Self-Healing Web Service Compositions

Sam Guinea
DEI - Politecnico di Milano
via Ponzio 34/5
Milan, Italy
guinea@elet.polimi.it

Advisor: Carlo Ghezzi - carlo.ghezzi@polimi.it

Research Area: Distributed computing Service Oriented Computing Web Service Compositions

Research Topic: Three research directions for obtaining trustworthiness of SOAs through use of self-healing web service compositions are presented: service and process description, monitoring and recovery strategies.

1. INTRODUCTION

SOAs (Service Oriented Architectures) are slowly becoming a new paradigm for flexible coordination between business partners within continuously evolving environments. SOAs use service technologies —such as Web Services— to achieve business goals in which collaboration with external parties is compulsory and interoperability issues arise constantly. The context of our analysis is, by definition, a very rapidly evolving environment; an environment in which the set of available services with which to do business changes constantly in number and type. The availability of a Web Service may vary for a number of reasons: it can be deployed or undeployed at any given time, it can change its specification or implementation, making it useless for the goal that a system sets out to achieve, or it can change its provided quality of service (QOS). This can hinder collaboration between services; the ephemerality of the set of players can easily cause a backlash on the trustworthiness perceived by the clients of the collaboration. The client's trust is placed in fact in the provider of the composed service and not in the services the composition uses.

In my research, focus will be placed on the collaboration that can be set up between organizations that choose web services are their guiding technology, and how to obtain trustworthiness through self-healing compositions.

Collaboration between services is enabled through the use of a series of standards for the definition of service compositions that provide business logics, typically in the form of workflow-like coordinations. BPEL4WS (Business Process Execution Language for Web Services) [1] is becoming the main player thanks to its industrial endorsement and adoption. This is the reason why it represents, for the time being, our composition technology of choice.

In the life-cycle of a service composition three different phases can be pointed out : design-time, deployment-time and run-time. To produce a sound composition that can be trusted, all three phases are of vital importance. These phases can also be found in component based architectures. The nature of SOAs, however, is profoundly different from that of component based architectures. In service oriented architectures, it is possible to design abstract processes and to postpone the binding to specific services until deployment-time or even run-time. Even after the necessary services have been bound to the engine executing the composition, we can envisage situations where the services are replaced on-the-fly in order to achieve better quality of service or to react to an erroneous behavior where a bound service changes or becomes unavailable. For these reasons, pre-deployment verification of composite services is not enough; correctness verification needs to be performed as part of the run-time phase of a composition.

Once an erroneous behavior is found during the execution of a process, recovery actions should be pursued. Simply shutting down the execution is not an option, since the trust placed in the system by the clients would suffer greatly. Possible recovery strategies can go from simply trying to re-invoke the bound service, all the way to dynamically re-organizing the composition.

Monitoring and recovery are two very important issues in SOAs because they allow to react on-the-fly to arising erroneous behaviors, possibly continuing to provide service to the customers.

Several research efforts are active in this area, even though most are still in an initial stage. Many different approaches for monitoring and for dynamic binding of services –necessary for implementing recovery strategies— have been proposed, each originating in different areas of interest such as Artificial Intelligence, Semantic Web, Software Engineering, etc.

The rest of this paper is structured as follows. Section 2 will introduce the goals of my research and underline the different dimensions on which I believe the problem must be confronted. I will also illustrate the initial goals that have already been met and the issues that remain open. Section 3 will present related work, while Section 4 will provide some conclusions.

2. RESEARCH CLAIMS

My research is based on two firm assumptions. First, I will develop solutions that can be implemented on top of existing (or emerging) technology. I will not try to re-invent the wheel, but stick to standard or standard de-facto technology with the intent of guaranteeing greater adoptability. In particular, I am presently using the BPEL language to describe composite services.

Second, in order to provide solutions to the monitoring problem and to the dynamic reconfiguration of service compositions, without straying from standard approaches, it is necessary to introduce overhead to the process design in the form of additional BPEL code. In my approach, I decided the business logic code should remain as separate as possible from the monitoring and recovery code added to provide sound compositions. This separation of concerns is useful to simplify design and achieve maintainability.

