Self-Adaptive Service Composition through Graphplan Repair

Yuhong Yan*, Pascal Poizat†‡ and Ludeng Zhao*
*Concordia University, Montreal, Canada
Email: {yuhong,ludeng.zhao}@encs.concordia.ca
†University of Evry Val d'Essonne, Evry, France
‡LRI UMR 8623 CNRS, Orsay, France
Email: pascal.poizat@lri.fr

***Abstract*—Service composition is nowadays mainly seen as a once-for-all activity. Supporting a dynamic service world, where both available services and needs may change, requires run-time adaptive features for service composition. In this paper we propose a repair technique for internal composition adaptation, as opposed to external adaptation. Moreover, setting up our proposal in the planning framework, we compare our repair technique with reference to re-composition, that is re-planning.**

I. Introduction

The fulfillment of business requirements and end-user needs by value-added applications built over reusable services is fostered by service composition [11]. Complementary to manual development which is often tedious and error-prone, automatic *(Web) service composition* (WSC) techniques have been proposed [3], [6]. Due to its support for under-specified requirements, *AI planning* is increasingly applied to WSC [12]. This technique generates compositions under the form of a *plan* where service invocations are chained on an input-output basis from user given information to user needs, or goals.

Problem. In real world, change is rather the rule than the exception. While most WSC algorithms endorse a static viewpoint over service composition, the composition environment, *i.e.,* the set of used services and the needs, can change. Due, *e.g.,* to user mobility or network failure, the available services may become unavailable, causing needs to be unsatisfiable. User needs may also change opportunistically, *e.g.,* when at an airport, a traveller realizes that duty-free shopping is available. In both cases, the composition model (and accordingly its orchestration or choreography implementation) becomes invalid. *Adapting* to change can be done externally or internally. *External adaptation* tries to bridge the gap between the unchanged composition and its new environment by computing adapters. *Internal adaptation* means evolving the composition. One may *re-compose*, *i.e.,* re-compute a new composition from scratch or, more efficiently, from the place impacted by change. Alternatively, one may *repair* a composition by performing only local modifications in it.

This work is supported by project "Building Self-Manageable Web Service Process" of Canada NSERC Discovery Grant, RGPIN/298362-2007 and by project "PERvasive Service cOmposition" (PERSO) of the French National Agency for Research, ANR-07-JCJC-0155-01.

Our Solution. We tackle WSC in an AI planning context, more precisely using planning graphs [5]. They enable a compact representation of relations between services and enable one to model the whole problem world. Moreover, even after some changes, part of a planning graph is still valid. In this context, we propose an internal adaptation technique for WSC based on planning graph repair. Our solution is based on a greedy search process which starts from a level in the planning graph where service inputs or user goals have become unsatisfiable due to change. The process then tries to add new services that can provide them, possibly generating, in turn, new unavailable inputs. Service selection is not only based on goal satisfaction but also triggered by heuristics.

Contributions. Our contributions can be summarized as follows. We propose a novel solution for adaptive WSC, going beyond 1-to-1 service replacement but also avoiding re-composition by applying *plan repair* principles [13], [4] to the WSC context. We carefully evaluate our repair technique with reference to planning based re-composition from scratch, *i.e., replanning*. Our experiments demonstrate that our repair technique is faster, still getting equivalent results in terms of quality of the adapted compositions (smaller number of services and resemblance to the original composition plan). Efficiency is achieved through service selection that is not only made on the basis of required data production but also using heuristic functions. Additionally, our experiments are done on big-size data sets using a standard benchmark framework for WSC.

Related Work. External adaptation originates from component-based frameworks and has more recently been applied to services [2], [7], [10]. External adaptation is well suited to cases where interacting entities cannot be evolved (black box entities). However, in case of multiple changes, external adaptation can lead to adapters stack (if adapters are adapted) or degrade to a form of re-composition (adapter re-built from scratch). Some service middleware, *e.g.,* [9], [8], proposes automatic adaptive features for WSC. Still, they are either limited to 1-to-1 service replacement (which may not be sufficient in practice when some service is to be replaced by the composition of several other ones) or proceed by re-composition (which we want to avoid). The principles of plan repair have been introduced as an alternative to

re-planning from a theoretical perspective in the AI planning community [13], [4]. However, the causal relation between services is not similar to the typical operations in AI planning. Thus, the way to evolve partially valid plans into a new solution is specific to our proposal. Up to our knowledge, we have proposed the first application of plan repair to WSC. A preliminary version of this work is presented in [14]. When a service without any substitute was removed, the solution in [14] degraded into a backward search from the goal to the state before service removal. To the contrary, the technique we propose here better reuses the original planning graph structure and has much better success rate.

