# Composition and undesired web service execution effects

Peter Bartalos and Mária Bieliková
Institute of Informatics and Software Engineering, Faculty of Informatics
and Information Technologies, Slovak University of Technology in Bratislava
{bartalos,bielik}@fiit.stuba.sk

***Abstract*—Automatic dynamic web service composition is showing to be an effective way how to deal with the dynamic character of the web services and business environment, while providing a mechanism supplying varying user goals. This paper briefly introduces the undesired service execution effects influencing the service composition and achievement of the user goal. We discuss a solution eliminating the undesired effects and show how it fits into the overall service composition process.**

## I. Introduction

Web service composition [1], [2], [3], [4] is a process of arranging several web services into workflows to provide a utility, which is not offered by any single service. To supply varying user goals, dynamic composition is necessary. In this case a lot of automation is in demand to handle the variability effectively. A lot of work had already been done in the field of automation of service composition. The automation brought additional requirements to the whole life-cycle of services and the composition approaches. On the other side, it provides features which are impossible to achieve in manual manner. There are several areas where web services and service composition are applied [5], [6].

The automatic dynamic web service composition problem is defined as follows: given a query describing the goal and providing some inputs, design a workflow (depicting the data- and control- flow) from available services, such that its execution produces the required goal. Even that there are plenty of composition approaches, each based on various methods, the automatic composition can be generalized to the following steps:

1) *Design of abstract composite service*: during this step we identify which services could be potentially chained, i.e. which service produces some results required by another services, or in the goal. Here, we consider each service with the same interface definition as one. At this point, it is neglected how many implementations exist, or at how many places is one implementation deployed. Only the functional properties of services are important.
2) *Finding the best candidate service*: at this step, for each service in the abstract composition, we look for the best candidate service which is used during the execution. The word *best* in this case means that the selected service has better attributes, which the user is interested in, than any other candidate service. These attributes relate to the non-functional properties of services.
3) *Execution of the final composition*: to definitely achieve the satisfaction of the user goal, the composite service is executed to produce the required results.

Note that the presented steps of automatic composition of services do not fit all over. In some approaches, the steps may interleave, or the overall process shows slightly differences. Beside this, concrete works usually deal only with sub-problems related to some parts of the whole process. This paper focuses on problems which may occur due undesired web service execution effects. These cause that the user goal is not satisfied, or unwanted side-effects are made. We discuss a usage of *counterpart services* to avoid execution of services which do not led to the desired results.

## II. Undesired service execution effects

Web services allow invocation of a remote software system, available through the web, using standard protocols. In general, there is no limitation to what kind of process is encapsulated in the web service. It can be a very simple procedure, but also a very complex business process. It can expose a part of a software system, which is this way accessible to the web users. Web services present a certain level of unpredictability related to the result of their execution. These affect the fulfillment of the user desires, according the web service. It is caused by the following issues:

- execution errors,
- unpredictable output values,
- unpredictable post-conditions.

The execution of a web service may fail due several technical reasons on the service provider, or customer side, or the communication link. These errors are usually fatal and make the service invocation unsuccessful. The execution errors are undesired for the user. Web services produce data of such type as defined in its description. However, the actual value is not known. In some cases, the value of the outputs may affect the user satisfaction. This is true also regarding the post-conditions. They may define several alternatives. It is not known, which particular alternative condition, will hold after the service execution. Since the user may be interested only in some of them, it is again not guarantied that the service will satisfy the user needs. Moreover, in some situations, the execution may cause world-altering side effects, which are undesired by the user and require some kind of compensation.

The web service composition is based on the service descriptions, i.e. information known when the service is deployed. However, as we just discussed, these information are (naturally) not enough. Based on these, we can find a composition having a potential to fulfill the user goal, but we cannot be sure that it will do so. When looking for a service realizing a particular task in the composite service, the composition should find all services having the potential to accomplish the required task. This means that each service producing the required outputs and whose post-condition includes a goal condition, should be candidates to realize a given task. The selection of the best candidate service usually requires additional information, which are not known from the service description. The execution of the service could be required to acquire them.

Consider a service realizing a hotel reservation as follows. Taking a hotel identifier, a defined time interval, and credit card information, it creates a reservation of the hotel for the specified time period and returns a reservation confirmation including the information regarding the room price and other details. The reservation could also fail, for example because there is no free room for the required time period. In this case the service returns a notification about the failure. These two alternatives are precisely described in the post-condition of the service, depicting two alternatives of the execution result.