To provide service compositions capable of self-healing in the wake of erroneous behaviors, many hurdles will have to be surpassed. There are three dimensions along which research must move, in order to achieve the aforementioned goals:

- Service and process description
- Monitoring
- Recovery strategies

I will now cover briefly each one and emphasize the particular difficulties that arise and how I believe they can be solved.

2.1 Service and process description

This area must provide the common grounds on which monitoring and recovery are to be achieved. Simply using existing languages like WSDL [2] is not enough, since it only provides a syntactical description of the operations provided by a service. Monitoring has to do with verifying that a service actually provides what it says it provides, both from a functional and a non-functional point of view. Recovery, on the other hand, has to do with taking action when something goes wrong, with the intent of completing the process in a satisfactory way for its client. Semantics play a decisive role in both areas. In our approach, semantics is given to services by means of contracts, as in design by contract [5]. For each operation provided by a service, pre- and post-conditions are defined. By looking at these pre- and post-conditions it is possible to reason on how services must be used within certain scenarios.

Using contracts can be very useful to designers, but the level of granularity is too low for it to be of any real use to the clients of the service. We advocate that a high level requirements language must also exist, a language with which clients of the composed process can express what they expect of it and, maybe more importantly, what should never happen during execution. By providing such a language and by exploiting the lower level semantics given to the different steps of the process, it should be possible to provide a semi-automatic transformation of the higher-level requirements into lower-level process specifications. This transformation would consist in a conjunction of the contract defined by the service provider and the contract derivable from what specified by the client of the process. The way this transformation can be defined and supported, at the moment, is an open issue in my research. Only lower-level requirements —contracts given by the service provider— have been considered and exploited so far.

2.2 Monitoring

Monitoring consists of verifying at run-time that the requirements, specified by the clients of a composition and later mapped onto lower-level process requirements and by the service providers, are met during execution. The requirements can obviously be of very diverse nature. We foresee three complementary dimensions to the monitoring problem coexisting. We shall call them:

- Assertion based monitoring
- Event-based monitoring
- History-based monitoring

Assertion-based monitoring has been the main concern of my initial research. It consists of asking the BPEL process to use an external monitor service to verify the correctness of certain assertions at given points of the process execution. The assertions that must be monitored are the pre- and post-conditions derived from the conjunction of those provided by the service provider and those provided by the client of the process. In our current implementation of the assertion-based monitoring mechanism only contracts defined by the service provider have been considered. We will now briefly provide an overview of the approach.

Starting from a complete BPEL process —the *unmonitored process*— assertions are added to the process in the form of commented annotations. The location in which they are added indicates where in a process an assertion must be checked. Once an annotated version of the unmonitored process is available, it is passed through a transformation algorithm called BPEL2BPEL, which produces what we call the *monitored process*. The algorithm adds BPEL code to the process to support the use of an external monitor service to check the validity of the assertions. The added code prepares the message to be sent to the external monitor, sends it, and verifies the monitor's answer. The message that is sent to the external monitor service contains two items: the assertion to be checked and the data on which the assertions must be checked. These data typically consist of information about the internal state of the process in execution. Depending on the implementation of the external monitor, the data on which to verify the assertions might also be obtained elsewhere[1]. This approach obviously has a substantial impact on performance, since the process execution is momentarily stopped to execute the monitoring. However, it is simply a technique that is offered to the process designer, which can be used as needed. For example, it might be used only sporadically or even be switched off totally or partially at any time. Its strength is that, since the monitoring is inserted in the form of annotations, the business logic remains separate from the monitoring logic up until when the BPEL2BPEL

[1]For more information regarding obtaining data outside of the process, please refer to [8].

performs the transformation that weaves the two to produce a new version of the monitored process. The original version is not touched. Second, the resulting process remains pure BPEL code and is executable on any standard BPEL engine. My initial research has already obtained some results in this direction and is currently investigating dynamic AOP approaches to do the weaving of the business logic and the monitoring logic. The inter-weaving should be able to take into account the different monitoring necessities of different stake-holders and/or different times of the life-cycle of the process. We have noticed how, with this approach, it is easier to monitor functional contracts, with respect to non-functional contracts.