Organization. Service composition models and graph planning for WSC are introduced in section II. Our solution for adaptive service composition is then presented in section III and we perform an empirical evaluation of it in section IV.

II. Models and Introduction to Planning

We consider a Web service is stateless and its internal behavior is unknown. We suppose a set D of concepts (coming from an ontology used to annotate Web services). A Web service w can be modelled as as tuple (in, out) where $in \subseteq D$ denote the input parameters of w and $out \subseteq D$ denote the output parameters of w. We denote $in^- \subseteq in$ the set of input parameters that are consumed by the service.

Composition requirements correspond to user needs. A composition requirement is a couple $(D_U^{\text{in}}, D_U^{\text{out}})$ where $D_U^{\text{in}} \subseteq D$ is the set of provided (or input) parameters and $D_U^{\text{out}} \subseteq D$ is the set of required (or output) parameters.

Definition 1: A composition requirement $(D_U^{\text{in}}, D_U^{\text{out}})$ is satisfied by a set of Web services $W = \{w_1, \ldots, w_n\}$ iff, $\forall i \in \{1, \ldots, n\}$:

- $\forall d \in in(w_i), \exists d' \in D_U^{\text{in}} \cup out(w_1) \cup \ldots \cup out(w_{i-1}), d' \sqsubseteq d$ and
- $\forall d \in D_U^{\text{out}}, \exists d' \in D_U^{\text{in}} \cup out(w_1) \cup \ldots \cup out(w_n), d' \sqsubseteq d$

The output of one Web service can be the input of another Web service if they are identical syntactically and semantically or can be related by semantic subsumption ($\sqsubseteq$).

AI planning has been successfully applied to solve the WSC problem through its encoding as a planning problem [12], [6].

Definition 2: A *planning problem* [5] is a triple $P = ((S, A, \gamma),\ s_0,\ g)$, where

- S is a set of states, with a *state* s being a subset of a finite set of proposition symbols L, where $L = \{p_1, \ldots, p_n\}$.
- A is a set of actions, an *action* a being a triple $(precond, effects^-, effects^+)$ where $precond(a)$ denotes the preconditions of a, and $effects^-(a)$ and $effects^+(a)$, with $effects^-(a) \cap effect^+(a) = \emptyset$, denote respectively the negative and the positive effects of a.
- γ is a state transition function such that, for any state s where $precond(a) \subseteq s$, $\gamma(s, a) = (s - effects^-(a)) \cup effects^+(a)$.
- $s_0 \in S$ is the initial state.
- $g \subseteq L$ is a set of propositions called *goal*.

A *plan* is a sequence of actions $\pi = \langle a_1, \ldots, a_k \rangle$, $k \geq 0$.

WSC can be mapped to a planning problem as follows [15]:

- a service w is mapped to an action w. Service input parameters are mapped to action preconditions ($in(w) \mapsto precond(w)$), output parameters to positive effects ($out(w) \mapsto effects^+(w)$), and consumable parameters to negative effects ($in^-(w) \mapsto effects^-(w)$).
- the composition requirement is mapped to the initial state and the goal ($D_U^{\text{in}} \mapsto s_0$ and $D_U^{\text{out}} \mapsto g$).

GraphPlan is one kind of planning techniques, the centric piece of which is a planning graph. In a planning graph, a *layered plan* is a sequence of sets of actions $\langle \pi_1, \pi_2, \ldots, \pi_n \rangle$, in which each π_i ($i = 1, \ldots, n$) is independent. A planning graph iteratively expands itself one level at a time (ref. Figure 2(Left)). This process continues until either it reaches a level where the proposition set contains all goal propositions or a fixed point level. The goal cannot be attained if the latter happens first. The planning graph searches backward from the last level of the graph for a solution. It is known that a planning graph can be constructed in polynomial time.

In a planning graph we use P_i and A_i to represent the set of propositions and the set of actions at level i. Thus, the initial proposal level D_U^{in} is P_0, the first action level is noted as A_1, and, in case of a solution, the last proposition level P_n contains D_U^{out}. Based on the above assumptions, we have $P_{i-1} \subseteq P_i$ and $A_{i-1} \subseteq A_i$.