When looking for a hotel reservation service, our example service should be one candidate. However, the service cannot guarantee that it will successfully make a hotel reservation. This depends on the actual hotel occupation rate. This fact is known in the reality, but the service requestor is unable to realize it, until the service invocation takes place. In the case when the reservation cannot be realized, the invocation of the service is not useful. A positive issue is that the execution of the service at least did not make any undesired changes of the world. Sometimes, this is not true.

Consider a situation that it is possible to reserve a hotel for a desired term, but we have additional requirement that the hotel price is no more than $500$ €. Our example service is again capable to achieve this goal. However, in the case that the hotel price exceeds $500$ €, the user goal is not fulfilled. Moreover, in this case, the execution of the service has an undesired, world-altering effect, because it made a hotel reservation, and charged money from the credit card. If we know that the price of the hotel exceeds $500$ €, we would rather try to find another hotel and do not invoke the service with the expensive hotel. The key point is how could we know the price of the hotel ahead of service execution time. The price, or the availability of the hotel can be seen as *critical information*, based on which it is decided if it is, or is not desired to invoke the given service.

If the only way to get the hotel price is to invoke the hotel reservation service, we have to deal with undesired, world-altering effects. In some cases these can be eliminated by executing a kind of undo operation. On the other side, some service executions are non-reversible and cause permanent effects.

Similar problems as just discussed were already being addressed [7], [8], [9], [10]. These works deal with web services transactions. Transactional behavior of web services is required to be able to manage the execution of multiple interrelated services, composed together to realize a complex task. If any of these services cannot complete the desired sub-task, the whole task fails. In this case, it is desired to leave the world in a state as before the execution of a transactional web service composition, i.e. as we never done it. To achieve this, similar behavior as presented in database management systems regarding transactions is required. To undo some actions, compensation must be supported also in the case of web services and their compositions. The problem is that not all services support compensation. Moreover, even if compensation is possible, rolling back a previously completed transaction could be expensive, or ineffective.

An approach, trying to minimize the need of transaction cancelation, or compensation is presented in [7], [8]. These works use a concept of *tentative holding* in web service composition. This concept had been implemented in a *tentative hold protocol*[1]. Its aim is the exchange of information across businesses prior to an actual transaction. The exchange of information tends to prevent the need of transaction cancelation, which can occur in dynamic business environment. The business parties make tentative commitments related to terms of an interaction (e.g. price, quantity). The protocol allows multiple clients to place holds on the same items (e.g. hotel reservation). Thus the hold is non-blocking. Whenever one client completes for example the purchase, the other clients receive notifications that their holds on the same item are no longer valid. Based on these notifications, the other clients know that they cannot finish the purchase. This allows them to try to accomplish the user goal an alternative way and there is no need of transaction compensation.

In [7], three phases of business transactions are described:

- *Pre-transaction phase*. During this phase the critical information are exchanged between partners.
- *Main-transaction phase*. In this phase, the main business process execution takes place.
- *Post-transaction phase*. This phase is responsible for observing the agreements and terms specified during the execution between the transacting parties.

Splitting the transaction into phases has important consequences. Taking our example, we first exchange the information about the hotel price. Based on this information, we decide if it is desired to continue with the reservation process. The decision is based on the user satisfaction degree with the hotel price. This way we can avoid undesired effects, and it is not needed to deal with undoing them, i.e. no compensation operation is required. This preemptive approach is much more feasible in the case of web service compositions realizing certain business processes.

The realization of *preemptive service composition*, aiming to avoid execution of services not producing the desired results,

[1] http://www.w3.org/TR/tenthold-1/

should be supported in several points:

- *service design*,
- *world-altering effects identification*,
- *critical information identification*,
- *critical information exchange*.

Web service composition aims at combining existing services in such a way that the execution of the composite service produces the required results. The single web services are atomic and cannot be changed on the demand of the actual composition. As far as we know, there is no methodology, describing how should be designed the web services aimed to be automatically composed. It is quite obvious that the design of the web service significantly affects the ability to use it during service composition. Taking our hotel reservation example, it is much suitable to split the whole process into several web service operations. The very important point is to provide separate operations, which can be used to get the information about the possible reservations. This should be realized without a need of booking a hotel. If the services are designed to provide a properly atomic functionality, the automatic composition can bring significant advantages. These arise from the ability to effectively select the best service candidate for each user. During this, the current situation regarding the service environment and the user context are considered.

