ECAI 2008
M. Ghallab et al. (Eds.)
IOS Press, 2008

doi:10.3233/978-1-58603-891-5-735

Automated Web Services Composition Using Extended Representation of Planning Domain

Mohamad El Falou[1] and **Maroua Bouzid**[1] and **Abdel-Illah Mouaddib** [1] and **Thierry Vidal** [2]

1 INTRODUCTION

WS are distributed software components that can be exposed and invoked over the Internet using standard protocols. They communicate with their clients and with other WS by sending XML based messages over the Internet.

Artificial Intelligence planning techniques can help solving the composition of WS problem. In fact, services can be modelled as actions and the business process as planning to connect the WS.

The main contribution of this paper is the extension of the model of actions to handle the creation or elimination of objects as effects of actions. This contribution allows us to answer new and more expressive requests, called *implicit* requests, in which goals may contain objects that have been generated by the plan.

2 Related Works

The work on web service composition in the university of Trento presented in [5] translates WS into a state transition. After translating WS, the system constructs a parallel product $\sum_{\|}$ which combines the n STS, this parallel product allows the n services to evolve concurrently. They use the Model Based Planner MBP [1] based on model checking techniques [6]. The drawback of this approach is that we must recalculate $\sum_{\|}$ whenever we add or remove a service from the domain.

In [3] an approach called *GOLOG* based on the situation calculus is presented. *GOLOG* composes web services by applying logical inference techniques on pre-defined plan templates.

Finally, in [7] the authors define a translation from DAML-S process models to the SHOP2 domains, and from DAML-S composition tasks to the SHOP2 planning problem. SHOP2 is a well-suited planner for working with the Process Model in an Hierarchical Task Network (HTN). HTN planning builds plans through task decomposition.

All the approaches cited above suppose that the domain objects are static. In other words there is no way to eliminate nor to create objects. Furthermore, all defined requirements for the composite Web Services are defined as explicit queries.

3 Motivating Example

Let us consider a set of WS which are intended to deal with files, images and tracks as follows:

1. WS_1 translates file languages. It has two services: $fr2en$ ($en2ar$) translates files from French (English) to English (Arabic).
2. WS_2 transforms text file formats. It has two services: $latex2doc$ ($doc2pdf$) transforms files from $latex$ (doc) to doc (pdf) format.
3. WS_3 merges files. It has two services: $mergepdf$ ($mergedoc$) merges two pdf (doc) files into a third one.

As an example, let us suppose that we have two files: the first is a doc format written in English, the second is in latex format written in French and we want to obtain a file which contains the content of the two files translated to Arabic. The existing approaches dedicated to WS composition cannot express or deal with this kind of problem. To overcome this limitation, we propose an approach where the specification language of the domain consists in an extension of the specification language PDDL [4] and the WS composition mechanism is based on two planning mechanisms which are Tree-search and GraphPlan.

4 Formal Framework

Our formal framework is based on extended Planning-Graph techniques [2] allowing the creation and elimination of objects when executing services (actions).

Contrary to classical approaches where a state is defined as a set of predicates, a state in our domain is defined by a set of objects, properties and relations between these objects, and we extend the definition of actions to allow the generation and elimination of objects in the environment, the assignment of new predicates to objects and the definition of new relations between them.

4.1 Preliminaries and Definitions

The **domain D** $= (C, P)$ is defined by a set of WS $C = (WS_1, WS_2, ..., WS_n)$ that we call a community of Web Services, and a set of predicate types P= $\{p_1, p_2, ..., p_n\}$ to specify the possible properties of objects and relations between them.

A **state** q=(V,P) of the plan execution is defined by a set of objects V and their types, and a set of predicates P specifying the properties of these objects and the relationship between them.

In section 3, the initial state is specified as : q_0=({(F1: **file**),(F2: **file**)},{(**doc** F1),(**en** F1),(**latex** F2),(**fr** F2)} , where F1,F2 are objects (files) and **file** is a type and **doc, en, latex**, and **fr** are properties.

A **Web Service** WS_i is defined by $WS_i = (T_i, A_i, S_i)$ which are respectively : type, attributes and services of WS.a

A **service** in WS_i is defined by $S_i^k = (Pin_i^k, Pout_i^k, Pinout_i^k, Prec_i^k, Effects_i^k)$ which are respectively : input, output and input-output objects, preconditions and effects of service execution.

The service **mergepdf** of WS_3 is defined as follows:

[1] University of Caen, France, email: melfalou, bouzid, mouaddib@info.unicaen.fr
[2] IRISA - INRIA Rennes, France, email: thierry.vidal@irisa.fr

- Pin_3= { (F1: file),(F2: file) }.
- $Pout_3$= {(F: file) }.
- $Pinout_3$= { }.
- $Prec_3$={(**pdf** F1),(**pdf** F2)}.
- $Effect_3^-$={(**pdf** F1),(**pdf** F2)}.
- $Effect_3^+$={(**pdf** F), (**merge** F F1 F2)}.

