2010 IEEE Fourth International Conference on Semantic Computing

Using Virtual Services to Bridge the Semantic Gap

Jicheng Fu[a] jfu@uco.edu
Wei Hao[b] haow1@nku.edu
I-Ling Yen[c] ilyen@utdallas.edu
Farokh Bastani[c] bastani@utdallas.edu

[a]Computer Science Department, University of Central Oklahoma,
[b]Computer Science Department, Northern Kentucky University
[c]Computer Science Department, The University of Texas at Dallas

Abstract

In cloud computing, data, software, and hardware are wrapped as services, which are made available on demand. Given a demand, the associated services need to interact with each other to fulfill the task. As a result, service reuse and composition are inevitable. The current prevailing Web service composition paradigms follow the bottom-up manner, i.e., the concrete services are physically constructed first. Then, some mechanisms are applied to raise the level of abstraction so that the service composition process can be performed at a higher level. However, such paradigms are very fragile. In the event that some semantic contents are missing, the composition will fail even though the majority of services may be available.

In this paper, we present a service composition approach that is a combination of the bottom-up and top-down strategies. In terms of bottom-up, our approach follows the current service composition methods and tries to use existing Web services to meet the given demand. If the composition process fails, virtual services are proposed by the composer to bridge the semantic gap so that the composition process will continue. Virtual services may not have physical counterparts and may exist only conceptually. However, it provides a high-level specification for a service that is missing. This is a typical top-down manner, i.e., by following the specification, developers and designers will have clear ideas regarding what to develop to meet the given demand. Therefore, our approach can further improve the level of service reuse.

Keywords: Virtual service, Cloud computing, AI planning, Graphplan, Web service composition.

1. Introduction

In cloud computing, infrastructure, platform, and software can all be provided as services. The goal is to relieve users from the burden of managing and maintaining hardware, software, and data and, perhaps, save money by shifting the burden to the cloud service providers [7]. Especially, cloud computing is characterized as making services available on demand. Different demands may involve different sets of services and in different order. This is the place where service composition techniques come into play.

There are currently many different service composition paradigms available, such as AI planning-based paradigm [9][13], pattern-based paradigm [4][12], and workflow-based paradigm [6], etc. Common to these paradigms is that they use different techniques to raise the level of abstraction above the concrete services and perform composition from a high level. This is typically a bottom-up manner. However, the problem with these paradigms is that when a single piece of information is missed, the whole composition process will fail. This hampers the reuse of existing services. Especially, given the huge number of services available in the cloud, simply returning failure leaves no clue for the concerned persons to figure out the nature of the problem.

In this paper, we propose a paradigm that is a mixture of the bottom-up and top-down strategy to improve the level of service reuse. Specifically, we use the AI planning-based paradigm to conduct the composition. If the planning process fails, a virtual service is proposed to connect the existing services to enable the planning process to go through. The planning result will provide valuable information to reveal the nature of the problem. The proposed virtual service provides a specification that serves as the blue print for further analysis, design, and development.

To ensure that the proposed virtual service has practical values, the metrics for virtual services are proposed to guide and evaluate the proposal of the virtual services. We present a novel technique to quantify the "hardness" of propositions, which makes it easier to evaluate the metrics of virtual services, i.e., usefulness, maximum reuse of existing services, and ease of implementation.

This work is an extension of our previous work in [5], in which an algorithm is presented to propose virtual services under the deterministic planning domains. However, deterministic (classical) planning makes several assumptions that are insufficient to express complex control structures, such as conditional and loop constructs. In this paper, we present an algorithm that can propose virtual services in the event of planning failure

978-0-7695-4154-9/10 $26.00 © 2010 IEEE
DOI 10.1109/ICSC.2010.60

under nondeterministic domains. This significantly increases the practical values of the proposed virtual services.

In addition, the algorithm proposed in this paper is an implementation of the ***general*** idea of the virtual service proposal, which is a bi-directional process in the event of composition failure. One direction is to start from the current state and proceed as far as possible towards the goal state. The other direction is to start from the goal and proceed as far as possible towards the current state. Then, a virtual service is proposed to connect these two extreme points to bridge the semantic gap. In this paper, we show how to use Graphplan's [1] intrinsic features, e.g., level-off, etc., to facilitate the implementation of the general idea.