A proper design of single services can ease the solving of the problem as proposed in our example. The basic principle is to provide a way how critical information could be obtained by the service requestor, without the need to execute a world-altering service. Already during the design time, it should be identified if the service produces a world-altering effect and based on which information could it be decided, if this effect is desired. Consecutively, the access to the critical information must be supplied. This may be realized in several ways. The most feasible way is by developing web services, which stand as counterparts to the main service, realizing the business process, with world-altering effects. Note that there may be multiple critical information related to one service. Thus, multiple counterpart services could be related to one world-altering service. Considering our example, there must be services which may be used to obtain information about the hotel availability, its price, and other information about the hotel reservation conditions, see Fig. 1. The crucial issue is that the counterpart services can be invoked to provide the information about the same business entity (e.g. hotel reservation), which is affected by the main service, realizing the actual business activity. For example, if we get the hotel price for the specified conditions (e.g. time period, room type), the same price will hold if we make a reservation with the same condition specification.

To allow preemptive service composition, the *counterpart interrelationships* between services must be explicitly stated in the service descriptions. Afterward, the composition is enhanced based on the following principle. If a world-altering service is required in the composition, the critical information is exchanged first, by invoking the counterpart services. By invoking the counterpart services, we gather information determining the ability of a particular service to satisfy our needs and are able to evaluate the level of satisfaction. Based on this, if we realize that the service cannot fulfill our needs, we do not consider it any further. From the suiting services, we choose the one fulfilling our needs the best.

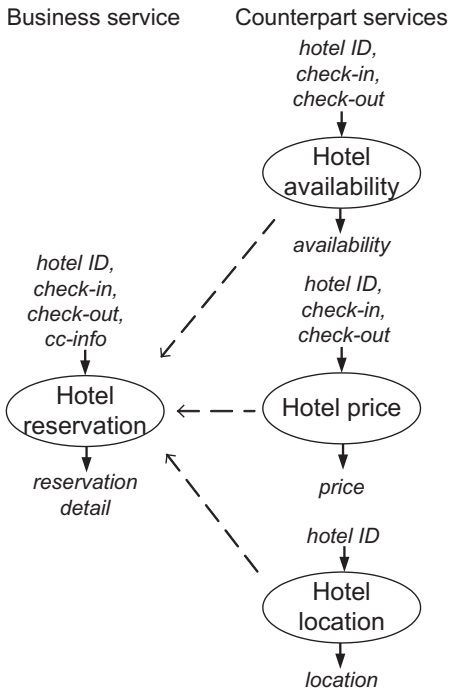


Fig. 1. Counterpart services to a business service.

The overall service composition, incorporating the methods solving different parts of the composition problem, should follow the following steps, see Fig. 2:

1) *Abstract composition design*: search a composition, consisting from abstract services, based on the functional requirements.
2) *Find concrete services*: for each task in the abstract service composition, find all concrete services fulfilling the given task.
3) *Gather information*: gather all information necessary to evaluate the degree of user goal satisfaction by the possible service compositions. This includes invocation of counterpart services to exchange critical information. No service with world-altering effect is executed.
4) *Select optimal concrete services*: Based on the information about the concrete services, gathered in the previous step, or known from service descriptions (SLA), select the optimal concrete services bringing the maximal user goal satisfaction. During optimization, the non-functional properties of services are considered. This includes the QoS characteristics and user preferences. At this step, the user preferences from the query, or from a user model are took into account. The actual

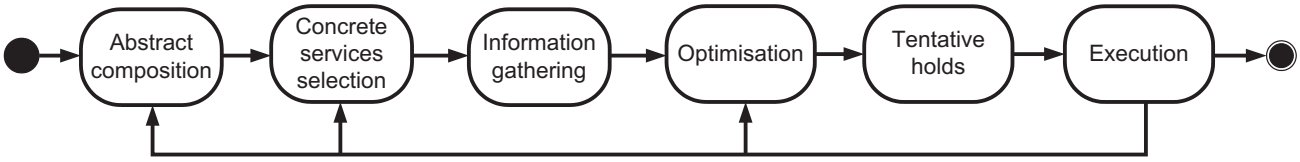


Fig. 2. Web service composition steps.

user context should be considered too. Multiple criteria decision making and constraint satisfaction techniques are used here to get the best possible solution, taking into account all the known requirements.