Event-based monitoring comes into play when it is necessary to verify non-functional qualities of a service or of a composition. It consists of a less-intrusive approach to monitoring where, in parallel to the execution of a business process, a monitoring component can listen to the events launched by the BPEL engine. Since this is done in parallel to the execution, almost no impact on performance is expected. Obviously, this approach is closely tied to the particular implementation of the BPEL engine in use. For example, using ActiveBPEL [4], an open source BPEL engine, it is possible to listen to a series of events that the engine provides. In particular, these events are tied to the standard BPEL activities (i.e. invoke, receive, etc.) and to their state changes. Each activity can be, for example, in a *Ready to execute* state, in an *Executing* state, in a *Terminated* state or in a *Faulted* state. By listening to the events, it is possible to reason on some non-functional qualities expressed in terms of LTL formulas. Another possible approach would be to position an observer service between the BPEL engine and the outside world and to monitor the contents of the SOAP [3] messages flowing in and out of the system. Both approaches must be further investigated in order to discover the strengths and weaknesses of each. At the moment, it is clear that both work at a lower-level with respect to the BPEL process and to the assertion-based approach to monitoring. As a consequence, once an erroneous behavior is observed, it is not possible to simply freeze the execution of the process to implement a recovery action, like dynamic reconfiguration of the process. Intervention in the execution could be obtained by selecting points in the process in which to introduce an assertion. This assertion could represent a synchronization point between the process execution and the collection of events, a point in which we assert that all the event-based LTL formulas verified up until then must be correct. By adding an assertion, we would be effectively mixing event-based monitoring with assertion-based monitoring. So, if the assertion is false, at least one of the checked LTL formulas must be false, meaning recovery actions should be taken.

History-based monitoring is an extension to event-based monitoring. By collecting events in a history event repository it is possible to reason upon QOS requirements — possibly expressed in LTL formulas— that deal with a history of process executions. An example of such a requirement could be "eighty percent of the times a process goes into execution it must complete within one minute". The analysis of recovery policies for history-based monitoring are an open issue of my research.

So far, in my initial research, only the assertion-based approach has been studied, and some preliminary yet encouraging results have been achieved. The event-based approach and the history-based approach still have to be studied in depth, although it already seems clear that the three should coexist to obtain complete functional and/or non-functional monitoring.

2.3 Recovery Strategies

Recovery strategies permit a system to continue execution even after erroneous behaviors are discovered. In my research, I foresee three possible recovery strategies:

- Retry
- Substitute
- Restructure

All three recovery strategies need additional BPEL code to be added to the original workflow to make it work. As we saw for monitoring, the additional code can be added to the process by means of annotations and BPEL2BPEL transformation or by means of AOP weaving of BPEL code. For each strategy we shall briefly describe the code to be added.
The first recovery strategy —*Retry*— simply tries to re-execute the service that is causing the problem. The BPEL code that must be added to the process, in this case, introduces exception handling and repetition.
In the second recovery strategy —*Substitute*— we simply try to bind the execution to another service, capable of providing the same functionality of the faulty service and acceptable non-functional qualities. The equivalence between the two service n_1 and n_2, is given in terms of implications between contracts. In particular, if n_1 must be substituted by n_2,the pre-condition of n_1 must imply that of n_2 and the post-condition of n_2 must imply that of n_1. In this case, the BPEL code to be added consists of the activities needed to invoke an external broker service, capable of dynamically looking-up and binding to an suitable service.
The third recovery strategy —*Restructure*— consists of reorganizing the process locally in order to introduce the invocation of a collection of services that, once composed correctly, provide the same functionality of the faulty service and acceptable non-functional qualities. For example, if the service n is to be substituted by the invocation sequence n_1, n_2 then the pre-condition of n must imply the pre-condition of n_1, the post-condition of n_2 must imply the post-condition of n and the post-condition of n_1 must imply the pre-condition of n_2. The BPEL code that must be added consists of the activities needed to invoke an external broker service, capable of finding the necessary services and combining them correctly into a secondary BPEL process that provides the same functionality of the faulty service and acceptable non-functional qualities. The main difficulty consists of finding the correct services and how to combine them. We propose to construct a lattice in which the known services are connected by implication relationships existing

between pre- and post-conditions. By reasoning on this lattice, it should be possible to reduce the dynamic composition problem to a graph problem, a problem in which we must decide if a path, that reaches a node in which the post-conditions of the service that has to be substituted are implied, exists.
Notice that recovery strategies can be constructed hierarchically. If one strategy does not work, a second one can be tried subsequently. For example, one could first retry service invocation after failure. If the failure persists, one could try to find a substitute or, if none is found, one could try to compose a substitute service out of existing elementary services. These three strategies are easier to understand when used to recover from functional errors. How these strategies, or others, can be used to recover from non-functional problems is part of my on-going work.

