Towards Adaptive Management of QoS-Aware Service Compositions – Functional Architecture*

Mariusz Momotko[1], Michał Gajewski[1], André Ludwig[2], Ryszard Kowalczyk[3], Marek Kowalkiewicz[4], and Jian Ying Zhang[3]

[1] Rodan Systems S.A., ul. Puławska 465, 02-844 Warsaw, Poland
{Michal.Gajewski, Mariusz.Momotko}@rodan.pl
[2] University of Leipzig, Marschnerstr. 31, 04109 Leipzig, Germany
ludwig@wifa.uni-leipzig.de
[3] Swinburne University of Technology, PO Box 218 Hawthorn, Victoria 3122, Australia
{jyzhang, rkowalczyk}@it.swin.edu.au
[4] Poznan University of Economics, Al. Niepodleglosci 10, 60-967 Poznan, Poland
M.Kowalkiewicz@kie.ae.poznan.pl

Abstract. Service compositions enable users to realize their complex needs as a single request. Despite intensive research, especially in the area of business processes, web services and grids, an open and valid question is still how to manage service compositions in order to satisfy both functional and non-functional requirements as well as adapt to dynamic changes. In this paper we propose an (functional) architecture for adaptive management of QoS-aware service compositions. Comparing to the other existing architectures this one offers two major advantages. Firstly, this architecture supports various execution strategies based on dynamic selection and negotiation of services included in a service composition, contracting based on service level agreements, service enactment with flexible support for exception handling, monitoring of service level objectives, and profiling of execution data. Secondly, the architecture is built on the basis of well know existing standards to communicate and exchange data, which significantly reduces effort to integrate existing solutions and tools from different vendors. A first prototype of this architecture has been implemented within an EU-funded Adaptive Service Grid project.

1 Introduction

Users want to realize their needs as simply as possible and, therefore, look for sophisticated services that would handle compound needs related to their life events or business activities as a single request. One of the most promising approaches for such sophisticated services is to implement them as compositions of other, simpler services (referred further to as atomic services).

* This work is partially supported by the European Union FP6 Integrated Project on Adaptive Services Grid (EU-IST-004617) and the International Science Linkages programme under the Australian Government's innovation statement, Backing Australia's Ability on Adaptive Service Agreement and Process Management project (AU-DEST-CG060081).

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 637 – 649, 2006.

In the last decade, a huge effort has been put into developing solutions targeting management of service composition, especially in the area of web services and grids. As its result, a number of new standards on various aspects of service composition management have been defined. In Web Service Business Process Execution Language (WS-BPEL) OASIS standardised a language to define service compositions. In Web Service Quality Model (WSQM) OASIS also proposed a quality model and a set of quality factors for web services. One of the recent OASIS standards is Web Service Distributed Management (WSDM) which enables management applications to be built using web services, allowing resources to be controlled (and monitored) by many managers through a single interface. Currently, OASIS is working on Web Service Quality Description Language (WS-QDL) which will describe WSQM in a standardised type of XML representation. At the same time, IBM Corporation has defined Web Service Level Agreement (WSLA) – a language to represent service level agreement (SLA) for web services. Recently, GRAAP working group of the Global Grid Forum has prepared a draft version of a WS-Agreement specification which describes domain-independent elements of a simple contracting process, extensible by domain specific elements.

Focusing on adaptive management of service compositions, several intensive research works have been carried out recently. The *eFlow* project at HP labs [3] proposes an adaptive and dynamic approach to manage service compositions focusing on their functional aspects such as dynamic service discovery and ad-hoc changes. The *QUEST* framework ([6]) extends the work done on eFlow introducing quality of service (QoS) provisioning. In that framework, contracting of service compositions and atomic services is done by SLA documents. The MAIS project ([5]) focused on negotiation of web service QoS parameters with the ability to use different negotiation strategies. In the area of SLA-based contracting and monitoring, there are several advanced approaches and frameworks such as those presented in [2] and [9].

