Model-based QoS-enabled self-healing Web Services

O. Nabuco[3], R. Ben Halima[1] K. Drira[1], M.G. Fugini[2], S. Modafferi[2], E. Mussi[2],

[1]LAAS-CNRS, Université de Toulouse; *7 avenue de Colonel Roche, 31077 Toulouse, France*
DEI, Politecnico di Milano; *Piazza da Vinci, 32 I-20133 Milano, ITALY*
[3]CENPRA; *Rod. D. Pedro I, Km 143,6 - Campinas 13069-901 BRAZIL*

Abstract

Failures during web service execution may depend on a wide variety of causes, such as network faults, server crashes, or application-related errors, such as unavailability of a requested web service, errors in the orchestration of choreography of applications, missing data or parameters in an execution flow, or low Quality of Service (QoS). In this paper, we propose a healing architecture able to handle web service faults in a self-healing way, discussing infrastructural faults and web service and Web application faults. The self-healing architecture manages repair actions, such as substitution of a faulty service or duplication of overloaded services. Implemented prototypes involving QoS in coordinated web services are illustrated and discussed. [1]

1 Introduction

The creation of e-services, based on dynamic resources, requires innovative design approaches to guarantee service quality; in particular the services must be reliable and available and must comply to quality dimensions, such as accordance to requisites, QoS aspects (such as the timeliness in the service execution, its transactional properties, security, and similar), stability, etc. The goal of integrating heterogeneous components has been addressed in the Service Oriented Architecture research area. Although web services address integration issues such as communication through firewalls and interoperability between heterogeneous components, QoS management is still an open issue. For instance, failures are handled and possibly recovered in a static way by employing precompiled compensation strategies (e.g., see BPEL). As a step forward in this direction, dynamic handling of failure and recovery should be supplied. In general, exceptions and faulty states are a primary issue in web service and workflow design, rather than a mechanism added after the fact via a library approach. Exception handling, a primary feature in language design, must be integrated with other major features, including advanced control flow, objects, coroutines, concurrency, real-time, and polymorphism. Self-healability is the property that enables a system to perceive that it is not operating correctly and, without human intervention, make the necessary adjustments to restore itself to normality [1]. Self-healing systems are clearly recovery-oriented and require monitoring the system to detect transitions to faulty modes, to diagnose the situation, and to appropriately choose and execute a recovery strategy. In order to apply diagnostic methodologies to web services, we focus on both composite and conversationally complex web services. Therefore, the diagnostic process needs to take into account not only the different cooperating services, but also the complexity of their interactions, that can take different turns and dynamically change the internal status of each service. Moreover, the diagnostic process needs to be hierarchical and decentralized: a different diagnostic process should take care of each composite web service, but such processes need to communicate in order to achieve a satisfactory diagnosis. Another type of error may be due to the inability of services(s) to fulfill QoS requirements. Service level agreements and contract specification have been studied in the literature [2,3]. In web services, QoS has to be tackled also during the execution of the services. Quality parameters are likely to be highly variable, in particular in mobile contexts, and with dynamically evolving services, so there is a need to observe and measure them on line. An ontology-based approach can bring the shared vocabulary among the partners defining a formal language that can evolve describing classes' relationship in terms

[1]This work has been developed within the WS-Diamond UE FET-STREP Project n IST-516933, started September 1^{st}, 2005 with 30 months duration.

of axioms and rules. This paper addresses self-healing systems through an ontology and a tools architecture.

Section 2 presents the Self-healing ontology, Section 3 the process-level self-healing architecture, Section 4 the interaction-level self-healing architecture, and Section 5 our concluding remarks.

2 WS-Diamond Self-Healing Ontology

The objective of a self-healing ontology is to describe a self-healing model. It can also provide or point to more detailed ontologies depending on the domain of application and other specifities. The concepts used in the construction of the ontology were gathered from the deliverables generated by the WS-Diamond project.

Figure 1. Healing ontology detail

These descriptions also support the detailed reproduction of the model by other implementers creating the "healing" vocabulary to project developers. Moreover each class has its role in the healing model providing a consistent service description and interaction.

The ontology has an upper level providing definitions of Scope, Service Level, Acting Level, and Behavior, giving the necessary context to designers, providers and consumers. The ActingLevel describes if the healing intervention action will occur: at the Process – the service running procedure - or at the Interaction level – service communication. The next two sections describe interventions in both levels. Scope: can be *Local* -if the repair action will take place in a single web service or *Global* - the context of a complex and composed web service.

