Automatic Web Service Composition with Abstraction and Refinement

Hyunyoung Kil Wonhong Nam Dongwon Lee
The Pennsylvania State University, University Park, PA 16802, USA
{hykil, wnam, dongwon}@psu.edu

ABSTRACT

The behavioral description based *Web Service Composition (WSC)* problem aims at the automatic construction of a coordinator web service that controls a set of web services to reach a goal state. However, solving the WSC problem exactly with a realistic model is *doubly-exponential* in the number of variables in web service descriptions. In this paper, we propose a novel efficient approximation-based algorithm using automatic *abstraction* and *refinement* to dramatically reduce the number of variables needed to solve the problem.

Categories and Subject Descriptors
H.3.5 [Information Storage and Retrieval]: Online Information Services—Web-based services
General Terms: Algorithms, Experimentation
Keywords: Service Composition, Abstraction, Refinement

1. INTRODUCTION

Web services are software systems designed to support machine to machine interoperation over the Web. The *Web Service Composition (WSC)* problem in this paper is, given a set W of (behavioral descriptions of) web services and a reachability goal G, to automatically synthesize a *coordinator web service* c that controls W to satisfy G.

Despite abundant researches on the WSC problem, only a few [4, 3, 2] employ realistic models (i.e., with *incomplete information*) for WSC on behavioral descriptions. Since this problem is known to be doubly-exponential in the number of variables in web service descriptions [2], studying efficient approximation approaches is required. Therefore, we present a novel approach to solve this computationally hard problem, using *abstraction* and *refinement*. To the best of our knowledge, it is the first attempt to apply automatic abstraction technique to the WSC problem. We describe a preliminary implementation, and demonstrate that our automatic abstract-refinement technique can solve efficiently 3 sets of realistic problems—8 instances.

2. WEB SERVICE COMPOSITION

Suppose that clients want to reserve both of a flight ticket and a hotel room for a particular destination and a period, and there exist only an airline reservation (AR) web service and a hotel reservation (HR) web service separately. Clearly, we want to combine these web services rather than implementing a new one. One way to combine them is to construct a coordinator web service (Travel agency system) which communicates with each web service to book up both a flight ticket and a hotel room. Fig 1 presents this example.

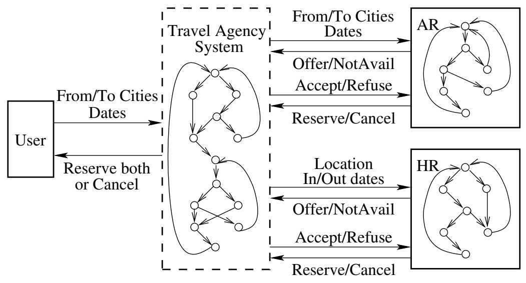

Figure 1: Travel agency system

A *web service* w is a 5-tuple $(X, X^I, X^O, \mathit{Init}, T)$ where:

- X is a finite set of *variables* that w controls. A state s of w is a valuation for every variable in X. We denote a set of all the states as S.
- X^I is a finite set of *input variables* which w reads from its environment; $X \cap X^I = \emptyset$, and every variable $x \in X \cup X^I$ has a finite domain. A state *in* for inputs is a valuation for every variable in X^I.
- $X^O \subseteq X$ is a finite set of *output variables.*
- $\mathit{Init}(X)$ is an *initial predicate* over X.
- $T(X, X^I, X')$ is a *transition predicate* over $X \cup X^I \cup X'$. For a set X of variables, we denote the set of primed variables of X as $X' = \{x' \mid x \in X\}$, which represents a set of variables encoding the successor states.

For a state s over X, let $s[Y]$ where $Y \subseteq X$ denote the valuation over Y obtained by restricting s to Y. Note that the process model for any web service described in WS-BPEL or OWL-S can be easily transformed into our representation above. In the WSC problem, given a set W of available web services, every web service in W communicates only with their coordinator but not with each other. Based on this assumption, a set $W = \{w_1, \cdots, w_n\}$ of web services can be represented by a tuple $(X, X^I, X^O, \mathit{Init}, T)$ by a general cartesian product of each web service.