Recently, the Adaptive Services Grid (ASG) project proposed a comprehensive approach towards adaptive management of QoS-aware service compositions. This approach integrates well known concepts and techniques and proposes various **execution strategies** (see [8] for further details) based on dynamic selection and negotiation of services included in a service composition, contracting based on service level agreements, service enactment with flexible support for exception handling, monitoring of service level objectives, and profiling of execution data. Due to the various execution strategies, the approach is able to cope with dynamic changes related to the contracted atomic services, re-negotiate a contract in case of QoS constraint violation, and re-select dynamically another atomic service that satisfies QoS constraints. In addition, the approach also considers service profiling and historical execution data and therefore is able to optimise its way of working.

In this paper we define an **(functional) architecture** that can support various execution strategies for service compositions. This architecture includes specification of software components, definition of their public interfaces and detailed description of interaction among them according to the basic tasks common for all execution strategies. Most of the data exchange formats and interaction protocols between components are based on standards as WS-BPEL , WS-Agreement, and WSQM. This approach gives the architecture appropriate flexibility and makes it open for implementations of different components from various software vendors.

The paper is organised as follows. Section 2 introduces the concept of execution strategies for adaptive service compositions management identifying a set of basic tasks common for all execution strategies and presenting briefly the main strategies. Section 3 provides an overview of an (functional) architecture that may support the described execution strategies. This architecture is described in terms of software components and their public interfaces. The next three sections focus on basic tasks organised in various ways within the individual execution strategies. Section 4 describes selection and contracting of atomic services. Section 5 presents enactment and monitoring of an atomic service. Section 6 provides detailed information on exception handling. The paper closes with a conclusion and an outlook of future work.

2 Execution Strategies for Service Compositions

There are many possible strategies for executing a service composition. However, there is a common feature of all possible strategies - they are built on the top of basic tasks. These tasks[1] concern a single atomic service and are specified in **Table 1**.

Table 1. Basic tasks used in all execution strategies

Basic task	***Main responsibility***
Service Selection and Contracting (SC)	Select and contract a concrete atomic service based on its functional (service type) and non-functional (QoS constraints) requirements. In some execution strategies, SC task is done for the whole composition (all atomic services included) before enactment and monitoring. In the other strategies, SC task is intertwined with enactment and monitoring of the individual atomic services according to the service composition specification.
Service Enactment and Monitoring (EM)	Enact (invoke) a concrete atomic service and monitor its execution (w.r.t QoS constraints). Enactment is done according to the service composition specification given as a WS-BPEL process.
Exception handling (Ex)	Three steps to handle exceptions have been identified: 1) renegotiate the contract with the current atomic service provider (and possibly with other service providers affected by the exception); if not possible then 2) re-select a replacement from other atomic service providers that match the service specification and new requirements; otherwise 3) re-plan the whole service composition.

The number of basic tasks used in a strategy depends on the strategy itself, and on the number of atomic services included in a given service composition. As usual, there is no one optimal strategy. Every strategy focuses on different aspects of service executions and has both advantages and disadvantages. Two basic (quite opposite) execution strategies (see also [11] for more details) and three intermediate execution strategies are described in the consecutive sections.

[1] The tasks which are not directly related to execution of a service composition have been omitted (e.g. registration or un-registration of an atomic service).

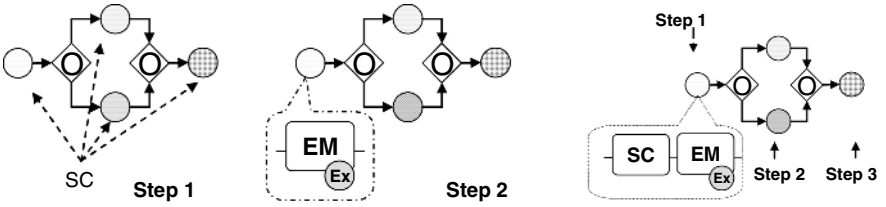

Fig. 1. The concept of the basic strategies: first-contract-all-then-enact (left) and step-by-step-negotiate-and-enact (right)

2.1 First-Contract-All-Then-Enact Strategy

This strategy assumes (see figure 1, left hand side) that selection and contracting of all atomic services included in the service composition (actual execution) is done before its execution (step 1). Execution and monitoring of the individual atomic services is done step by step according to the control flow defined for the service composition. (step 2) Any failure reported during service execution is handled by exception handling mechanism described in the previous section.

