Stepwise and Asynchronous Runtime Optimization of Web Service Compositions*

Philipp Leitner, Waldemar Hummer, Benjamin Satzger, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, A-1040, Vienna, Austria
lastname@infosys.tuwien.ac.at

Abstract. Existing research work considers runtime adaptation of service compositions as a viable tool to prevent violations of service level agreements. In previous work we have formalized the optimization problem of identifying the most suitable adaptations to prevent a predicted set of violations, and presented suitable algorithms to solve this problem. Here, we introduce the idea of stepwise optimization as a solution to the problem of how to deal with situations when the optimization result is not available in time, i.e., when decisions need to be taken before the optimization problem can be fully solved.

1 Introduction

In information systems based on the concept of Service-Oriented Architecture (SOA), business processes are implemented as higher-level compositions of Web services (service compositions [1]). Providers of service compositions often guarantee certain quality characteristics using service level agreements (SLAs). Basically, SLAs are collections of target qualities (service level objectives, SLOs) and monetary penalties that go into effect if the promised target quality cannot be achieved. Hence, providers of service compositions have strong incentives to prevent cases of SLA violation. One promising approach to achieve this is predicting violations at runtime, before they have actually occurred, and using adaptation to prevent these violations [2, 3]. Evidently, an important part of runtime adaptation is deciding which adaptations to apply. We argue that this decision should be based on both, the costs of violation (the penalties associated with SLOs), and the costs of adaptation. We have presented a formalization of this decision process as an optimization problem as part of our work on the PREvent (Event-Based Prediction and Prevention of SLA Violations) project [4]. However, a limitation of the PREvent approach so far is that it is inherently assumed that the optimization problem can be solved in time, before the first adaptation has to be applied. Even using fast meta-heuristics this is not guaranteed, especially so for shorter service compositions.

* The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 215483 (S-Cube) and grant agreement 257483 (Indenica).

A. Bouguettaya, M. Hauswirth, and L. Liu (Eds.): WISE 2011, LNCS 6997, pp. 290–297, 2011.

In this paper, we improve on this by proposing an asynchronous and stepwise optimization model, in which we do not generate a decision for all adaptations at once. Instead, we run the optimization in parallel to the execution of the composition. At so-called decision points we make a decision for only those adaptations that absolutely need to be decided at that moment, update the optimization problem according to the made decisions, and continue optimizing for all remaining possibilities for adaptation. Note that the contribution presented here is not specific to the approach presented in [4]. Much more, the same ideas are applicable for other runtime adaptation approaches facing similar problems as well, e.g., [3].

2 Runtime Optimization of Service Compositions

The decision which combination of runtime adaptations is best suited to prevent one or more predicted SLA violations in an instance of a service composition can be formulated as an optimization problem. For this paper the concrete structure of this optimization problem is not essential. However, for ease of understanding we briefly summarize the formalization used in PREvent, which is the basis of the remaining discussions. This model has originally been presented in [4].

We assume the following inputs to the optimization. Let I be the set of all possible instances of the service composition, and let $i \in I$ be one instance that we need to optimize. Furthermore, let the relevant SLA be given as a set of SLOs $S = \{s_1, s_2, \dots s_k\}$. Every SLO s has an associated penalty function, which governs the payment that the composite service provider has to pay based on a measured SLO value m_s. Penalty functions are defined as $p_s : \mathbb{R} \to \mathbb{R}, s \in S$. We refer to the collection of all penalty functions as $P = \{p_{s1}, p_{s2}, \dots p_{sk}\}$. Moreover, let $A = \{a_1, a_2, \dots a_l\}$ be the set of all possible adaptations, and $A^* \in \mathcal{P}(A)$ one concrete subset of adaptations. Applying adaptations transforms composition instances, which we capture with the $\circ$ operator ($\circ : I \times \mathcal{P}(A) \to I$). For simplicity, we assume that all adaptations have constant costs, defined as $c : A \to \mathbb{R}$. However, adaptations are not necessarily independent, i.e., there can be constraints on which adaptations can be selected at the same time. We use a simple penalty term to express that two adaptations are mutually exclusive ($v(A^*) = \infty$ if A^* contains at least one constraint violation, $v(A^*) = 0$ otherwise).