WSLevel is composed of Instance – indicating if the intervention of healing is been taken in a occurrence of a single or global service and Class – indicating if the intervention is been taken in the model description (this task is performed by a human specialist).

WSBehaviorDescription - Each service has a description composed of its orchestration and/or its choreography, or BPEL or OWL-S description. The model doesn't prescribe nor language neither ontology to do so but indicates that it shall have one and indicates the nature and point where the description is.

The DiamondHealingModel is the class that encompasses the healing model. As shown in Fig.1 the classes correspond to the used concepts: Goals, Policies, and Targets to be pursued by the healing components. Definition of components to Monitoring, Diagnosis and Repair as also the classes that can determine if a service is decaying its performance, corresponding to designed expectations in QoS, and what parameters can be sensors for the service (Observable Parameters).

The healing process begins defining its goals: prevent or repair web services from QoS deviations. In order to prevent errors, it has to monitor specific QoS parameters, defined at the selection phase of the composition of the service, through observable parameters (that can be part of the original service or be added to it). Monitoring of these observable parameters can be done at the service level – the instance that can be observed by its choreography –or the flow level –the procedure of running the sequence of web services.

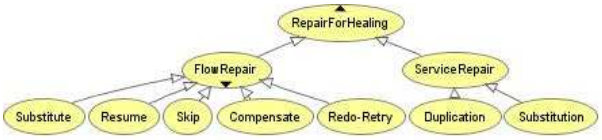

Figure 2. RepairForHealing Class detail

This healing process was translated into classes, parameters, rules, and relationships of the DiamondHealingModel ontology.

In order to Recover from QoS degradation, the healing model takes advantage of the Diagnoser which receives information from the monitor module. The Diagnoser then analyzes the situation recommending the repairing actions described by the Repair for Healing class.

RepairForHealing class contains two classes, representing the possible repair actions able to prevent or recover from faulty states. **ServiceRepair** class represents the preventive action at Service target. It has relationship with the choreographed description, and can have Local or Global scope. Actions can be in terms of Substitute – for a semantic equivalent or Duplicate - starting the same service using another infrastructure - a service. The **FlowRepair** class represents the repair actions at Flow target, for orchestrated services. Actions can be taken during workflow execution in terms of Redo-Retry, Compensate, Resume, Adjust, Inform, Skip and ValChange. The **QoSParameters** class rep-

resents the attributes that characterize the desirable behavior of the service representing shared concepts for users and providers. The QoS classes and parameters are based on previous and current works defining the QoS in web service domain [4], [5], [6] among others.

Monitoring for Healing class determines what parameters can be monitored at MessageLevel – in terms of message's envelope or protocol – or ProcessLevel – parameters inside the workflow, for instance, time out related parameters. **HandledFaults** class means if the kind of fault the system can deal with, like data semantic faults or non-functional faults, like QoS violations.

Policies take into account the managing actions and components that deal with the preventive or recovery characteristics of the chosen healing.

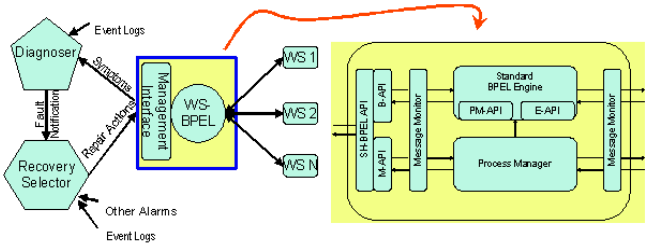

Figure 3. Diagnosis and Repair approach and SH-BPEL engine architecture

3 Process-Level Self-Healing Management

Self-healing can be obtained at the level of the single service, and also at a more global level, with support to identify critical misbehavior of groups of services and to provide web services with reaction mechanisms to global level failures. Our goal is not only to provide methodologies and techniques for on-line diagnosis and recovery of web services, but also to provide guidelines for designing diagnosable and recoverable web services. The focus of WS-Diamond is on composite and conversationally complex web services.

Composite Web Service: a composite web service relies on the integration of various other services: the integrated service is the result of the interaction among simple services, each carrying out a part of the job. Such interaction, which can be described as a workflow, is hidden from the consumer application, which perceives the overall service as an atomic one.