This strategy makes it possible to guarantee non-functional requirements for the whole service composition (global level). Since contracting is done before execution, concurrent selection and negotiation is allowed. As a result, it is possible to consider aggregated concessions and preferences (e.g., if the same provider provides services for several atomic services, then some discount may be regarded), so that the service composition QoS parameters can be optimised.

On the contrary, in this strategy all the activities on conditional branches need to be selected and contracted although some of them may never be enacted. A reservation mechanism is needed. Also service implementations registered during service execution cannot be selected. Finally, the strategy requires coordinated negotiation mechanisms with a coordination agent and a set of negotiation agents.

2.2 Step-by-Step-Contract-and-Enact Strategy

This strategy assumes (see figure 1, right hand side) that selection and contracting is done just before atomic service execution. The order of the executed services is determined by definition of the composition. That means that the first atomic service in the service composition can be executed and monitored when its SLA document is established (see figure 1, step 1, first SC then EM). After completion of this atomic service, the selection and contracting is carried out for each subsequent atomic service and followed by its execution (steps 2 and 3). Any failure occurred during service execution is handled by exception mechanism described in the previous sections.

This strategy allows for ‘on-the-fly’ selection and contracting based on actual QoS values of services that have been executed. This will lead to more accurate and efficient contracting (esp. negotiation) since it is based on what it had been done for the executed services. Only the invoked atomic services are contracted. Services included in the composition but not executed (i.e. included in conditional branches that have not been chosen) are not considered. In addition, selection is able to use atomic services registered after starting execution of the service composition.

On the contrary, the strategy mainly optimises QoS for atomic services which are about to be invoked (local level). Optimisation of the global QoS parameters is hard. Instantiation and execution of the whole service composition can not be guaranteed (i.e. a service implementation is executed but it may be impossible to select and contract a subsequent service implementation) thus failing the whole service composition and wasting already executed services (a need for un-doing the services).

2.3 Other Strategies

The **late-contracting-then-enact strategy** assumes that the selection and contracting of atomic services is done before their execution, as soon as it is sure that they will be executed within a given composition. If, for example, there are two alternative branches, as soon as it is known which of them will be taken, all atomic services on the satisfied branch are selected and contracted. Execution of the atomic services is carried out according to the control flow definition. This strategy is similar to the first-contract-all-then-enact strategy but minimises the risk in contracting services which will never be executed. The risk to not satisfy the global QoS requirements is less than for the mentioned strategy but still exists.

The **first-contract-plausible-then-enact strategy** tries to select and contract first (before service composition execution) all atomic services that belong to the composition path which is the most likely to be executed. The path is predicted on the basis of historical data from previous executions of the service composition. The services that belong to other paths are not selected and contracted. Execution of the atomic services is carried out according to the control flow definition. This strategy minimises the risk of a) contracting services that with high probability of not being executed, b) satisfying the global QoS requirements. However, it will work properly only for that cases in which execution concerns the most probable path in the composition. For the other paths it will have similar problems as the step-by-step-contract-and-enact strategy.

The **first-contract-critical-then-enact strategy** selects and contracts before execution only those atomic services which are hard to be contracted dynamically. 'Hard' in this context means that the number of service candidates for those service specifications is significantly lower than the number of candidates for the other services included. This strategy is similar to the step-by-step-contract-and-enact strategy but reduces the risk of not satisfying the global QoS requirements. On the other hand, it has similar problems with branches which will never be executed.

2.4 Representations for Service Compositions

Specification of a service composition is provided at the design phase of adaptive management of service compositions. This specification describes the composition in terms of control flow (i.e. the order of the invoked atomic **services**) and data flow (mapping between input and output parameters of atomic services). This specification operates on classes of atomic services, instead of concrete services. Basically, such a specification provides a solution to satisfy all (static) functional requirements (classes of atomic services that together satisfy them) and leaves flexibility during processing a request (i.e. execution of the composed service) to select and contract concrete

atomic services of given service classes that also satisfy its (dynamic) non-functional requirements.