The rest of this paper is organized as follows: In Section 2, we present notational conventions as well as an overview of Graphplan and its favorable features for identifying virtual services. In Section 3, we propose the metrics for evaluating the quality of the virtual services. In Section 4, the algorithm for proposing virtual services under nondeterministic domains is presented, and in Section 5, we conclude the paper and identify some future research directions.

2. Notations and Background

In this section, we first present the notations used in this paper. Then, we review the Graphplan algorithm [1] and its favorable features, based on which our algorithm is designed for virtual service proposal.

2.1. Notations

Typically, an AI planning system can be modeled as a state transition system. An action can trigger a transition from one state to other state(s). In classical planning domains, all actions are deterministic.

A planning ***action*** a can be modeled with the precondition and effect, which is formally denoted as $pre(a)$ and $eff(a)$.

Definition 1 (Deterministic Domain). A deterministic planning domain is a 4-tuple $\Sigma = \langle P, S, A, \gamma \rangle$, where:

- P is a finite set of propositions;
- $S \subseteq 2^P$ is a finite set of states in the system;
- A is a finite set of actions;
- $\gamma: S \times A \rightarrow S$ is the state transition function.

However, as pointed by [6], classical planning systems are insufficient to solve Web service composition. For example, the composite Web services may contain conditional and/or loop constructs that are beyond the expression power of the classical planning systems. Hence, we relax the definition of the state transition function γ to nondeterministic functions, i.e,

$$\gamma: S \times A \rightarrow 2^S$$

The effect of an action is no longer a single state. Instead, it is a set of possible states. In addition, we formally define a planning problem as follows:

Definition 2 (Planning Problem). A planning problem is a triple $\langle s_0, g, A \rangle$, where s_0 is the initial state, g is the goal state, and A is the finite set of actions.

2.2. Graphplan

Graphplan is a deterministic planning algorithm based on planning graph, which is a directed, layered graph interleaved with proposition levels and action levels. As illustrated in Figure 1, given a planning problem $\langle s_0, g, A \rangle$, the planning graph starts from a proposition level P_0, which represents the initial state s_0. Then, all the actions that are applicable to P_0 will be put to the first action level A_1. The effects of the actions in A_1 together with the propositions in P_0 will form the next proposition level P_1. A special type of actions, called "No-op", is used to propagate the propositions from P_0 to P_1. This corresponds to the *graph expansion* phase, during which the planning graph is extended in the forward direction until it has achieved a necessary (but perhaps insufficient) condition for plan existence. Then, the algorithm switches to the *solution extraction* phase, which performs a backward-chaining search on the graph to identify a valid plan.

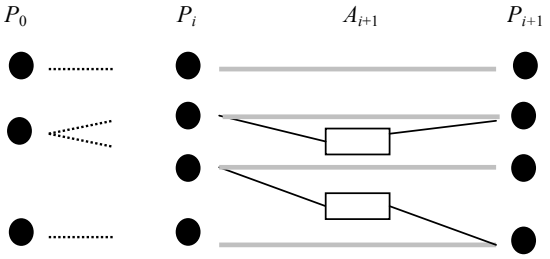

Figure 1: Planning Graph

A valid plan found during the solution extraction phase is a planning-graph where actions at the same level are not mutex, each action's preconditions are made true by the plan, and all the goals are satisfied. If no plans are found, then the termination condition for Graphplan is that if two adjacent proposition levels of the forward planning-graph are identical, i.e., they contain the same set of propositions and have the same mutex relations, then the planning-graph has *leveled off* and the algorithm returns with failure [1].

When level-off happens, the last proposition level contains all the propositions that can be reached from the initial state s_0. In terms of the distance to the goal, the last proposition level is the farthest point that the planning process can go when level-off occurs. This feature can be utilized to facilitate virtual service proposal in Section 4.

3. Metrics for Virtual Services

To ensure that the proposed virtual services indeed have practical values, we define the following metrics for virtual services proposal.

