http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (1): 48-54, 2010
ISSN 1812-5638

Composing Semantic Web Services with PDDL

Bo Yang and Zheng Qin
Institute of Computer Science and Technology, Xi'an Jiaotong University,
710049, Xi'an, Shaanxi, People's Republic of China

Abstract: The aim of this study is to solve the problem of composing semantic Web services with AI planner. Automatic semantic Web service composition is a key technique in the research of Web service. The idea of composing semantic Web services using AI planner has been presented in several studies. In our opinion, since the different application domains and the constraint of precondition that AI planners have, none of the existing planners can deal with the problem of semantic Web services composition perfectly. In this study, we present an algorithm for translating semantic Web service composition problem to AI planning problem described in PDDL. Based on this algorithm, according to the situation of the problem, we can use the most suitable AI planner to compose Web services. One of our goals is to support the claim that AI planner can be used in the domain of semantic Web service composition. The other goal of ours is that the description of semantic Web services can be translated into a description written in PDDL. There are also several key procedures of the translating algorithm in this study.

Key words: Web service composition, AI planner, PDDL

INTRODUCTION

Semantic Web services, like conventional Web services, are the server end of a client-server system for machine-to-machine interaction via the world wide Web. Nowadays, there are a great amount of Web services provided by companies and organizations only implement their core business. However, often time it happens, there is no single service capable of performing a task, but there are combinations of existing services that could. In this case, the technique of efficiently and effectively composing the services on the Web, with little or no direct human intervention, is an important focus in the research of the Web services community (Rao and Su, 2005).

Rao and Su (2005) summarized the five conceptually separate phases of automatic composition of Web services: (1) presentation of single service, (2) translation of the language, (3) generation of composition process model, (4) evaluation of composite service and (5) execution of composite service. We will focus on the second and third phase in this study.

The work of these two phases needs two critical techniques: a machine-readable description language and a composition approach. There are several machine-readable languages for Web service composition. But not all of these languages can be used in automatic Web service composition, mostly due to the absence of semantic representation of the Web services available on the Internet. Against this problem, the semantic Web community and others proposed the Web Ontology Language of Web Services (OWL-S) and the Web Services Modeling Ontology (WSMO).

There are a lot of approaches of Web service composition, such as: automata (Fu *et al*., 2004; Diaz *et al*., 2005), Petri net (Hamadi and Boualem, 2003), AI planner and so on. Among these approaches, AI planning is one of the most suitable and simplest techniques of automated Web service composition, because there are a lot of similar features between the Web services and AI actions. Various AI planning approaches have been considered during previous efforts related to Web service composition. It includes HTN (Sirin *et al*., 2004, 2005), classic AI planning (Rao *et al*., 2006), Golog (McIlraith and Tran, 2002; Narayanan and Sheila, 2002) etc. Even though great success has been get in this area due to the hard work of the researchers, there are two shortcomings in these approaches. First, none of the existing AI planners can perfectly solve all of the planning problem. Some planner may feel helpless in dealing with some particular problem and the performance of some planners drops rapidly because of the tremendous information. None of the existing planners can be better than the other ones in the aspects of universality and performance. Second, high coupling between AI planner and the other applications, such as

Corresponding Author: Bo Yang, Institute of Computer Science and Technology, Xi'an Jiaotong University, 710049, Xi'an, Shaanxi, People's Republic of China

Web services discovery application, will restrict the applicability of the approach of Web service composition.

Against these two problems, we suggest to translate the problem of Web service composition to a problem of AI planning described by PDDL. Thus, we can select the most suitable AI planner for further service synthesis. PDDL is a widely accepted language used in AI planning community to make the expressing of planning problems and domains uniform. Moreover, OWL-S has been strongly influenced by PDDL language; mapping from one representation to another is straightforward.