Making the service composition specification generic is very useful, but on the other hand it prevents from using standard execution engines that process WS-BPEL processes. In order to cope with this problem we propose an intermediate solution based on the concept of generic workflows proposed in [1]. We represent the specification as a **WS-BPEL process** and, instead of invoking concrete atomic services, we invoke concrete **brokers (services) for atomic service classes**. Every atomic service class has its own broker. Such a broker has input and output parameters as every concrete service of this class. In addition, it receives as an input parameter a set of non-functional requirements that makes it possible to choose an appropriate atomic service of a given class.

The way of using service composition specification at the execution phase depends on the applied execution strategy. For the execution strategies which carry out contracting of more than one atomic service before their enactment (e.g. first-contract-then-enact), the specification will be analysed by service selection and contracting component (based on WS-BPEL) in order to contract appropriate atomic services. After that, the input parameter representing non-functional requirements will be replaced with information of the selected atomic service and a reference to the agreed contract (SLA document). For the other execution strategies, which intertwine contracting and enactment, the specification will be interpreted by service enactment component (again based on WS-BPEL) which will invoke appropriate service class brokers. These brokers will be responsible for organising selection and contracting by themselves and then assure appropriate invocation and monitoring.

History from execution of a service composition is represented as a **service composition execution**. This includes basic information about service composition and invoked atomic services (transformed information from invocation of service class brokers) in the form of a workflow log. This log is used for service profiling.

3 Architecture Overview

On the basis of the execution strategies briefly presented in the previous sections we present a functional architecture of an adaptive service composition management subsystem (i.e. part of a larger system to manage service compositions, for example Adaptive Service Grid platform [7]). In general, this architecture aims at implementation of all previously described strategies.

The subsystem provides two public interfaces (see figure 2): *scEnactment* and service *scMonitoring*. The first interface includes just one method *enact* which is responsible for execution of a service composition. As the input this method requires specification of a service composition (given as a WS-BPEL process), input data and non-functional constraints (e.g. maximal duration). In the first step, the subsystem determines the best strategy to execute the composition. This strategy is determined on the basis of composition specification (static) as well as profiling data (dynamic, given for concrete service but also for service type). The next steps are specific for individual execution strategies and are related to basic tasks described earlier.

In the first-contract-all-then-enact strategy, all abstract atomic services (i.e. service type brokers) included in the composition are used to select concrete atomic services and to contract one of their composition that is able to satisfy all QoS constraints. Within service specification all abstract services are replaced by concrete already contracted atomic services. In the next stage such composition defined as a WS-BPEL process is executed. In the step-by-step-contracting-and-enactment strategy the composition is executed step by step. The composition is represented as a WS-BPEL process which instead of concrete services includes calls to service type brokers. When a broker is executed it first triggers selection and contracting for a given abstract service and then, having concrete atomic service, executes it.

The second subsystem interface, namely *scMonitoring* is responsible for monitoring and administration of the executed service compositions. It provides methods to get information about execution history of a given composition (in a form of execution log), to visualise its execution or to do some administration tasks such as suspending or resuming.

The subsystem requires three interfaces from atomic service providers: one (optional) for negotiation of the contract between the service composition provider and a given atomic service provider, one for invocation of the atomic service (obligatory), and one for (also optional) gathering the values of QoS parameters agreed to be monitored by the service provider.

The subsystem consists of six software components: Execution Coordinator, Service Selection & Negotiation Manager, Service Level Agreement Manager, Service Enactment Engine, Service Monitor, and Dynamic Service Profiler.