III. Self-Adaptive Planning Graphs

After considering the changes, we start with a partially valid planning graph G which contains unsatisfied preconditions BP and unimplement goals Ω. At a proposition level i, the unsatisfied preconditions are $BP_i \subseteq P_i$, and $BP_n = \Omega$. We develop an adaptive algorithm. which grows the partial planning graph heuristically so as to rapidly obtaining a graph containing at least one solution. The resulted graph is only part of a "full" planning graph if we start from scratch. When extracting a solution from this graph using backward search, it can be faster than extracting a solution from a "full" planning graph.

Our adaptive algorithm is a greedy search process which starts from the highest level m where $BP_m \neq \emptyset$ (Fig. 1(a)). We try to find a set of services A which can take the propositions at the $m-1$ level as inputs and can satisfy BP_m. If such A exists, A is added to the m-th action level (Fig. 1(b)), and the m-th proposition level is fixed. We can move down to $m-1$-th level. Figure 1(c) shows a more complex case. In this case, we can find a set A to satisfy BP_m. However, A needs more propositions than P_{m-1}. In this case, introducing A causes more unsatisfied preconditions. Rather than adding A into the m-th action level, we prefer to create another action level above m-th proposition to be the new $m+1$-th action level and add A to it. The newly created $m+1$-th proposition level copies the original m-th proposition level, except the broken conditions are satisfied by A (thus $BP_{m+1} = \emptyset$).

We use two evaluation functions to select services to be added into the graph. Assume we want to add a service w to

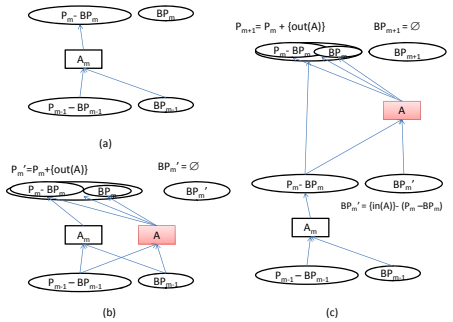

Fig. 1. Insert a new level in partial Planning Graph (a) original, (b) A can be satisfied by P_{m-1}, (c) add a new level for A

the highest action level n, the evaluation function is:

$$f(G,w) = |\Omega \cap out(w)| + |P_{n-1} \cap in(w)| - |in(w) - P_{n-1}| \quad (1)$$

Where $|\Omega \cap out(w)|$ is the number of unimplemented goals that can be implemented by w;
$|P_{n-1} \cap in(w)|$ is the number of the inputs of w that can be provided by the known parameters at the level P_{n-1};
$|in(w) - P_{n-1}|$ is the number of the inputs of w that **cannot** be provided by the known parameters at the level P_{n-1}. This set needs to be added into BP, if w is added.

Assume we want to add an action w to action level m, and m is not the goal level, the evaluation function is:

$$f(G,w) = |\Omega \cap out(w)| + |\sum_{i \geq m} (BP_i \cap out(w))| + |P_{m-1} \cap in(w)| - |in(w) - P_{m-1}| \quad (2)$$

Compared to equation 1, the above equation added term $|\sum_{i \geq m} (BP_i \cap out(w))|$ which is the number of the broken propositions in the level P_m and above that can be satisfied by the outputs of w.

Let us illustrate our repair algorithm with an example.

The request is ({a}, {e}) and we have nine services: A2BC: a→b,c, A2D: a→d, C2E: c→e, D2E: d→e, D2F: d→f, F2G: f→g, F2H: f→h, G2E: g→e and H2I: h→i. The resulting planning graph is shown in figure 2(left) and solutions (plans) are A2BC;C2E and A2D;D2E. Let us suppose we commit to the second one. This is the plan we need to repair if anything is broken. We prefer to reuse as many as possible the original services in a new composition, because they are the commitment we need to hold. If we remove both C2E and D2E, the graph is broken, because e is no longer produced by any service. This yields a partial planning graph (Fig 2(left)).

If we had used replanning, we would simply run the planning algorithm again with a new set of available services and a set of updated goals. We would then get a new planning graph (Fig. 3(left)), from which we can get two new solutions: A2D;D2F;F2G;G2E and A2D;D2F;F2H;H2I. If we rather use

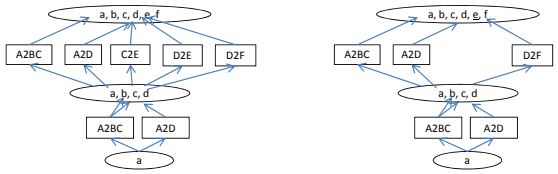

Fig. 2. Planning Graphs. (Left:) original; (Right:) after removal of C2E and D2E.