5) *Tentative hold*: when it is known which business resources bring the maximal user satisfaction and which services should be used to exploit them, place holds for each resource.
6) *Execute composition*: If no notification is received stating that some holds of resources are no longer valid, execute the concrete services in the required order. If some service permanently fails to execute, or some holds for any resource are no longer valid, update the selection of concrete services. During this, optimize the composition based on the current situation. If the problem requires changing an already executed part of the composition, transaction mechanisms are used to recover from the undesired state. If it is required, go back to previous steps to adapt the composition.

The most critical phase of the overall composition process is the execution. One reason is that the former steps could be realized relatively quickly. Thus, the changes in the environment are not expected and the final composition is the best considering the current situation, i.e. if it could be executed immediately, then the user satisfaction is guaranteed. However, the execution of some services may be a long-running process. Hence, several changes may happen in the environment. The changes have different character and require different management. If the change causes that a service selected to be a part of the final composition is no longer available, or does already not guarantee the required non-functional properties, this service should be replaced with another one. In the case of tremendous changes in the service environment, e.g. services with new functionality are deployed, it is desired to perform again the abstract composition design.

## III. Conclusion

Automatic dynamic web service composition is showing to be an effective way how to deal with the dynamic character of the web service, and business environment, while providing a mechanism capable to supply varying user goals. Several research results concerning different aspects of the overall problem had been proposed, and they present promising results. In some specific scenarios, the current results make the composition practically applicable.

To exploit more potential of the service composition, several problems remain to be solved. More attention should be given to those issues which are not related to the actual arrangement of services into a composite one. The whole life-cycle of web services and their composition should be covered. More interest is required to the design of web services. The design decisions significantly affect the further possibility to compose the developed services. Semantic annotation of web services is also a subject of further research. Methodologies and tools should be developed to support this process. It should be analyzed if the description focusing on the I/O and pre-/post-conditions is sufficient. One potential area to enhance the annotations is to provide information about which world-altering effects may be caused by the service, and identification of the critical information. These together with a precise design of business services and counterpart services could help improving the composition to achieve higher user satisfaction.

## Acknowledgment

This work was supported by the Scientific Grant Agency of SR, grant No. VG1/0508/09 and it is a partial result of the Research & Development Operational Program for the project Support of Center of Excellence for Smart Technologies, Systems and Services II, ITMS 26240120029, co-funded by ERDF.

## References

[1] S. Dustdar and W. Schreiner, "A survey on web services composition," *IJWGS*, vol. 1, no. 1, pp. 1–30, 2005.
[2] P. Bartalos and M. Bielikova, "QoS aware semantic web service composition approach considering pre/postconditions," in *Int. Conf. on Web Services 2010*. IEEE CS, 2010, pp. 345–352.
[3] ——, "Effective QoS aware web service composition in dynamic environment," in *Int. Conf. on Information Systems Development 2010*. Springer, 2010.
[4] S. Kona, A. Bansal, M. B. Blake, and G. Gupta, "Generalized semantics-based service composition," in *ICWS '08: Proc. of the 2008 IEEE Int. Conf. on Web Services*. IEEE CS, 2008, pp. 219–227.
[5] P. Bednar, K. Furdik, M. Paralic, T. Sabol, and M. Skokan, "Semantic integration of government services - the access-egov approach," in *IIMC '08: EChallenges e-2008*, 2008.
[6] M. Sarnovsky and M. Paralic, "Text mining workflows construction with support of ontologies," in *SAMI '08: 6th Int. Symposium on Applied Machine Intelligence and Informatics*, 2008, pp. 173–177.
[7] M. P. Papazoglou, "Web services and business transactions," *World Wide Web*, vol. 6, no. 1, pp. 49–91, 2003.
[8] B. Limthanmaphon and Y. Zhang, "Web service composition transaction management," in *ADC '04: Proceedings of the 15th Australasian database conference*. Darlinghurst, Australia: Australian Computer Society, Inc., 2004, pp. 171–179.
[9] J. E. Haddad, M. Manouvrier, and M. Rukoz, "Tqos: Transactional and qos-aware selection algorithm for automatic web service composition," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, pp. 73–85, 2010.
[10] B. A. Schmit and S. Dustdar, "Towards transactional web services," in *CECW '05: Proceedings of the Seventh IEEE International Conference on E-Commerce Technology Workshops*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 12–20.