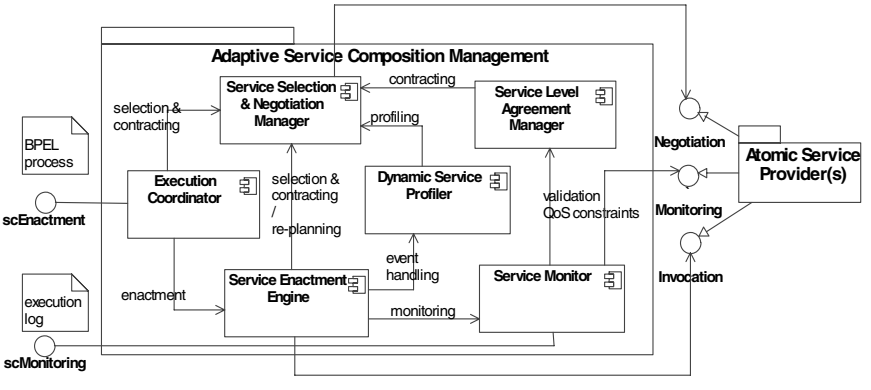

Fig. 2. The functional architecture for adaptive service composition management

Execution Coordinator is responsible for selection of the best execution strategy for a given service composition. To do that it analyses specification of service composition and uses profiling data. It also validates strategy selection rules checking what strategy satisfies the greatest number of the rules.

Service Selection & Negotiation Manager is responsible for selection and contracting atomic service implementations that may be invoked within a given service composition. This selection is done for individual service specifications and is

based on both functional as well as non-functional requirements. On the basis of a generic process (i.e. WS-BPEL process which invokes service type brokers, extended of information required to select service implementations), it makes a concrete process replacing atomic service type brokers with the most appropriate atomic service implementations. The component also provides negotiation features that support the service composition provider with possibility to negotiate some/all non-functional requirements that were not satisfied initially by the selected services. The contracts used during negotiation and contracting are taken from SLA Manager. To negotiate with atomic services it uses *Negotiation* interface provided by their providers and FIPA Iterative Contract Net Protocol. Profiling data are extracted from Dynamic Service Profiler.

Service Level Agreement Manager (SLA Manager) is responsible for management of SLA documents compliant with WS-Agreement (XML format). The SLA Manager creates SLA templates that are used during negotiation and contracting, and stores the agreed SLA documents. In addition, it verifies SLA documents (i.e. QoS constraints) during invocation of respective atomic services. Meaning of QoS parameters is defined in WSQM standard.

Service Enactment Engine is responsible for enactment of a service composition represented as a WS-BPEL process. According to the process definition, the component invokes either individual atomic services (first contract then enact strategies) or service type brokers (step by step strategies) passing appropriate input data and gathering the invocation results. For atomic services (described in WSDL) such invocation is done by calling *Invocation* interface provided by the service provider (or any additional invocation layer). When a service invocation begins, the component starts monitoring. The component is also responsible for generating service execution events that may be handled by the other components. These events are related to behaviour of service execution entities (positive) or functional/non-functional runtime exceptions (negative). Dynamic Service Profiler uses this information to gather service execution history and update service profiles (i.e. service composition and atomic services profiles). Finally, in case of failures (functional), or violation of QoS constraints (non-functional), the component asks Service Selection & Negotiation Manager to either update the existing contract or to re-select and contract another service implementation.

Service Monitor component is responsible for monitoring of QoS constraints at two levels: the whole service composition as well as the individual atomic services. Monitoring at the service composition level starts/stops when the Service Enactment Engine starts/stops its enactment. Monitoring at the atomic service level is carried out for every invoked service. By analogy, it starts/stops atomic service monitoring when the atomic service starts/stops its invocation. For every atomic service there is one monitoring rule, which is triggered periodically (the resolution for this activity is determined on the basis of previous executions as well as required QoS parameters related to its duration). When it is triggered, Service Monitor asks the atomic service provider via *Monitoring* interface to provide the values for the QoS parameters defined as 'monitored by the provider' in respective SLA document. These QoS parameters are completed with other QoS parameters monitored by the subsystem itself and included in the SLA document. This document is then verified against QoS

constraints by *Validation* interface provided by SLA Manager. If there is any violation of QoS constraint, it is reported to Service Enactment Engine.

Dynamic Service Profiler is responsible for gathering data describing basic QoS parameters and, on their basis, calculating more advanced QoS parameters. To collect all basic QoS parameters it listens to service execution events.

4 Selection and Contracting

The detailed scenario for selection and contracting of an atomic activity is presented in figure 3. *Service Selection & Negotiation Manager* finds all atomic services of a given type (steps 1, 2). In addition, for all returned services the component asks Dynamic Service Profiler for profiling data that may be used in further validation of QoS constraints (steps 3, 4).