With these definitions, we can can describe the total costs (TC) of a composite service provider for one instance of the business process as $TC(A^*) = v(A^*) + \sum_{s_x \in S} p_{s_x}(i \circ A^*) + \sum_{a_x \in A^*} c(a_x) \to min!$ In this definition, the first term is the potential penalty for constraint violations in A^*. The second term represents the costs accrued via penalty payments for SLA violations. Finally, the third term represents the costs of adaptation. Naturally, the goal of the provider is to select A^* so that TC is minimal for a given i.

Evidently, the exact penalties that have to be paid ($p_{s_x}(i \circ A^*)$) are unknown at runtime for any combination of adaptations (even if we do not apply any adaptations, we still do not know for sure which SLOs are going to be violated). However, we assume that it is possible to predict the penalty before and after

adaptation with a reasonably small estimation error, using a set of estimation functions $e_s : I \rightarrow \mathbb{R}, s \in S$. We can then replace the penalty payments with their estimation, leading to the following estimation: $TC(A^*) \approx v(A^*) + \sum_{s_x \in S} e_{s_x}(i \circ A^*) + \sum_{a_x \in A^*} c(a_x) \rightarrow min!$

In practice, estimation can be implemented for instance use machine learning based regression, as presented in [5, 6]. Using this technique one can estimate monitorable SLO values m_s in advance, and use these estimated values to calculate what the penalty will be after applying a given set of adaptations. This is the approach that we have chosen to follow in the PREvent framework, but in principle our model is not restricted to these machine learning based estimation functions.

3 Stepwise Service Composition Optimization

The problem presented in Section 2 can be solved at runtime, however, generating a good solution may be time-consuming. One promising approach that we have utilized in PREvent with good results are genetic algorithms [7] (GA). In the remainder of the paper, we assume that the runtime optimization of A^* is implemented using GA, however, the general principles presented here still apply if other means are used. However, even using GA, optimization is still time-consuming. Therefore, in order not to delay the execution of the composition, it is desirable to execute the optimization asynchronously, i.e., in parallel to the service composition. However, in this case we need to keep timing aspects of the optimization and the composition in mind.

Before explaining optimization timing, we need to concretize what adaptation actually means in the scope of this paper. In general, we assume that adaptation can either be implemented via service rebinding, i.e., exchanging a service in the composition for another, or via structural adaptation of the composition, i.e., freely adding, removing or modifying activities in the composition. Figure 1 exemplifies these types of adaptation. This adaptation model is in line with related work, as other approaches to self-adapting compositions usually assume similar possibilities for adaptation [8, 9]. The excerpt in Figure 1 is a small part of an assembling case study presented in [4]. We will use this example in the remainder of the paper. Note that even though we present our work on a simple sequential process for simplicity, the same ideas can be used for arbitrarily complex composition graphs, as long as they are circle-free.

For any adaptation of the types discussed above we can identify the affected region in the service composition, i.e., the activities in the composition which are affected by the adaptation. For rebinding, this is exactly one activity. For structural adaptation the affected region may be arbitrarily large, but is still always clearly defined. In the remainder of this paper, we use the term "beginning of the affected region" (t_a^x) as the time that the first activity affected by adaptation x starts to execute. In Figure 1, the beginning of the affected region is indicated by **X**.