3.1. Maximal Reuse of Existing Services

To some extent, the proposed virtual service works like a magic black box, which connects the existing services to enable the composition process to go through. If we go to an extreme, the precondition of the virtual service contains the initial state of the planning problem; while the effect of the virtual service contains the goal state. Then, the generated plan simply contains this single virtual service. However, such kind of result is not useful in practice. Therefore, the generated plan should reuse as many existing services as possible. This is actually a greedy strategy. Assume that the semantic content required by the given demand is constant, then, the more we reuse the existing services, the less we need to propose to bridge the semantic gap.

In addition, we propose a method to rank the difficulty of the propositions involved in the composition process. We consider the propositions in the goal state as the hardest conditions to reach. This is easy to understand because if the goal is easy to achieve, there is no need to use AI planning to compose the services at all. In contrast, the propositions in the initial state are considered as easiest because they are given. Other propositions are ranked according to their distance to the goal state.

Formally, we use the level-membership (*lms*) function to denote the difficulty of a proposition. Function *lms* is defined as *lms*: $P \rightarrow N$, where P is the set of propositions defined in Definition 1 and N is the set of integers. Specifically,

$\forall p \in P$, $lms(p) = min\{i \mid p \in P_i$ in the planning graph$\}$.

Hence, function *lms* returns the id of the first proposition level in which p appears in the planning graph. Therefore, the smaller the value of $lms(p)$ is, the farther p is from the goal state and the easier to reach the proposition p.

The proposed virtual service is not expected to generate "hard" conditions. In other words, the virtual service should contribute to the goal state indirectly. The effect of the virtual service should enable other existing services to be involved in the composition process and rely on existing services to achieve the goal state.

3.2. Easy to Implement

This metric is two-folds. First, the specification of the virtual service should be informative. The more information it conveys, the easier for the designers to understand the nature of the problem. For example, the precondition of the virtual service could be "true", which means that this service can be applied under any situations. However, such specification is not informative enough and may not truly reveal the nature of the problem.

Second, the proposed virtual service should be practical and easy to implement. To be practical, the precondition of the virtual service should contain "hard" propositions. It means that this virtual service does not rely on the given initial state s_0 and can be applied at a later point during the planning process to contribute to the goal state. On the other hand, the effect generated by the virtual service should contain "easy" propositions. The reason has been explained in Section 3.1. Since the virtual service generates "easy" propositions, we assume that this service is relatively easy to be implemented. Therefore, when multiple choices are possible, the effect of the virtual service should choose easier propositions. This metric is fully consistent with the previous one, i.e., maximum reuse of existing services when deciding the effect of the virtual service.

4. Algorithm for Proposing Virtual Services

Given a planning problem $\langle s_0, g, A\rangle$, the proposed algorithm follows the conventional planning approach trying to find a plan. In the case of failing to find a plan, the algorithm proceeds in two directions. First, it starts from the initial state s_0 and goes as far as possible towards the goal g. Then, it starts from the goal g and goes as far as possible towards the initial state s_0. Finally, a virtual service is proposed to connect these two processes such that both processes will succeed.

In Section 4.1, we present how to transform a nondeterministic domain into a deterministic one so that the classical Graphplan algorithm can be used to solve the nondeterministic problems.

In Section 4.2, we address the issue of how the algorithm can proceed in these two directions to propose the virtual service. The outline of the algorithm is shown in Figure 2. The idea is that a plan is generated by considering only the most likely outcome of each action. The generated plan is called a weak plan, which is defined as "have a chance to achieve the goal, but is not guaranteed to do so" [2]. Then, a subplanning is initiated

for each "omitted" effect in the weak plan. We presented an efficient algorithm in [3] that treats different sub-planning processes as inter-connected and avoids wasting time to explore the same states repeatedly. The proposed algorithm in Figure 2 follows the same idea in [3] to achieve efficient planning. However, efficiency is beyond the scope of this paper. Our focus is on how to propose virtual services in the event of planning failure (i.e., lines 2 – 3 in the function of *GetPlan*).

```
/* Given a planning problem ⟨s0, g, A⟩, a backbone weak plan
*WPb is generated first.
*/
Main(⟨s0, g, A⟩){
1.    Run a Graphplan-based planner on ⟨s0, g, A⟩ and
        obtain WPb;
2.    If (WPb == NULL)
3.        Return "No plan exists!";
4.    Else
5.        Return GetPlan(WPb);
6.    End If
7. }

GetPlan (wp){
1. For each nondeterministic branch i in wp, start a subplanning to
   generate a weak plan wpi;
2.    If failed to find a plan Then
3.        Propose a partial virtual service pvsi to enable the
          current planning process to succeed;
4.    Else
5.        wpi = GetPlan(wpi);
6.    End If
7.    Assemble(wp, wpi);
8. End For
9. Return wp;}
```