Conversationally Complex Web Service: a web service can be complex in the sense that during the service provision it needs to carry out a complex interaction with the consumer application, where several conversational turns are exchanged. In some cases, the order of the conversation turns is not completely predetermined, but depends on the activities carried out by the web service.

In WS-Diamond, two self-healing levels are addressed: process and interaction. The process level acts on the BPEL flows using SH-BPEL (Self-Healing BPEL) [7], a self-healing plug in for BPEL engines. In Fig.3, the overall integrated WS-Diamond approach to diagnosis and repair is depicted. The mechanism works with a possible notification to the Diagnoser of a Symptom, detected through a timeout, and the corresponding faulty service. The Diagnoser can be activated either through this Symptom or by its internal monitoring routines. The Diagnoser sends a fault notification to the Recovery Selector which determines the appropriate repair action. In WS-Diamond, a Service Execution Engine handles the correct execution of complex services based on such integrated diagnosis and repair approach. The focus of the Engine is on mechanisms to execute self-healing actions on web services of a process. Process representation in the self-healing environment supports monitoring of web services choreography/orchestration and its conversational behavior, and data and temporal dependencies among process activities. The activity performed in this layer is the evaluation of actual and potential violations of process functions and quality, for obtaining self-healing composed services by exploiting diagnosis and repair actions.

Recovery in WS-Diamond acts both on single web services and on processes composed of several web services. One tool provided for self-healing services and processes is SH-BPEL (see Fig.3), designed to execute recovery actions such as: (i) *retry* the execution of a process activity, (ii) *redo* the execution of a process activity but using parameters different from the ones used in the previous execution, (iii) *compensate* a process activity by executing proper recovery actions, (iv) *update* the value of an internal process variable, and (v) *substitute* one or more faulty web services. In particular, the substitution can be indifferently performed over both stateless and statefull web services. SH-BPEL can be controlled by means of a *Management Interface* implemented using the WSDM standard. From the *methodological* point of view, SH-BPEL performs recovery actions by means of three different mechanisms (see [7] for details): i) *Standard BPEL recovery mechanisms* explicitly defined by the designer; ii) *Pre-Processing based BPEL recovery mechanisms* obtained by requesting to the designer the insertion of proper

recovery tags, the tagged BPEL is then pre-processed by an interpreter that transforms it in a standard one able to perform also the selected recovery actions; iii) *Extended recovery mechanisms*. These are executed by a separate module of the SH-BPEL plugin and do not interfere with the process orchestration, in that recovery actions are *transparently* executed.

Techniques are being developed for providing recovery mechanisms in self-healing web services, in addition to fault handlers and with more general functionalities not requiring specific programming by the designer. In particular, services and processes have a *Management Interface* enabling the analysis of the state of a service/process and hence allowing the execution of given business operations on it (e.g., a repair action), depending on its state. Compared with the existing web service enabled application servers, a WS-Diamond server provides an environment to run adaptive web services on the basis of a QoS driven approach. The requester may specify quality requirements at web service invocation time or these requirements may be implicitly specified as *annotations* of the services. If the WS-Diamond platform realizes that the QoS of a web service is decreasing, then it perform a channel switching, or web service substitution. Similarity computation, supported by the ontology, is based on a semantic-based analysis of the involved web service. Since substituted and substituting web services might have different signatures, a semi-automatic wrapper generator reconciliates differences in the provided interfaces, possibly with human intervention.

Monitoring captures process-related potentially faulty behaviors and triggers recovery actions. The focus of monitoring is on aspects which are not under investigation in the diagnostic modules, and in particular: proactively monitoring time constraints, to anticipate possible future time violations; monitoring errors due to architectural problems, or to QoS violations; monitoring conversations and analyzing their behavior with respect to their expected behavior. Repair actions are performed on web services through the management interface part of the SH-BPEL engine (see Fig. 3) through which they can be invoked for a given process to be repaired from a failed state. Repair actions can be classified as *service level actions* and as f*low level actions*. The output set of extensions is built so that it meets two requirements. i) The set of extensions is sound and complete with respect to the input. This means that the portion of model represented by the assignments in the set is equivalent to the portion of the model represented by the input assignment plus possibly the local observations. ii) Each extension in the set is admissible with respect to the local model. The notion of admissibility is meant to formalize the least-commitment strategy: intuitively, a partial assignment is admissible if it does not allow inferring anything about the unassigned variables that couldn't be inferred using the model alone.