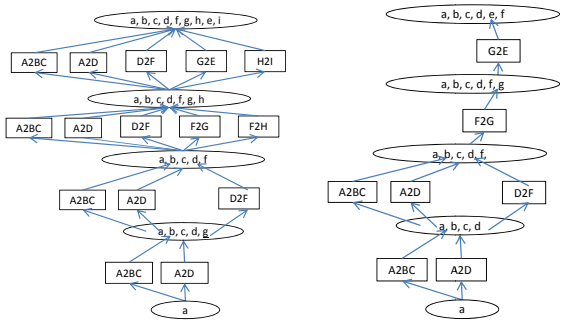

Fig. 3. Planning Graphs. (Left:) by replanning; (Right:) grown by our repair algorithm.

our repair algorithm, we obtain another graph (Fig. 3(right)) and solution A2D;D2F;F2G;G2E. Our approach is not to build an entire planning graph again, but to fast "grow" the partial planning graph again in order to obtain a feasible solution. During the growing process, we heuristically search into a direction that can satisfy the broken goals and preconditions, as well as making use of the existing partial graph.

Due to greedy search, our adaptive algorithm sometimes does not find a solution even if one exists. However, with the WSC data set (ref. Section IV), an added service does not lead us to a dead end, because it always has matching services in the other levels. Therefore, we can always success if a solution exists with the WSC data set.

IV. Empirical Evaluation

We use the Web Service Challenge (WSC) 2009 [1] platform and its tools in our experiments. The platform can invoke the composition algorithm as a Web service and evaluate composition time. The data generator generates ontology concepts in OWL and a set of Web services interfaces in WSDL which use the concepts. Subsumption is modeled in OWL and is used during composition. Negative effects are not modeled in this kind of data sets. Given a number as solution depth, the generator algorithm generates a group of solutions, each of which has the given solution depth. At each level, multiple services can substitute each other as they use the same input set and produce the same outputs to be used in the next level. Any two solution groups do not share any services. The generator also randomly generates a set of "padding" Web services around these services used in solutions. These

"padding" services do not have the outputs that can be used by the next level.

We generated a data set with 351 available services which use 2891 parameters in their input and output messages. These parameters are from 6209 instances of 3081 concepts defined in an OWL file. This data set has four solution groups. Group 1 contains solutions with 9 levels, Group 2 contains solutions with 18 levels, Group 3 contains solutions with 19 levels, and Group 4 contains solutions with 27 levels.

The purpose of this experiment is to compare our repair algorithm with the replanning algorithm applied to the updated problem. We pick up a solution in Group 1. The experiments remove different numbers of services in this solution. At each experiment, we use repair and replanning algorithms to find solutions. We compare their composition time, as well as the quality of found solutions against the criteria of number of services in the plan, levels in the plan, and distance to the original plan (the number of actions not in the original plan, plus the number of actions in the original plan but not in the new plan [4]). We repeat the same experiment with a solution in Group 2.

The following comparison of performance is recorded in the cases that both repair and replanning algorithms can find solutions. Each data point is obtained from the average of three independent runs.

Figure 4 shows that the repair algorithm is faster than the replanning algorithm. Figure 5 and Figure 6 compare the number of services and the level of services in the solutions found by the two algorithms. At the left graphs in the two figures, we see that the replanning algorithm "jumps" to other solution groups which have more service numbers, but less levels, than the solutions found by the repair algorithm. However, it is just a random case. At the right, we see that with solution 2, the two algorithms have similar performance - actually they find the same solutions in most cases. Similarly, in Figure 7 left, we find the repair algorithm finds solutions closer to the original solutions than the replanning algorithm with solution 1. However, with solution 2, the two algorithms find the same solutions, thus the same distance to the original composition. We can draw the conclusion that repair algorithm works faster than the replanning algorithm in composition time and the two algorithms have similar performance in the quality of the solutions.