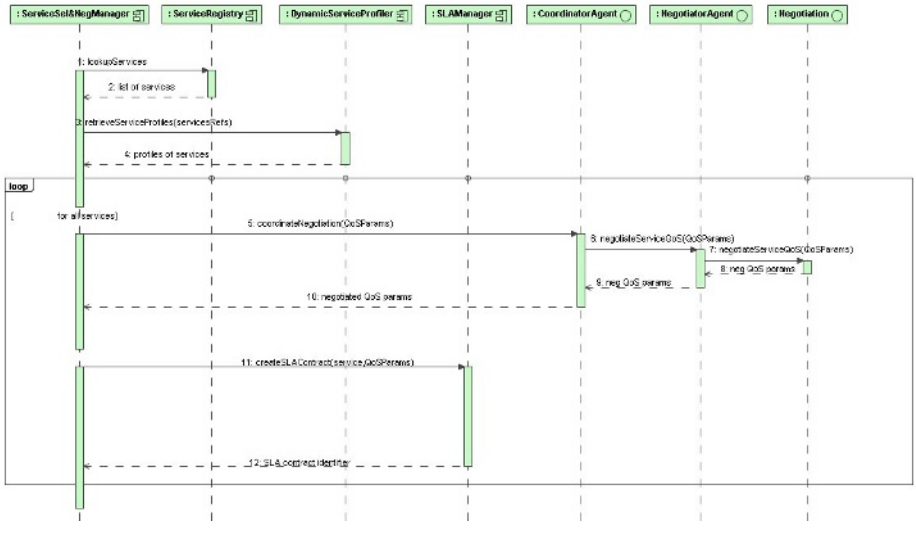

Fig. 3. Selection and contracting services by iterative negotiation

Then an iterative process of service selection starts (steps from 5 to 12). This process is organised by a Coordination agent and usually based on negotiation techniques [3]. This agent starts and then coordinates all negotiation agents which are responsible for negotiation QoS parameters with individual service providers (steps from 6 to 9). Negotiation may be iterative. In general, there are some configuration parameters which describe how many iterations is possible, and what is the weighted value of the negotiated QoS parameters. If this value is acceptable (i.e. between minimum and maximum required values) then the selection process is completed and the selected atomic service is contracted. To contract it, *Service Selection & Negotiation Manager* asks *SLA Manager* to create a contract with the agreed QoS parameters. This contract is represented as a SLA document and managed by *SLA Manager*. Reference to the SLA document is added to service composition

specification. This gives an opportunity to verify actual QoS values against expected QoS ones for every invoked atomic service.

5 Enactment and Monitoring

The detailed scenario for (positive) enactment and monitoring of an atomic activity is presented in figure 4. Based on the atomic service implementation reference *Service Enactment Engine* calls *Invocation interface* (of *Atomic Service Provider*) to retrieve a service instance.