Figure 2: Algorithm Outline

The function of *GetPlan* is a recursive function. The termination rule is that there are no nondeterministic branches in the subplanning process. This is guaranteed to happen because we have an additional rule: If action a_1 is the action generating the effect in branch i in line 1 (*GetPlan*), a_1 will be removed from the next recursive call in line 5 to avoid a_1 interfering the subsequent planning process. Hence, the size of the resulting action set A' will be at least one less than the original action set A. Since A is a finite set, there will be no infinite recursive calls.

4.1. Nondeterministic Domain to Deterministic Domain Transformation

We employ a method similar to that introduced in [10] to transform a nondeterministic domain to a deterministic one so that a classical AI planner can work on it. Essentially, each nondeterministic action with multiple possible effects is converted into a set of deterministic actions, with one deterministic action for each effect. To illustrate this, let φ be a nondeterministic action and $E_\varphi = \{e_1, e_2, \ldots, e_n\}$ denote the set of φ's effects. Also, let $A_\varphi = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ denote the corresponding set of deterministic actions generated from φ. We have

$\forall i, pre(\alpha_i) = pre(\varphi)$ and,
$eff(\alpha_i) = e_i$.

4.2. Virtual Services Proposal

In this section, we address the issue of how to conduct planning in two directions to propose the virtual service when the conventional planning process fails. In both directions, we want to go as far as possible so that the semantic gap will be as small as possible.

4.2.1 Using Level-Off to Identify the Extreme Levels in Both Planning Directions. Recall that a planning graph will level off if no plan is possible. Semantically, level-off means that all the reachable propositions have been included in the planning graph while the goal state is still not met. In other words, we cannot go any further towards the goal state beyond the point of level-off. This perfectly fits in our needs, i.e., trying to go as far as possible toward the goal.

For the reverse direction, i.e., from the goal state to the initial state, we follow the same idea. Nevertheless, the planning process is based on the reverse actions. The idea of reverse actions is used in [8], in which the planning graph first grows backwardly to the initial state. Then, the resulting graph is used to guide the normal planning process. Although the reverse actions are not as rigorously defined as the normal actions, they can facilitate the backward planning process and help us identify the nearest point to the initial state in the backward planning. We first define the reverse operator $^{-1}$ to facilitate the subsequent discussion.

Definition 3 (Reverse Operator $^{-1}$): The reverse operator is defined over $2^P \times 2^P \rightarrow 2^P \times 2^P$ such that $(Pre, Post)^{-1} = (Post, Pre)$, where P is the set of propositions in the planning domain in Definition 1 and $Pre, Post \in 2^P$.

Given an action a, $pre(a^{-1}) = eff(a)$ and $eff(a^{-1}) = pre(a)$. We can extend the definition of the reverse operator to planning problems as well. Hence, $\langle s_0, g, A\rangle^{-1} = \langle g, s_0, A^{-1}\rangle$ and $\forall a \in A \Rightarrow a^{-1} \in A^{-1}$.

Therefore, to go as far as possible towards s_0 from g, we simply run the planner for the reverse planning problem until level-off takes place. In [5], we have shown that level-off will definitely happen in the reverse

planning problem $\langle g, s_0, A^{-1}\rangle$ if the regular forward planning $\langle s_0, g, A\rangle$ fails.