Since a *coordinator* is also a web service, it is a tuple $c = (X_c, X_c^I, X_c^O, \mathit{Init}_c, T_c)$. Although T_c can define a nondeterministic transition relation, in this problem, we want only a *deterministic* transition relation for c. Given a set $W = (X, X^I, X^O, \mathit{Init}, T)$ of web services and a coordinator $c = (X_c, X_c^I, X_c^O, \mathit{Init}_c, T_c)$ where $X^I = X_c^O$ and $X^O = X_c^I$, we can define an *execution tree*, denoted by $W||c$, which represents the composition of W and c as follows:

- Each node (s, s_c) in $W||c$ is in $S \times S_c$. (s, s_c) is the root node iff $\mathit{Init}(s) = \mathit{true}$ and $\mathit{Init}_c(s_c) = \mathit{true}$.
- Each node (s, s_c) has a set of child nodes, $\{(s', s_c') \mid T(s, \mathit{in}, s')=\mathit{true}, \mathit{in}=s_c[X^I], T_c(s_c, \mathit{in}_c, s_c')=\mathit{true}, \mathit{in}_c= s'[X^O]\}$. Intuitively, the web services W, by receiving the input *in* from the current state s_c of the coordinator, collectively proceeds from s to the next state s',

and then the coordinator, by receiving the input in_c from the new state s' of W, proceeds from s_c to s_c'.

A *goal* G is a set of states to reach, and specified as a predicate. Given a set W of web services, a coordinator c, and a goal G, we define $W||c \models G$ if every path $(s_0, s_{c0}) \cdots (s_n, s_{cn})$ in the execution tree $W||c$ reaches a goal state eventually (i.e., $s_n \in G$). Formally, given a set W of web services and a goal G, the WSC problem in this paper is to construct a coordinator c such that $W||c \models G$.

Theorem 1. *The WSC problem with no internal variable (i.e., $X = X^O$) is EXP-hard, and the WSC problem for a general case is 2-EXP-hard [2].* ■

3. ABSTRACTION AND REFINEMENT

Theorem 1 implies that more efforts to devise efficient approximation solutions to the WSC problem be needed. In this paper, therefore, we propose an *approximation*-based algorithm. Algorithm 1 presents a high-level description of our algorithm. First, we abstract a given web service set W into W^{Abs}, which has less variables than W but includes all the behaviors of W. Given a set of web services $W(X, X^I, X^O, Init, T)$ and a set $Y \subseteq X$ of variables, the abstraction of W with respect to Y is $W_Y(X_Y, X_Y^I, X_Y^O, Init_Y, T_Y)$ where:

- $X_Y = Y$, $X_Y^I = X^I$, and $X_Y^O = X^O$.
- For every $s_Y \in S_Y$, $Init_Y(s_Y) = true$ iff $\exists s \in S.\ (Init(s) = true) \wedge (s_Y = s[Y])$.
- For every $s_Y, s'_Y \in S_Y$, $T_Y(s_Y, in, s'_Y) = true$ iff $\exists s, s' \in S.\ (T(s, in, s') = true) \wedge (s_Y = s[Y]) \wedge (s'_Y = s'[Y])$.

Since the abstraction W_Y contains all the behaviors of W, W_Y satisfies the following property.

Theorem 2. *Given a set W of web services and a goal G, if a coordinator web service c satisfies $W^{Abs}||c \models G$ where W^{Abs} is an abstraction of W, then c satisfies $W||c \models G$.* ■