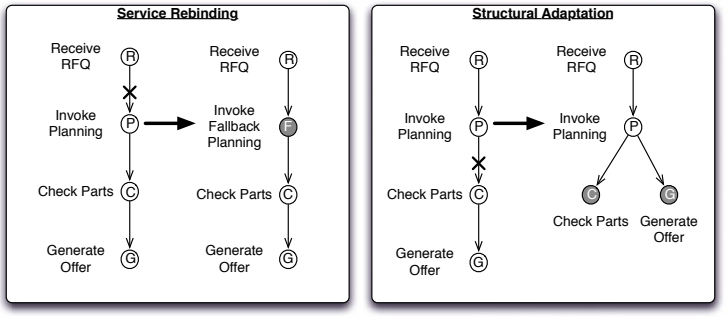

Fig. 1. Types of Adaptation

3.1 Timely Optimization and Stale Results

Figure 2 showcases the timing of asynchronous optimization. For an instance of the composition, an optimization is triggered at time t_0 (e.g., because SLO estimation mechanisms have predicted that this instance is going to violate its SLA). A meta-heuristic optimization algorithm starts searching the solution space. Meanwhile, the service composition continues executing. Firstly, assume that at time t_1 the optimization has converged and delivers the result that two adaptations have to be applied. For both actions the affected region has not yet been reached, i.e., the optimization was timely and the result is useful. However, if we assume now that the algorithm takes more time and delivers its result at time t_2. Now, the activity "Invoke Planning" (`P`) has already been executed, and part of the result (the decision to adapt `P`) came too late.

Intuitively, for every adaptation x with a defined affected region there is also a decision point t_d^x, the latest time in the execution of the composition when a decision needs to be made. Assuming that we know t_a^x, and the time that the application of the adaptation technically takes (d_x), we can define the decision

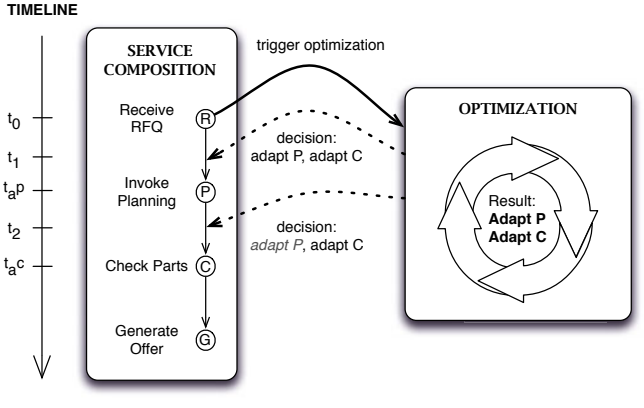

Fig. 2. Optimization Timing

point of an adaptation as $t_d^x = t_a^x - d_x$. We refer to an optimization result A^* produced at time t as stale if $\exists a \in A^* : t_d^a < t$. Stale optimization results cannot be applied in full anymore when they are available, and evidently should be avoided.

3.2 Stepwise Optimization

Two approaches can be used to handle the problem of stale results. Firstly, one can decide not to deal with the problem at all, ignoring stale adaptations and applying only what is still possible when the result becomes available. This approach is very simple, and even in the worst case this is at least never worse than not doing optimization to begin with, even if the result may be suboptimal in the presence of stale results. Secondly, one can drop the idea of asynchronous optimization and halt the service composition while the optimization is running. This trivially prevents stale results, but severely degrades the performance of the service composition. It is well possible (if the optimization takes more time than what can be gained using adaptation) that using this approach is actually worse than not doing any optimization at all. It is easy to see that both of these ideas are not optimal.