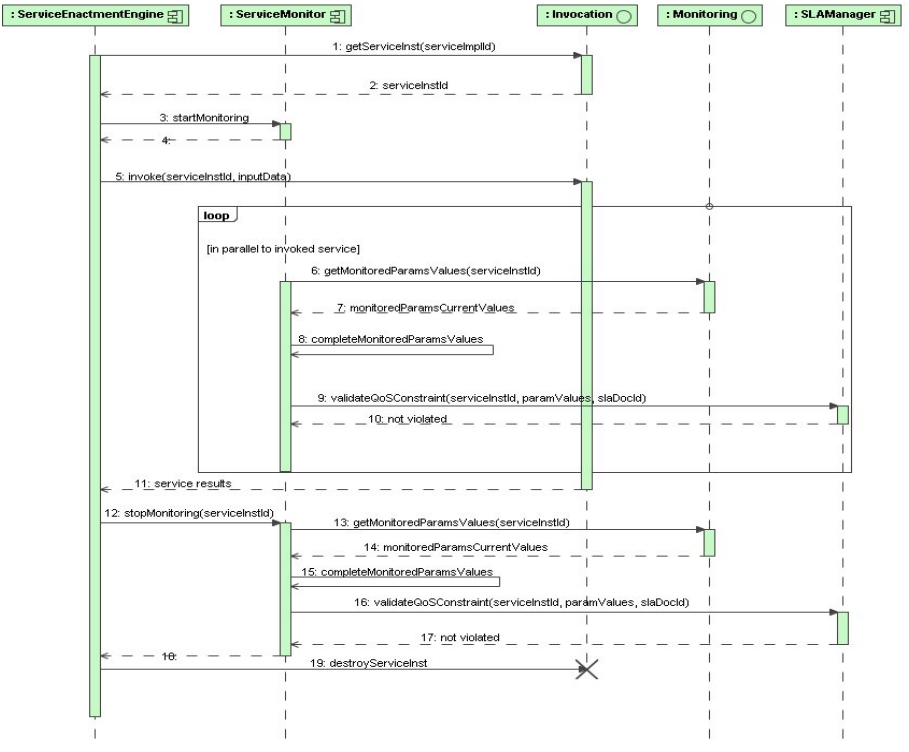

Fig. 4. The positive scenario of an atomic service execution

After that it uses *Service Monitor* to run monitoring for this particular service instance. Then again *Service Enactment Engine* calls *Invocation* interface to invoke the service instance passing its input data. Service invocation and monitoring are conducted in parallel. With the assumed resolution (i.e. how often the monitor verifies a QoS constraint) *Service Monitor* gathers current values of monitored parameters through *Monitoring* interface and verifies them against QoS constraint (for the service) using *SLA Manager*. The resolution can be expressed via different strategies – once the whole service execution, as often as possible or service profile value, which is the default choice that is equal to the half of the minimal service execution

time (assigned a priori and adjusted periodically after real executions). When the service instance finishes its execution, *Service Enactment Engine* stops monitoring and the post-execution QoS validation is conducted. Additionally *Service Monitor* performs composed level QoS validation i.e. it checks whether (even positive) result of the currently finishing atomic service does not harm the overall (at composed level) QoS constraints. At the end *Service Enactment Engine* calls *Invocation* interface to destroy the service instance.

6 Exception Handling

During execution *Service Monitor* must be prepared to deal with different exceptional situations, which are caused either by QoS violations or any other runtime exceptions. Figure 5 presents a (negative) atomic service execution scenario where *Service Monitor* is informed by *SLA Manager* about QoS constraint violation. In such case the first step is to suspend execution of the whole composition. Then *Service Enactment Engine* calls *Service Selection and Negotiation Manager* (*SS&NM*) to re-plan the composed service passing a composed service execution log (i.e. history of execution together with the current state).