In Algorithm 1, the procedure *Abstraction* constructs abstract web services W_Y for the given variable set Y. Since our first abstraction is performed with $Y = X^I \cup X^O$ (lines 1–2), W_Y includes no internal variable and in this case we can exploit the procedure *WSC_NoInternalVars* that is more efficient (i.e., EXP-hard). *WSC_NoInternalVars* (line 3) and *WSC_General* (line 8) try to construct a coordinator c for a given web services W_Y wihtout/with internal variables, respectively. If we find a coordinator c such that $W_Y||c \models G$, then c also satisfies $W||c \models G$ by Theorem 2. Otherwise, we refine W_Y by adding variables, and try to find c for the new abstraction (lines 5–9). For selecting variables to be added, we construct a *variable dependency graph* that is a directed graph, in which each vertex is a variable and a directed edge from x to y exists iff the value of y depends on x (e.g., $y := x$). We then cluster variables according to the number of hops to variables appearing in a given goal predicate G. In every iteration of our algorithm, we add a set of variables that is closest to variables in G (i.e., 1-hop, 2-hop, $\cdots$). We repeat the abstration/refinement step until we identify a coordinator c satisfying $W_Y||c \models G$ or the variable set used for abstraction equals to the original variable set. The later case implies no solution for the given problem. Although from the second loop, we should employ a general algorithm *WSC_General* that is 2-EXP, once we identify a coordinator using small abstract web services, searching space is shrunken (double-)exponentially in the number of variables that we save. In Algorithm 1, the procedures, *WSC_NoInternalVars* and *WSC_General*, can be implemented by using *automated planning on complete information* and *incomplete information*, respectively.

Algorithm 1: Abs/Ref Web Service Composition

```
   Input  : A set $W$ of web services and a goal $G$.
   Output: A coordinate web service $c$.
 1 $Y := X^I \cup X^O$;
 2 $W_Y := Abstraction(W, Y)$;
 3 if $((c := WSC\_NoInternalVars(W_Y, G)) \neq null)$ then
 4     return $c$;
 5 while $((NewVars := SelectNewVariables(W, G)) \neq null)$ do
 6     $Y := Y \cup NewVars$;
 7     $W_Y := Abstraction(W, Y)$;
 8     if $((c := WSC\_General(W_Y, G)) \neq null)$ then
 9         return $c$;
10 return $null$;
```

Table 1: Experiment result

Problem	T var	I/O var	Basic	Abs/Ref	S var
TAS1	36	18	**0.1**	**0.1**	8
TAS2	70	24	32.5	**25.6**	12
TAS3	111	25	>7200.0	**2589.7**	12
P&S1	43	17	**3.1**	3.2	15
P&S2	48	18	56.0	**43.1**	15
P&S3	59	20	2003.2	**1244.9**	20
VOS1	57	17	64.8	**18.2**	18
VOS2	67	17	2940.8	**520.8**	23

To demonstrate that our tool efficiently synthesizes coordinator web services, we compared a basic algorithm [4] (i.e., without abstarction/refinement) and our method with 3 sets of realistic examples (8 instances); Travel agency system (TAS) explained in Section 2, Producer and shipper (P&S) [4], and Virtual online shop (VOS) [1]. All experiments were performed on a PC using a 2.4GHz Pentium processor, 2GB memory and a Linux OS. Table 1 presents the number of total variables (T var) and input/output variables (I/O var) in boolean. It also shows the total execution time in seconds for a basic algorithm (Basic) [4] and our method (Abs/Ref), and the number of boolean variables that we saved (S var). Our experiment shows that our technique outperforms the basic algorithm in 6 instances (1 tie).

4. CONCLUSION

We have proposed an automatic technique for WSC problems based on abstraction and refinement. Our preliminary experiment shows promising results. As future work, we plan to investigate other abstraction methods and various refinement techniques in order to early converge the conclusion. Finally, it is worth pointing out that our approach can readily be adopted for other WSC methods such as knowledge-level composition [3].

5. REFERENCES

[1] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *Proc. of ICWS*, pages 63–71, 2006.

[2] H. Kil, W. Nam, and D. Lee. Computational complexity of web service composition based on behavioral descriptions. In *Proc. of ICTAI*, pages 359–363, 2008.

[3] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *Proc. of IJCAI*, pages 1252–1259, 2005.

[4] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *Proc. of ISWC*, pages 380–394, 2004.