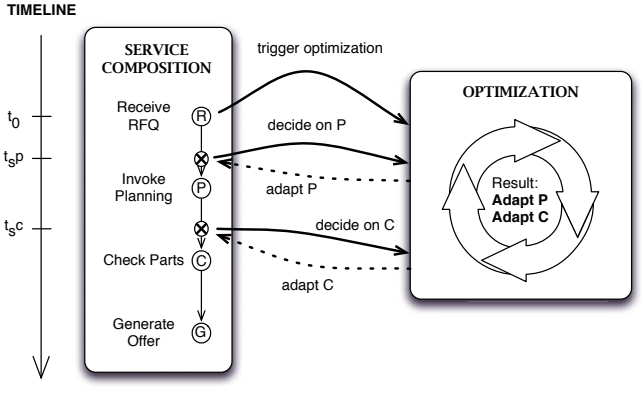

Fig. 3. Stepwise Asynchronous Optimization

Hence, we now introduce stepwise asynchronous optimization as an alternative principle to prevent stale results. The general approach is sketched in Figure 3. First of all, we order all adaptations according to their decision points. Actions with identical decision points ($t_d^x = t_d^y$) are collected in decision sets (D_i). Let t_s^x be the decision point of a decision set D_x, defined as the decision point of all adaptations contained in the set ($\forall a \in D_i : t_d^a = t_s^i$). In the figure, two decision sets with decision points t_s^p and t_s^c exist. The first decision set contains only the action "Invoke Planning" (P), the second contains only the action "Check Parts" (C).

As before, an optimization is triggered at t_0. However, results are now delivered differently as in the naïve approach sketched before. Instead of waiting for

```
1  decide(DecisionSet ds, Optimization opt):
2
3    suspend_process()
4
5    foreach(Adaptation a in ds):
6      decision = decide_on_adaptation(a, opt)
7      if(decision == true)
8        apply_adaptation(a)
9      adapt_target_function(a, decision, opt)
10
11   resume_process()
```

Fig. 4. Decision Procedure for Stepwise Optimization

the optimization to converge, we now trigger the decision procedure sketched in the algorithm in Figure 4 when t_s^p and t_s^c are passed.

For every decision point associated with a decision set, we briefly halt the composition and decide on all adaptations associated with this set. If it is decided to apply one or more adaptations, we do so. Finally, we resume the composition. This way, instead of producing the solution for the optimization problem in one big bang (and risking that the result arrives too late), we use a stepwise, constructive approach which defers all decisions about adaptations to the latest possible time, but never later. Hence, results are never stale, and the service composition needs to be halted only briefly. Note that it is generally advantageous to wait as long as possible before making a decision (per definition, that means waiting until t_d^x), under the assumption that the quality of a decision is monotonically increasing with optimization time. This is true for most implementations of optimization algorithms, including GA (if elitism is used).

The decision algorithm in Figure 4 contains three separate challenges. Firstly, one needs to be able to apply adaptations at runtime (Line 8). We do not discuss solutions for this problem here, and refer the reader to existing work instead [4]. Secondly, we need to be able to actually make a decision on a single adaptation based on a still ongoing optimization (Line 6). One simple, yet promising, strategy is to base the decision on the best currently known intermediary result, i.e., decide to apply an adaptation if and only if the currently best known solution applies the adaptation. We refer to this strategy as current-optimum based decision. Thirdly, after deciding on an adaptation, the target function of the optimization problem needs to be modified. This is to reflect the fact that whenever a decision is made (and a given adaptation is applied or rejected) the underlying problem of the ongoing optimization has in fact changed. If the adaptation has been applied, all solutions that do not use this adaptation are invalid. Similarly, if the adaptation has not been applied, all solutions using the adaptation are invalid. This change is easily represented using an additional penalty term v_x in the target function. For instance, if a_x is applied, a new penalty term v_x defined as $v_x(A^*) = \infty$ if $a_x \notin A^*$, and $v_x(A^*) = 0$ otherwise, is added to the

target function. Hence, in Line 9 of the algorithm, we pause the optimization algorithm, add an additional constraint for each adaptation in the decision set and resume optimizing.