BACKGROUND

OWL-S: The OWL-service language (OWL-S), is an ontology, within the OWL-based framework of the semantic Web, for describing semantic Web services (Martin *et al*., 2004). The OWL-S is designed to enable users and software agents to automatically discover, invoke, compose and monitor Web resources offering services, under specified constraints. The knowledge about a Web services can be divided into three types: service profile, service grounding and service model. What we need in the work of Web service composition, is the service model. In the OWL-S service model, operations are modeled as processes. A process is not a program to be executed. It is a specification of the ways a client may interact with a service. There are three kinds of processes:

- Atomic processes
- Composite processes
- Simple processes

Atomic processes correspond to the actions that a service can perform by engaging it in a single interaction; composite processes correspond to actions that require multi-step protocols and/or multiple server actions; finally, simple processes provide an abstraction mechanism to provide multiple views of the same process. Composite processes are decomposable into other (non-composite or composite) processes; their decomposition can be specified by using control constructs. The set of control constructs includes: Sequence, Split, Split+Join, Any-Order, Choice, If-Then-Else, Iterate, Repeat-While and Repeat-Until.

In the process ontology, each process has several properties, including, (optional) input, (conditional) outputs, preconditions and (conditional) effects (IOPE). Preconditions presents logical conditions that should be satisfied prior to the service being invoked. Effects are the physical side-effects on the world influenced by the execution of a Web service. Inputs and outputs specify the data transformation produced by the process. That is to say, inputs and outputs correspond to the preconditions and effects of knowledge, respectively.

PDDL: The Planning Domain Definition Language (PDDL) proposed by Ghallab *et al.* (1998) is a standard encoding language for planning tasks. The PDDL is inspired by well-known STRIPS and it is an action-centered language as well as STRIPS. It uses precondition and post-condition to describe the applicability and effects of actions. which is wildly accepted in AI community despite some argument in the some features of PDDL, the language has been wildly accepted in AI community, since it standardize the domain description and the problem description in planning research. Then we can compare the AI planner and share planning resources. The introduction of PDDL has facilitated the scientific development of planning.

Planning tasks specified in PDDL are separated into two files:

- A domain file
- A problem file

The domain file is composed of the predicates which are used to describe the knowledge in the world and the actions which can change the world states. The problem file includes objects in the world, initial state and goal specification.

AN APPROACH OF TRANSLATING OWL-S TO PDDL

A planning problem can be described as a five tuple $<S, S_0, G, A, \Gamma>$ (Rao and Su, 2005) where, S is the set of all possible states of the world, $S_0 \subset S$ denotes the initial state of the world, $G \subset S$ denotes the goal state of the world that the planning system attempts to reach, A is the set of actions the planner can perform in attempting to change one state to another state in the world and the translation relation $\Gamma \subset S \times A \times S$ defines the precondition and effects for the execution of each action. Tackling Web service composition problem via AI planning, we can treat S_0 and G as the initial states and the goal states of Web services requests. A is the set of available services. Γ is the state change function of each service. There are two kinds of actions in PDDL:

- Primitive actions
- Composite actions

An action with no expansion is called a primitive action, or just a primitive (Ghallab *et al*., 1998).

Definition 1: A primitive action in PDDL is an expression of the form (a($v^{\rightarrow}$) Pre Eff) where:

- a is an action functor, $v^{\rightarrow}$ is the list of input parameters of a
- Pre is the action's preconditions
- Eff is the action's effect in which the things will change after the action's execution properly

Definition 2: A composite action in PDDL is an expression of the form (a($v^{\rightarrow}$) Pre Eff Exp) where:

- a is an action functor, $v^{\rightarrow}$ is the list of input parameters of a
- Pre is the action's preconditions
- Eff is the action's effect in which the things will change after the action's execution properly
- Exp specifies all the ways the action may be carried out in terms of (presumably simpler) actions

Figure 1 shows the process of composing Web services using PDDL. The relevant OWL-S services description set is the semantic Web services set acquired by semantic Web services discovery. The process of composing Web services using AI planner can be done in three steps. At first, we must convert the Web services description of domain and problem written in OWL-S to the planning problems described in PDDL. Then, input the result generated by the converting process into AI planner. At last, convert the actions sequence that AI planner got in the precede step into a composite Web service. In the whole process, the step converting OWL-S to PDDL is the difficult and important one. The third step converting an actions sequence to a composite Web Services is the inverse step of this step and is easier than it. In this section, we will put our focus on the work of converting OWL-S to PDDL.