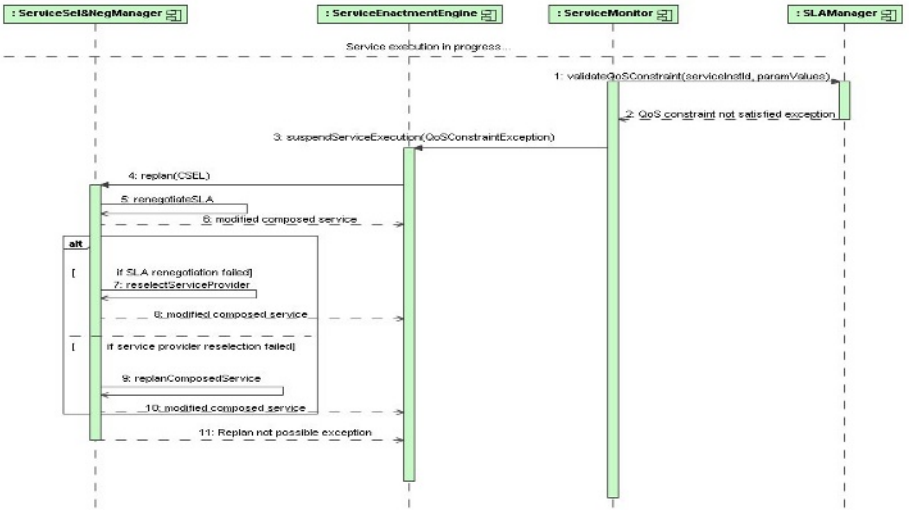

Fig. 5. Exception handling in case of QoS violation

The latter component tries to resume service execution using the following approaches. First it tries to re-renegotiate the contract with the existing service (i.e. increase its cost using spare money and decrease its duration). If it is not possible, the in the second step the component checks if a given atomic service may be replaced by another atomic service of the same type. Such verification is carried out as a standard selection & contracting procedure. The only difference is the set of atomic service candidates which excludes service(s) that already failed. If it is also impossible then

final possibility is to re-plan the whole service composition that will meet the overall request requirements. This process involves a service re-composer and is described more in detail in [10].

If all of the mentioned steps fail, *SS&NM* throws an exception. In the aftermath of it *Service Enactment Engine* is not able to continue service execution and reports it to the service composition provider. Otherwise *SS&NM* delivers a modified composed service so that *Service Enactment Engine* could continue the execution.

7 Conclusions

Adaptive management of service compositions is an area of web services research that has recently been attracting more and more attention. The most important questions stimulating research in this area include proper management of service compositions in order to satisfy not only functional requirements, but also non functional ones. Another important issue is adaptation to dynamic changes in the service environment, which so far has not been researched to a satisfactory extent, and there is still a clear need for improvement of current methods and tools.

The work described in this paper aims at defining an (functional) architecture towards adaptive management of QoS aware service compositions. This architecture is able to support various execution strategies. In addition, because of the use of the standard data exchange formats and communication protocols, the architecture is flexible and can be implemented by various vendors. A first positive validation of the architecture has been done for the Dynamic Supply Chain of Internet Services (DSC) scenario [7] implemented in the ASG project using the first-contract-then-enact strategy.

In the future steps, the architecture needs to be validated for other strategies and yet more advanced real cases. We also plan to investigate usefulness of other relevant standards especially related to Web Service Enhancement 2.0. Finally, efficiency of the architecture will be evaluated for further improvements.

References

1. Aalst, W.M.P. Van der: Generic Workflow Models: How to Handle Dynamic Change and Capture Management Information? Computer Systems, Science and Engineering, 15, 2001.
2. Boström, G., Giambiagi, P., Olsson, T.: Quality of Service Evaluation in Virtual Organizations Using SLAs. submitted to 1st Workshop on Interoperability Solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems, 2006.
3. Braun, P., J. Brzostowski, J., Kersten, G., Kim, J., Kowalczyk, R., Strecker, S., Vahidov, R.: E-Negotiation Systems and Software Agents Methods, Models, and Applications. In i-DMSS: Foundations, Applications and Challenges. UK, 2005.
4. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Ming-Chien Shan, M., Ch.: Adaptive and Dynamic Service Composition in eFlow. HP technical Report HPL-2000-39, March.
5. Comuzzi, M., Penrici, B., An Architecture for Flexible Web Service QoS Negotiation, EDOC, 2005.

6. Gu, X., Nahrstedt, K., Chang, R., Ward, and C.: QoS-assured service composition in managed service overlay networks. Proc. of Distributed Computing Systems, 2003.
7. Integrated Project "Adaptive Services Grid", http://asg-platform.org
8. Momotko, M., Gajewski, M., Ludwig, A., Kowalczyk, R., Kowalkiewicz, M., Zhang, J. Y.: Towards Adaptive Management of QoS-aware Service Compositions, International Journal on Multiagent and Grid Systems, volume 2:2, Sept 2006 (to appear).
9. Salle, A., Bartolini, C.: Management by Contract, HPL-2003-186, HP labs, 2004.
10. Weske, M., Gajewski, M., Momotko, M., Mayer, H., Schuschel, H.: Dynamic Failure Recovery of Generated Workflows. DEXA'2005, BPMPM Workshop, 2005.
11. Zeng, L., Benatallah, B., Lei, H., Ngu, A., H., H., Flaxer, D., and Chang, H.: Flexible composition of enterprise web services. Electronic Markets - Web Services, 2003.