semantic Web. Scientific American (2001)
8. Narayanan, S., McIlraith, S.: Simulation, Verification and Automated Composition of Web Services. Proceedings of the Eleventh International World Wide Web Conference, Honolulu, Hawaii (2002)
9. Reiter, R.: Knowledge In Action: Logical Foundations for Specifying and Implementing Dynamical Systems, The MIT Press (2001)
10. McIlraith, S., Son, T.: Adapting Golog for Composition of Semantic Web Services. Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning, Toulouse, France, (2002)
11. Matskin, M., Rao, J.: Value-Added Web Services Composition Using Automatic Program Synthesis. Web Services, E-Business, and the Semantic Web, CAiSE 2002 International Workshop, WES 2002, Toronto, Canada, (2002)
12. Paolucci, M., Shehory, O., Sycara,K.: Interleaving planning and execution in a multiagent team planning environment. Electronic Transactions of Artificial Intelligence, (2001)
13. Paolucci, M., Sycara,K., Kawamura, T.: Delivering semantic web services. Technical Report CMU-RI-TR-02-28, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2002