4 Related Work

The general idea of optimizing running composition instances is related to the larger research field of QoS-aware service composition. QoS-aware service composition is usually a static process, which aims at finding the best instantiation of an abstract composition before or during deployment. The optimization problem is finding the most suitable combination of concrete services. The seminal work that introduced this idea already dates back to 2004 [10], however, newer research is still able to provide new insights. For instance, [11] defined a domain-specific language from which dynamic QoS-optimized compositions are generated. [12] improved on the methods of optimization that are used in QoS-aware composition, and proposed to use a combination of local selection and global optimization. The approach presented in this paper differs from all these contributions in that we do not consider optimization statically. Indeed, traditional QoS-aware composition does not face the problem of timeliness at all, as optimization is done once and is not repeated for every problematic instance, as it is the case in our approach. The research work most closely related to this paper is work on runtime adaptation of service compositions. First ideas on self-adaptive compositions can be found in [13], even if this work is more closely related to adaptive workflows than service compositions. Recently, work in this area seems to have gravitated towards using the aspect-oriented programming (AOP) paradigm to technically implement adaptation, as exemplified by [9,14]. Other approaches use pure service rebinding [8] or parametrization of compositions [15]. The PREvent framework [2] supports adaptation on many different levels, and forms the basis of the research work presented in this paper. Other research of note in the area of adaptation include [3], which uses machine learning techniques similar to the mechanisms used in PREvent to trigger adaptation. All of these approaches have a similar base premise (optimization of the performance of a service composition through monitoring and runtime adaptaption), but none discusses timing aspects explicitly. We argue that our work is complementary to all these approaches, and similar ideas as discussed here may be worthwhile to incorporate in any framework that aims at optimizing running composition instances.

5 Conclusions

In this paper we have introduced the problem of stale results in the optimization of service compositions, and proposed stepwise optimization as a possible solution to tackle this issue. As future work, we plan to investigate how appropriate stepwise optimization is for usage with different optimization algorithms, e.g., Ant Colony Optimization or Simulated Annealing, and compare the stepwise optimization approach with quick construction heuristics.

References

1. Dustdar, S., Schreiner, W.: A Survey on Web Services Composition. International Journal of Web and Grid Services 1, 1–30 (2005)
2. Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: Monitoring, Prediction and Prevention of SLA Violations in Composite Services. In: ICWS 2010, pp. 369–376 (2010)
3. Kazhamiakin, R., Wetzstein, B., Karastoyanova, D., Pistore, M., Leymann, F.: Adaptation of Service-Based Applications Based on Process Quality Factor Analysis. In: MONA+, pp. 395–404 (2009)
4. Leitner, P., Hummer, W., Dustdar, S.: Cost-Based Optimization of Service Compositions. Technical Report TUV-1841-2011-1, Vienna University of Technology, Austria (2011)
5. Leitner, P., Wetzstein, B., Rosenberg, F., Michlmayr, A., Dustdar, S., Leymann, F.: Runtime Prediction of Service Level Agreement Violations for Composite Services. In: NFPSLAM-SOC 2009, pp. 176–186 (2009)
6. Zeng, L., Lingenfelder, C., Lei, H., Chang, H.: Event-Driven Quality of Service Prediction. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 147–161. Springer, Heidelberg (2008)
7. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, Reading (1989)
8. Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B., Plebani, P.: PAWS: A Framework for Executing Adaptive Web-Service Processes. IEEE Software 24, 39–46 (2007)
9. Charfi, A., Mezini, M.: AO4BPEL: An Aspect-oriented Extension to BPEL. World Wide Web 10, 309–344 (2007)
10. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering 30, 311–327 (2004)
11. Rosenberg, F., Celikovic, P., Michlmayr, A., Leitner, P., Dustdar, S.: An End-to-End Approach for QoS-Aware Service Composition. In: EDOC 2009, pp. 151–160 (2009)
12. Alrifai, M., Risse, T.: Combining Global Optimization With Local Selection for Efficient QoS-Aware Service Composition. In: Proceedings of the 18th International Conference on World Wide Web (WWW 2009), pp. 881–890 (2009)
13. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.: Adaptive and Dynamic Service Composition in eFlow. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 13–31. Springer, Heidelberg (2000)
14. Leitner, P., Wetzstein, B., Karastoyanova, D., Hummer, W., Dustdar, S., Leymann, F.: Preventing SLA Violations in Service Compositions Using Aspect-Based Fragment Substitution. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 365–380. Springer, Heidelberg (2010)
15. Karastoyanova, D., Leymann, F., Nitzsche, J., Wetzstein, B., Wutke, D.: Parameterized BPEL Processes: Concepts and Implementation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 471–476. Springer, Heidelberg (2006)