4.2.2 The Precondition of the Virtual Service. After both planning processes level off, we assume that the last proposition level in the planning problem $\langle s_0, g, A\rangle$ is P_{lf} and the last proposition level in the planning problem $\langle g, s_0, A^{-1}\rangle$ is P_{lb}. Then, we calculate the precondition of the virtual service *pvs* as

$$pre(pvs) = P_{lf} - s_0 - P_{lb}$$

This fits in the metrics for virtual service proposal. In Section 3.2, we conclude that the precondition of the virtual service should contain "hard" propositions measured by the level membership function *lms*. Therefore, all the propositions in s_0 are removed first because they are given and considered as "easy". Next, the overlapping portion in P_{lf} and P_{lb} is removed. This follows the following heuristic. When an action is applied, part of the preconditions will no longer be valid and, thus, will be moved into the delete effect. We consider this part as the essential part in the semantic meaning of an action. Since the virtual service *pvs* is used to connect the two planning processes, the proposition level P_{lb} is treated as the next proposition level after P_{lf}. Hence, the propositions in $P_{lf} - s_0$ but not in P_{lb} are considered as the essential part of the precondition.

4.2.3 The Effect of the Virtual Service. According to the metrics in Sections 3.1 and 3.2, the propositions in the effect of the virtual service should be "easy" so that it will be easily implemented and only contribute to the goal state indirectly. The effect is proposed as follows,

$$eff(pvs) = (P_{lb} - g - P_{lf}) \cup (g - \bigcup_{a \in A_{lb}} pre(a)),$$

where A_{lb} is the last action level in the backward planning graph. In the first step, we remove all the propositions in g from P_{lb} because we consider that propositions in g are "hard" to achieve. Then, we remove the overlapped portion in P_{lb} and P_{lf} because this set of propositions is reachable from the initial state s_0 while the virtual service is used to bridge the semantic gap and its effect should contain propositions that are not reachable from s_0.

The last part $(g - \bigcup_{a \in A_{lb}} pre(a))$ is a little complex. This set of propositions is a subset of the goal state. They reach the last proposition level because of the "No-op" actions passing them all the way there. Hence, they are not consumed by any actions in the backward planning process $\langle g, s_0, A^{-1}\rangle$. Since all the actions in this planning process are in the reverse form of the original planning problem $\langle s_0, g, A\rangle$, we can equally say that this set of propositions is not generated by any actions during the planning process $\langle s_0, g, A\rangle$. The only way to generate them is to include them in the effect of the virtual service so that the semantic gap is filled.

Due to the use of "No-ops", all the actions that appear in the current action level will appear in the next action level as well. As a result, the last action level A_{lb} contains all the actions in the planning graph. $\bigcup_{a \in A_{lb}} pre(a)$ is the set of propositions that are consumed by some actions in the planning graph. Hence, $(g - \bigcup_{a \in A_{lb}} pre(a))$ is the set of propositions that are not consumed by any actions and should be contained in the effect of the virtual service.

In summary, the algorithm that is designed for virtual service proposal is shown in Figure 3.

```
/*pvs: is the virtual service to be proposed*/
Propose_Virtual_Service(⟨S0, G, A⟩)
1.  Start planning ℘1 on the planning problem ⟨S0, G, A⟩;
2.      If level-off happens Then
3.          Start another planning process ℘2 on the planning
              problem ⟨G, S0, A^-1⟩
4.          Assume that Plf is the proposition level that level-off
              is identified in ℘1; and Plb is the proposition level
              that level-off is identified in ℘2.
5.          pre(pvs) = Plf − S0 − Plb;
6.          eff(pvs) = (Plb − G − Plf) ∪ (G − ⋃_{a∈Alb} pre(a));
7.      Else
8.          Generate the plan;
9.      End If
```

Figure 3: The Algorithm for Virtual Service Proposal

4.3. Combine Partial Virtual Services into a Complete Virtual Service

The function "Propose_Virtual_Service" shown in Figure 3 is invoked in line 3 in Figure 2, the outline of the algorithm. The function proposes a partial virtual service for each specific subplanning. For the sake of simplicity, we will combine all the partial virtual services into a complete service at the end.