There are three types of process in OWL-S:

- Atomic processes
- Composite processes
- Simple processes

We need convert each type of processes into PDDL. The nature of three types of processes can be summarized as follows. The execution of an atomic process is a call to the corresponding Web accessible program with its instantiated parameters. The execution of a composite process is the execution of some atomic processes in some construct, ultimately. A simple process is only an abstraction of an atomic process or a composite process. It is not considered to be executable, but it is an abstract view of an action.

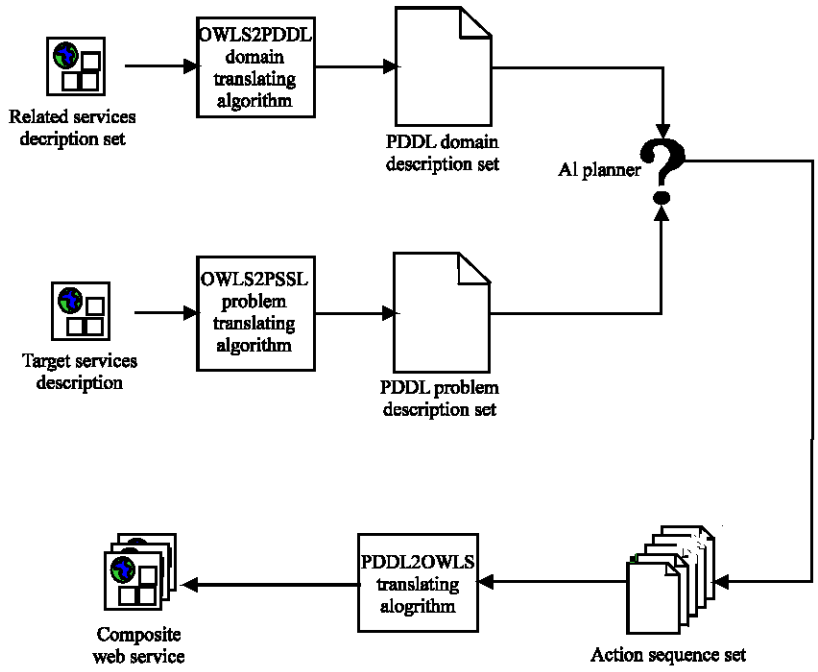

Fig. 1: The algorithm flow of composite Web services based PDDL

Encoding OWL-S process models as PDDL domains: Here, based on the study of Sirin *et al.* (2004), we introduces an algorithm for translating a collection of OWL-S process models K into a PDDL domain D. Before our translation, we must make an assumption. Each atomic process in K has either effects or outputs but not both. According to (Reiter, 2001) the outputs of a Web service characterize the knowledge effects of executing Web services, while the effects of a Web service characterize the physical effects of executing the Web service. The process having only outputs can be seen as a purely information-providing Web service. And the process having only effects can be seen as a purely world-altering Web service. In generally, we do not want the world state changes during the planning. However, we want to gather enough information from information-providing Web services. To get essential information during planning time, we need the atomic processes be either exclusively information-providing or exclusively world-altering.

There is no composite actions in PDDL corresponding with the OWL-S's Repeat-While and Repeat-Until control constructs, because the behave of iteration is not encouraged in planning domain. Therefore, in our translation, we only consider OWL-S process models that have no composite process using Repeat-While and Repeat-Until. Since, PDDL does not provide an Any-Order and Split+Join construct, we will not consider these construct in our algorithm. But both of constructs can be expressed by other constructs in PDDL with some labor, we will not discuss it in details.

Encoding OWL-S process models: The algorithm of encoding OWL-S process models in PDDL is accomplished by handling three categories of process, separately. In respect to atomic process, we divided it into information-providing Web service and world-altering Web Service, then we deal with them, separately. For each simple or composite process, we encode it as a composite action according to its control constructs.