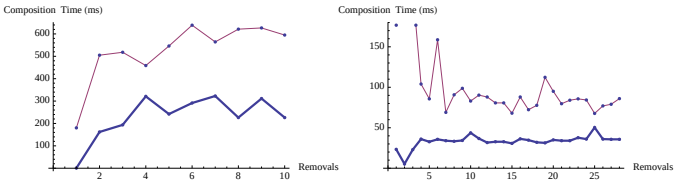

Fig. 4. Composition time with solution 1 (left) and solution 2 (right) (repair - thick line, replanning - thin line)

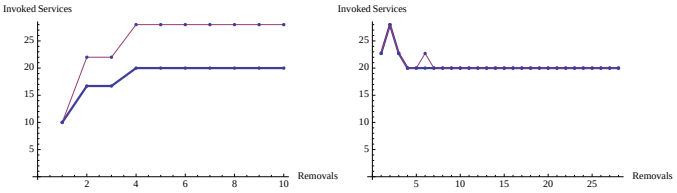

Fig. 5. Number of Services Invoked with solution 1 (left) and solution 2 (right) (repair - thick line, replanning - thin line)

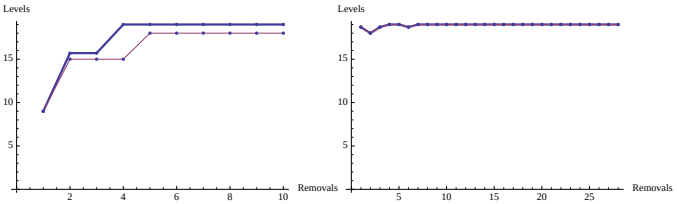

Fig. 6. Number of Levels with solution 1 (left) and solution 2 (right) (repair - thick line, replanning - thin line)

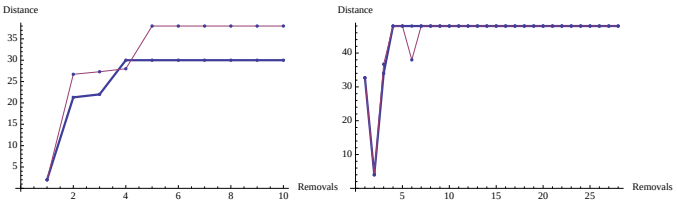

Fig. 7. Distance to the Original Plan with solution 1 (left) and solution 2 (right) (repair - thick line, replanning - thin line)

REFERENCES

[1] Web service challenge 2009 web site. http://ws-challenge.georgetown.edu/wsc09/index.html.

[2] A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. In *Proc. of ICSOC*, 2006.

[3] S. Dustdar and W. Schreiner. A Survey on Web Service Composition. *International Journal of Web and Grid Services*, 1(1):1–30, 2005.

[4] M. Fox, A. Gerevini, D. Long, and I. Serina. Plan Stability: Replanning versus Plan Repair. In *Proc. of ICAPS*, 2006.

[5] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers, 2004.

[6] A. Marconi and M. Pistore. Synthesis and Composition of Web Services. In *Proc. of the 9th International School on Formal Methods for the Design of Computer, Communications and Software Systems: Web Services (SFM)*.

[7] R. Mateescu, P. Poizat, and G. Salaün. Adaptation of Service Protocols using Process Algebra and On-the-Fly Reduction Techniques. In *Proc. of ICSOC*, 2008.

[8] H. Meyer, D. Kuropka, and P. Tröger. ASG - Techniques of Adaptivity. In *Proc. of the Daghstuhl Seminar on Autonomous and Adaptive Web Services*, 2007.

[9] O. Moser, F. Rosenberg, and S. Dustdar. Non-Intrusive Monitoring and Service Adaptation for WS-BPEL. In *Proc. of WWW*, 2008.

[10] H. R. Motahari Nezhad, G. Y. Xu, and B. Benatallah. Protocol-aware matching of web service interfaces for adapter development. In *Proc. of WWW*, 2010.

[11] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: a Research Roadmap. *International Journal of Cooperative Information Systems*, 17(2):223–255, 2008.

[12] J. Peer. Web Service Composition as AI Planning – a Survey. Technical report, University of St.Gallen, 2005.

[13] R. van der Krogt and M. de Weerdt. Plan repair as an extension of planning. In *Proc. of ICAPS*, 2005.

[14] Y. Yan, P. Poizat, and L. Zhao. Repairing service compositions in a changing world. In *Studies in Computational Intelligence, selected papers from SERA'2010*, 2010.

[15] X. Zheng and Y. Yan. An Efficient Web Service Composition Algorithm Based on Planning Graph. In *Proc. of ICWS*, 2008.