Formally, the whole planning process proposes a set of partial virtual services $S_v = \{pvs_1, pvs_2, \ldots, pvs_n\}$. According to the PDDL (Planning Domain Definition Language) [11], these partial virtual services can be connected together by using the conditional effects, which have the following form:

```
(When
    CONDITION_FORMULA
    EFFECT_FORMULA)
```

Hence, the conditional effect for the complete virtual service will be composed as follows:

(When $pre(pvs_1)$ $eff(pvs_1)$)
(When $pre(pvs_2)$ $eff(pvs_2)$)
… …
(When $pre(pvs_n)$ $eff(pvs_n)$)

5. Conclusions and Future Research Directions

In this paper, we presented an algorithm to identify virtual services in the event of composition failure. Virtual services are used to bridge the semantic gaps and enable the composition process to continue even though the exiting service composition method may fail. This algorithm works on nondeterministic planning domains and, hence, is capable of solving practical problems. The metrics of evaluating the quality of the proposed virtual services are also proposed. To facilitate the evaluation, we present a technique to rank the difficulty of propositions such that the evaluation process can be quantified. This technique significantly simplifies the evaluation process and provides valuable information to guide the proposal of virtual services.

Virtual services are treated in the same way as regular services during the composition and, more importantly, their specifications are useful for further analysis, design, and development. This is especially useful under the cloud computing environment as the number of available services can be huge. The generated plan will help reveal the nature of the demand and the virtual service will help the concerned persons develop concrete ideas about what is lacking and how to implement the missing content. This is typically a top-down fashion to develop services. To the best of our knowledge, we are the first to combine the bottom-up and top-down strategies together to improve the level of service reuse.

In the next step, we will turn our focus on proposing virtual services with QoS concerns. The proposed virtual services are not only used to bridge the semantic gap, but also maximize QoS during the selection of services by considering reputation, price, duration, reliability, etc. This will enable virtual services to provide more useful information and better reflect the true needs in cloud computing.

6. References

[1] A. Blum and M. Furst, "Fast planning through planning graph analysis", *Artificial Intelligence*, 90, 1997, pp. 281–300.

[2] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, "Weak, strong, and strong cyclic planning via symbolic model checking", *Artificial Intelligence*, 147(1-2), 2003, pp. 35–84.

[3] J. Fu, F. B. Bastani, V. Ng, I. Yen, and Y. Zhang, "FIP: A Fast Graphplan-Based Iterative Planner", 20*th IEEE International Conference on Tools with Artificial Intelligence*: pp. 419–426, 2008.

[4] J. Fu, F. Bastani, I. Yen, W. Hao, "Using Service Patterns to Achieve Web Service Composition", *ICSC* 2009: 402-407.

[5] J. Fu, W. Hao, M. Tu, B. Ma, J. Baldwin, and F. Bastani, "Virtual Services in Cloud Computing", *will appear in the proceedings of SERVICES* 2010, 2010

[6] J. Gekas and M. Fasli, "Automatic Web Service Composition Based on Graph Network Analysis Metrics", *OTM Conferences* (2) 2005: 1571-1587.

[7] B. Hayes, "Cloud computing", *Communications of the ACM*, Volume 51, Issue 7, July 2008.

[8] R. Kambhampati, E. Paeker, and E. Lambrecht, "Understanding and extending graphplan", In *Proc. 4th European Conference on Planning*, Sept. 1997.

[9] M. Klusch, A. Gerber, and M. Schmidt, "Semantic web service composition planning with OWLS-Xplan," *Proc.* 1*st Intl. AAAI Fall Symposium on Agents and the Semantic Web*, Arlington VA, USA, 2005.

[10] U. Kuter, D. Nau, E. Reisner, R. P. Goldman, "Using Classical Planners to Solve Nondeterministic Planning Problems", Proceedings of *ICAPS*-08, pp. 190-197.

[11] D. McDermott, et al., "The PDDL Planning Domain Definition Language", *the AIPS*-2004 *Planning Competition Committee*, 2004.

[12] A. t. Teije, F. v. Harmelen, B. Wielinga, "Configuration of Web Services as Parametric Design", *Proceedings of the Proceedings of the* 14*th International Conference on Knowledge Engineering and Knowledge Management*, pp. 321-336, 2004.

[13] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia, "Automatic Web services composition using SHOP2", *In Workshop on Planning for Web Services*, Trento, Italy, June 2003.