The study of encoding a collection of OWL-S process definitions K into a PDDL domain D in pseudo code is shown in algorithm 1:

Algorithm 1: Convert process PDDL(K)

Input: OWL-S process model set K
Output: PDDL action sequence set AS
AS = 0;
For each atomic process k ∈ K **do**
    If k with only effects: **then**
    A = Translate Atomic Process Out put (k);
End
If k with only outputs: **then**
    A = Translate Atomic Process Effect(k);
End

Algorithm 1: Continued

    Add A to AS;
End
For each simple process k ∈ K **do**
    A = Translate Simple Process(k);
    Add A to AS;
End
For each composite process k ∈ K **do**
    invoke the different functions according to the type of control construct;
End
Return AS;

Translate atomic process with only effects: The algorithm 2 shows how to translate an OWL-S definition of an atomic process with only effects into a PDDL primitive action.

Algorithm 2: Translate atomic process effect (K)

/* translate an OWL-S atomic process with only effects to a primitive action */
Input: K is the formalize definition of an atomic process P with only effects
Output: a PDDL primitive action A
$\vec{v}$ = input parameter list of P, as defined in K;
Pre = conjunct of all preconditions of P, as defined in K;
Eff = collection of all effects of P, as defined in K;
Return A = (P($\vec{v}$) Pre Eff);

The above algorithm translate an atomic process with only effects to an action in PDDL that will simulate the effect of a world-altering Web service by changing its local state via an action. Such Web services will never be executed at planning time, for it will change the world state which is not allowed at planning time.

Translate atomic process with only outputs: The algorithm 3 shows how to translate an OWL-S definition of an atomic process with only outputs into a PDDL primitive action.

Algorithm 3: Translate atomic process output (K)

/* translate an OWL-S atomic process with only outputs to a primitive action */
Input: K is the formalize definition of an atomic process P with only outputs
Output: a PDDL primitive action A
$\vec{v}$ = input parameter list of P, as defined in K;
Pre = (and (conjunct of all preconditions of P, as defined in K) (executable P))
Where executable is a function to judge whether the Web Services is executable;
Eff = (exe P)
where exe is a function of executing the Web Service;
Return A = (P($\vec{v}$) Pre Eff);

The above algorithm translates an atomic process with only outputs to an action in PDDL. Because the Web service to be converted is an information-providing service, in order to gather enough information during planning process, we will invoke the service in the effect component of the action.

Translate simple process: The algorithm 4 shows how to translate an OWL-S definition of a simple process into a PDDL composite action.

Algorithm 4: Translate simple process (K)

/* translate an OWL-S simple process to a composite action */
Input: K is the formalize definition of a simple process P
Output: a PDDL composite action A
$v^{\rightarrow}$ = input parameter list of P, as defined in K;
Pre = (conjunct of all preconditions of P, as defined in K);
Eff = $\varnothing$;
{(b_1, ..., b_m)} is the list of atomic and composite processes that realizes or collapse into P as defined in K;
Exp = (choice b_1', ..., b_m');
where choice express a expansion choice of this action and what different between b_i' and b_i is b_i' = (in-context b_i: precondition (executable b_i))
Return A = (P($v^{\rightarrow}$) Pre Eff Exp);

The above algorithm translate a simple process to a composite action in PDDL. In this algorithm, instead of handling the simple process directly, we replace the simple process with an expansion of it which is an executable one.

Translate sequence process: The algorithm 5 shows how to translate an OWL-S definition of a composite process with Sequence control construct into a PDDL composite action.

Algorithm 5: Translate sequence process (K)

/* translate an OWL-S composite process with Sequence control construct to a composite action */
Input: K is the formalize definition of a composite process P with Sequence control construct
Output: a PDDL composite action A
$v^{\rightarrow}$ = input parameter list of P, as defined in K;
Pre = (conjunct of all preconditions of P, as defined in K);
Eff = $\varnothing$
Exp = (series b_1'...b_m') = the list of atomic and composite processes that realizes *P* as defined in *K*,
Where series is a control construct which means the following actions will be executed in order and what different between b_i' and b_1 is b_i' = (in-context b_i: precondition (executable b_i))
Return A = (P($v^{\rightarrow}$) Pre Eff Exp);

The above algorithm translates a composite process with Sequence control construct to a composite action in PDDL with a series control construct in its expansion component.