3. RELATED WORK

In this field, a considerable amount of research has been done on the semantic description of web services. Of paramount scope are the solutions based on ontologies and in particular the Meteor-S Web Service Annotation Framework project [9]. This framework provides a semi-automatic annotation of web services that is based on ontologies. In particular, WSDL files are annotated and matched to domain ontologies. Another approach, proposed by Hamadi et al. [10], semantically describes services and compositions by means of Petri-nets.
Regarding service execution monitoring, we recall the Cremona (Creation and monitoring of WS-Agreements) project [12]. WS-Agreement is a standardization effort of the Global Grid Forum that defines an agreement protocol based on XML. This standard defines agreements for interfaces, security and quality of service properties. Cremona provides a framework that simplifies the definition, the management and the run-time monitoring of the state of the agreements.
Another approach to the monitoring problem has been proposed by Spanoudakis et al. [11]. This is an event-based approach in which the properties that must be monitored are expressed in event calculus. The approach also proposes a semi-automatic extraction of these properties from the BPEL definition of the service composition to be monitored.
In the field of dynamic composition of web services, WSCG (Web Service Composition Graph) [7] proposes a framework that provides visual design, validation and development of compositions using graph theories. In another approach, Pernici et al. [6] propose a framework that uses a proprietary service description and dynamically orchestrates the composed services.

4. CONCLUSIONS

Here I have presented a possible roadmap for research finalized to obtaining self-healing service compositions. Three dimensions have been indicated, all necessary in order to achieve the goal: service and process description, monitoring, and recovery strategies. In all three, research has started and some initial goals have been reached successfully. Many issues, however, remain open and will be the focus of my on-going work.
Service and process description will play a decisive role, since it must pave the way for the other two research dimensions. *Monitoring* is fundamental since, in order to perform recovery, one must first know if something goes wrong, as soon as possible. A compromise needs to be found between the amount of delay of recovery from discovery of erroneous behavior and the performance impact due to the monitoring of the process. Finally, the importance of having *recovery strategies* is clear, and intriguing research problems arise in this area.
My initial direction of research consisted of annotating service compositions with functional contracts to be checked by monitoring and recovery strategies to execute when something goes wrong. BPEL2BPEL currently uses two different implementations of external monitors and is capable of adding BPEL code for implementing the *Retry* recovery strategy. A CASE tool for simplifying the annotation processes is almost complete and it will be presented under the form of an Eclipse plugin.

5. REFERENCES

[1] BEA, IBM, Microsoft, SAP and Siebel. Business Process Execution Language for Web Services Version 1.1. 2003.

[2] W3C. Web Services Description Language (WSDL) 1.1. 2001.

[3] W3C. Simple Object Access Protocol (SOAP) Version 1.1. 2000.

[4] ActiveBPEL. ActiveBPEL Engine - the open source BPEL engine. `http://www.activebpel.org`.

[5] B. Meyer. *Object-oriented Software Construction*. Prentice Hall, New York, N.Y., second edition, 1997.

[6] F.P. Pesicce, M. Mecella and B. Pernici. Modeling E-Service Orchestration through Petri Nets. *TES*, 2002, pages 38-47.

[7] B. Jun, Z. Ren and J. Li. A New Web Application Development Methodology: Web Service Composition. *WES*, 2003.

[8] L. Baresi, C. Ghezzi and S. Guinea. Smart Monitors for Composed Services. *ICSOC'04. ACM.*, 2004.

[9] A. Patil, S. Oundhakar, and K. Verma. METEOR-S Web Service Annotation Framework. *WWW 2004*, 2004.

[10] R. Hamasi and B. Benatallah. A Petri net-based model for web service composition. *Proceedings of the Fourteenth Australasian database conference on Database technologies 2003 - Volume 17*, 2003.

[11] K. Mahbub and G. Spanoudakis. A Framework for Requirements Monitoring of Service Based Systems. *ICSOC'04. ACM.*, 2004.

[12] H. Luwig, A. Dan and R. Kerney. Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. *ICSOC'04. ACM.*, 2004.