A **plan** is defined by a sequence of sets of services where every set is called a partial plan. More formally $\Pi =< \pi_1, \pi_2, ..., \pi_n >$ is a plan such that $\forall i \in [1..n], \pi_i = (s_{i1}, ..., s_{in})$ is a partial plan of independent services, and each s_{ik} is instantiated with real objects in the domain.

One plan solution for the problem introduced in section 3 is $\Pi =< \pi_1, \pi_2, \pi_3, \pi_4, \pi_5 >$ where :

- π_1=(fr2en [F1], en2ar [F2]).
- π_2=(en2ar [F1], doc2pdf [F2]).
- π_3=(latex2doc [F1])
- π_4=(doc2pdf [F1]).
- π_5=([#F0]=mergepdf[F1, F2]).

A **request** $R = (\mathbf{D}, q_0, g)$ is defined for a domain **D** of WS by the initial state q_0 and the goal state g. The initial and final states are defined by a set of objects and a set of associated predicates (V, P).

In the previous example, q_0=[{(F_1:file),(F_2:file)},{(doc F_1),(en F_1),(latex F_2),(fr F_2)}] and g=[{($\#F_0$:file)},{(pdf $\#F_0$),(ar $\#F_0$),(merge F_1 F_2 $\#F_0$)}].

The aim of using the symbol # before the name of the object is to state that it is a generated object (in the output set of the executed service), and any other object having the same type beginning with # can replace it in the domain.

5 Planning Algorithm

We have implemented two algorithms to build the solution of our problem. The first one is based on the classical **Tree-search** algorithm and the second one is based on the **Graph plan** method.

The basic idea behind the Tree-search algorithm is to apply from the initial state all executable services. By doing this (expand a state) we obtain a set of new states S, if the goal is in S, a solution is found. If not, based on the strategy of Tree-search we select one of the unexpanded states. If all states are expanded we get a failure. By using this algorithm, we obtain a sequential plan of services. After that, we transform this plan into a sequential set of partial plans (set of independent services) $\Pi =< \pi_1, \pi_2, ..., \pi_n >$.

5.1 Graph plan algorithm

The GraphPlan algorithm performs a procedure close to iterative deepening, discovering a new part of the search space at each iteration. It iteratively expands the planning graph by one level, then it searches backward from the last level of this graph for a solution. The first expansion, however, proceeds to a level P_i in which all the goal propositions are included and no pairs of them are mutex, and the set of services executed for reaching g are not mutex; and so on until reaching P_0 (then the plan is found), or until reaching failure ($P_i = P_{i+1}$ and no plan is found).

6 Implementation and Results

By implementing the Tree-search and the Graph Plan algorithms, we prove that our new approach of composition WS under implicit request is feasible. In our implementation we use a part of the *PDDL* language, and extend it to fit our model.

Table 1. Results of Tree-search algorithm

		strategy			
		depth		width	
Problem	nbr of objects	node nbr	plan size	node nbr	plan size
P1	1	6	4	9	4
P2	1	4	0	4	0
P3	2	8	7	232	6
P4	2	9	8	2585	8
P5	3	13	12	> 4200	$\oslash$
P6	3	14	13	583	7
P7	4	18	17	> 2900	$\oslash$

$\oslash$ solution not found

We tested our algorithms on 7 examples that contain many objects and many types of variables (files, tracks and images). P_2 is the problem given in section 3.

In table 1 we give the number of initial objects of the different problems, the number of expanded nodes and the plan size by using depth and width strategies. We have 16 available services in the domain (illustrated in section 3). From these results we can observe that the depth strategy is very effective and in few seconds we obtain a plan by expanding a few number of nodes.

By using the *Graphplan* algorithm, we obtain solutions for simple problems, but not for complex problems that contain a high number of objects. By applying the extended techniques of Graph-plan, we get a combinatorial explosion. The combinatorial explosion is due to the execution of services that create new objects in each level.

7 Conclusion And Perspective

In this paper, we give an extended view of the composition of Web Service problem by modelling the problem as a planning problem. In our work we propose an extended model of service to answer to composition problems that require the creation and elimination of objects as effects of the execution of a service. We also overcome the limitations of other approaches, by giving a dynamic and distributed definition of our domain. This allows us to add, remove and/or replace services without recalculating some other part of the domain. Finally, our model overcomes the limitation of pre-defined plans by defining the implicit request only through an initial and a goal states.

REFERENCES

[1] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, 'Mbp: a model based planner', *the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information, Seattle, August 2001.*, (2001).

[2] Paolo Traverso Malik ghallab, Dana Nau, *Automated Planning , theory and practice*, MORGAN KAUFMANN PUBLISHERS, Jun 2005.

[3] S. McIlraith and T. Son. Adapting golog for composition of semantic web services, 2002.

[4] C.Knoblock D.McDermott A.Ram M.Veloso D.Weld D.Willkins M.Ghallab, A.Howe, 'Pddl — the planning domain definition language', (1998).

[5] M. Pistore, P. Bertoli, F. Barbon, D. Shaparau, and P. Traverso, 'Planning and monitoring web service composition', *ICAPS 2004*.

[6] Marco Pistore and Paolo Traverso, 'Planning as model checking for extended goals in non-deterministic domains', 479–486, (2001).

[7] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. Automating daml-s web services composition using shop2, 1998.