Translate If-Then-Else process: The algorithm 6 shows how to translate an OWL-S definition of a composite process with If-Then-Else control construct into a PDDL composite action.

Algorithm 6: Translate If-Then-Else process (K)

/* translate an OWL-S composite process with If-Then-Else control construct to a composite action */
Input: K is the formalize definition of a composite process P with If-Then-Else control construct
Output: a PDDL composite action A

Algorithm 6: Continued

$v^{\rightarrow}$ = input parameter list of P, as defined in K;
Pre = (conjunct of all preconditions of P, as defined in K);
π_{if} = condition for If as defined in K;
Eff = $\varnothing$
Exp = (series (in-context p_1: precondition π_{if}) (in-context p_2: precondition (not $_{if}$)))
where (in context action: preconditon pre) in PDDL is defined as action will be executed, when pre is true;
Return A = (P($v^{\rightarrow}$) pre Eff Exp);

The above algorithm translates a composite process with If-Then-Else control construct to a composite action in PDDL. Because there are no similar control construct in PDDL with the If-Then-Else in OWLS, we use the in-context and series construct to implement the same function.

Translate choice process: The algorithm 7 shows how to translate an OWL-S definition of a composite process with Choice control construct into a PDDL composite action.

Algorithm 7: Translate choice process (K)

/* translate an OWL-S composite process with Choice control construct to a
composite action */
Input: K is the formalize definition of a composite process P with Choice control construct
Output: a PDDL composite action A
$v^{\rightarrow}$ = input parameter list of P, as defined in K;
Pre = (conjunct of all preconditions of P, as defined in K);
Eff = $\varnothing$;
Exp = (choice b_1,...,b_m)
where (b_1,...,b_m) is the bag of component processes as defined in P and (b_1,...,b_m) in PDDL is defined as any action can be chosen from b_1,...,b_m during the execution.;
Return A = (P($v^{\rightarrow}$) Pre Eff Exp);

The above algorithm translates a composite process with Choice control construct to a composite action in PDDL.

Translate split process: The algorithm 8 shows how to translate an OWL-S definition of a composite process with split control construct into a PDDL composite action.

Algorithm 8: Translate split process (K)

/* translate an OWL-S composite process with Split control construct to a Composite action */
Input: K is the formalize definition of a composite process P with Split control construct
Output: a PDDL composite action A
$v^{\rightarrow}$ = input parameter list of P, as defined in K;
Pre = (conjunct of all preconditions of P, as defined in K);
Eff = $\varnothing$;
Exp = (parallel b_1,...b_m)
where (b_1,...b_m) are the component processes which are executed in parallel as defined in K and (parallel b_1,...b_m) in PDDL is defined as the actions b_1,...b_m can be concurrently executed.;
Return A = (P($v^{\rightarrow}$) Pre Eff Exp);

```
<query xmlns= 'http://foo.com/discover#'>
<inputs>
<input>http://foo.com/#Book</input>
</inputs>
<outputs>
<output>http://foo.com/#Price</output>
</outputs>
</query>
```

Fig. 2: A service request described by OWL-S

```
(define (problem getbookprice )
(:domain foo)
(:init
book ?b)
(:goal (price ?b))
)
```

Fig. 3: The corresponding problem description by PDDL

The above algorithm translates a composite process with Split control construct to a composite action in PDDL.

Encoding OWL-S target service description to PDDL problem description: A service request is a desired Service profile and it can include the desired location and inputs/outputs. In the service profile, what attracts us is the description about desired world state that service wants. It is absolutely that these world-states can be described by first-order propositions. Further, the PDDL problem definition is just the conjunction of some first-order propositions. So, we can translate one to another, easily. Figure 2 and 3 give an OWL-S code and a PDDL code about a problem querying the price of a book, respectively.

The process converting a plan to a composite Web service is the inverse process of converting a composite Web service to action expansions and the forth one is simpler than the latter one since OWL-S has more powerful ability to express than PDDL.

CONCLUSION

In this study, we presented an algorithm translation from OWL-S process models to the PDDL domain for further automatic Web service composition. Based on this study, the problem of automatic Web service composition can be accomplished by a wide range of AI planners. The present study, is based on the close-world assumption, but actually the Web services is an open-world assumption. There may be other changes, such as in the mental states of the agents involved and the world state changed during the planning time, these are not considered in this study. We will explore this problem in our future study. In Web services, the way of getting planning information is not only acquired from the description in target service, but also can be acquired from information-providing service. In this study, we does not consider the information provided by the Web services which includes both information providing and world altering, because it can not be executed at planning time due to the obvious reason. We will investigate the service with both functions to gather more information which will be great help in planning time. The reality that PDDL does not support iteration construct does restrict its expression, while some planners have, such as HTN. We will explore PDDL to support this function, either by modifying existing constructs or by developing new construct.

ACKNOWLEDGMENTS

This study was supported by the national Natural Science Foundation of China under grant No. 60673024 and also supported by the 11th-Five-Year Preliminary Research Project of China under grant No. 282660008 and No. 570851613.

REFERENCES

Diaz, G., P. Juan-Jos, C. Mara-Emilia, V. Valentn and C. Fernando, 2005. Automatic Translation of WS-CDL Choreographies to Timed Automata. In: Lecture Notes in Computer Science, Diaz, G., P. Juan-Jos, C. Mara-Emilia, V. Valentn and C. Fernando (Eds.). Vol. 3670, Springer, Versailles, France, pp: 230-242.

Fu, X., B. Tevfik and S. Jianwen, 2004. Analysis of interacting BPEL Web services. Proceedings of the 13th International Conference on World Wide Web, May 17-20, New York, USA., pp: 621-630.

Ghallab, M., A. Howe, C. Knoblock, D. Mcdermott and A. Ram *et al*., 1998. PDDL-the planning domain definition language. Technical Report, http://www.informatik.uni-ulm.de/ki/Edu/Vorlesungen/GdKI/WS0203/pddl.pdf.

Hamadi, R. and B. Boualem, 2003. A petri net-based model for Web service composition. Proceedings of the 14th Austral asian Database Conference, Feb. 2003, Australian Computer Society, Adelaide, South Australia, pp: 191-200.

Martin, D., M. Burstein, J. Hobbs, O. Lassila and D. Mcdermott *et al*., 2004. OWL-S: Semantic markup for Web services. Technical Report, UNSPECIFIED, Member Submission, W3C. Intelligence, Agents, Multimedia. http://eprints.ecs.soton.ac.uk/id/eprint/12687.

McIlraith, S.A. and S.C. Tran, 2002. Adapting golog for composition of semantic Web services. Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning, April 22-25, Morgan Kaufmann, Toulouse, France, pp: 482-496.

Narayanan, S. and M.A. Sheila, 2002. Simulation, verification and automated composition of Web services. Proceedings of the 11th international conference on World Wide Web, Nov. 9-16, Honolulu, Hawaii, USA., pp: 77-88.

Rao, J. and X. Su, 2005. A Survey of Automated Web Service Composition Methods. In: Lecture Notes in Computer Science, Rao, J. and S. Xiaomeng (Eds.). Vol. 3387, Springer, Berlin, pp: 43-54.

Rao, J., D. Dimitar, H. Paul and S.M. Norman, 2006. A mixed initiative approach to semantic Web service discovery and composition: SAP's guided procedures framework. Proceedings of IEEE International Conference on Web Services, Sept. 18-22, IEEE Computer Society, Chicago, Illinois, USA., pp: 401-410.

Reiter, R., 2001. Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. 1st Edn., The MIT Press, USA., ISBN: 978-0-262-18218-8.

Sirin, E., P. Bijan, W. Dan, H.A. James and N.S. Dana, 2004. HTN planning for Web service composition using SHOP2. J. Web Semantics, 1: 377-396.

Sirin, E., P. Bijan and H. James, 2005. Template-based composition of semantic Web services. Proceedings of AAAI Fall Symposium on Agents and the Semantic Web, AAAIFSAS'2005, MINDSWAP Research Group, pp: 85-92.