Diagnosed faults indicate which service originated the fault and faulty messages, in particular the erroneous message(s) deriving from the wrong execution in the faulty service. A fault is identified by a *service-message pair* $F =< S, M >$ (S is the faulty service, M is the erroneous message). For each *failure-fault pair*, a plan contains the repair actions needed to resume the correct process execution. Repair actions in a plan are those defined above for SH-BPEL, and include repair from QoS faults.

4 Interaction-Level Self-Healing Architecture

The interaction level acts on the communication protocols. It proceeds by intercepting messages, by extending headers with QoS parameters values, and by processing content's information at the body level of the SOAP envelope. The interaction level relies on a connector-based architecture which interconnects WS requesters to WS providers. The monitoring-level connectors manage QoS by monitoring, stamping, measuring and logging requests and responses. The repair-level connectors are generated automatically from WSDL specifications. They are capable of substituting a deficient web service by a functionally equivalent web service. The repair-level connectors may be deployed on the requester-side or on the provider-side according to a set of authorizations. Three main steps are distinguished in the self-healing process [8]: *Monitoring* to extract information about the system health (using knowledge about the system configuration), *diagnosis* to examine and analyze them, and *repair* by executing recovery actions in order to heal the system. In our architecture, we operate these three steps in three software layers: the *Monitoring Layer*, the *Diagnosis and Planning Layer*, and the *Reconfiguration Layer*. Each layer is composed of several components interconnected as depicted in Fig.4 and exchanging messages which are presented in Table 1.The *WS Requester* sends a request (message *M1*) to the *Virtual WS Provider*. This message is intercepted by the *Requester Side Monitoring Connector*. Message *M1* is then extended by the first QoS parameter value (*QoSP1*) in the output message *M2*. For example, *QoSP1* may represent the invocation time of the service by the requester. Message *M2* is intercepted by the *Provider Side Monitoring Connector* for a sec-

ond time. *M2* is extended by the QoS parameter value (*QoSP2*) in the output message *M3*.

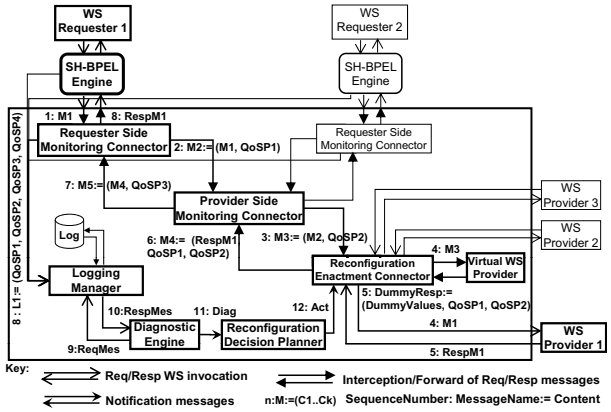

Figure 4. Interaction-level SH architecture

Message	Description
M1	Request Message
QoSP1	QoS parameter associated with the request M1 at the requester side
QoSP2	QoS parameter associated with the request M1 at the provider side
RespM1	Response Message for M1
QoSP3	QoS parameter associated with the response RespM1 at the provider side
QoSP4	QoS parameter associated with the response RespM1 at the requester side
L1	Stored log of monitored QoS values
ReqMes/RespMes	Extracted statistical measures related to QoS
Diag	Diagnosis report
Act	Reconfiguration plan

Table 1. QoS Message Description

For example, *QoSP2* may represent the communication time spent by the message to reach the provider side network. Message *M3* is intercepted by the *Reconfiguration Enactment Connector* (*REC*). The functional data are extracted from *M3*. This corresponds to the initial content of message *M1*. This content is used to dynamically create an invocation request with the same content towards the concrete *WS Provider* being bound to *REC*. It is also forwarded to the *Virtual WS Provider*. Responses of these two services are collected by *REC* which substitutes the *Virtual WS* response values by the concrete *WS* response values. In other terms, it replaces *DummyValues* by *RespM1* in the message *DummyResp*. As a result, we obtain *M4* as a response for the request. *M4* is intercepted by the *Provider Side Monitoring Connector* for a third extension by the QoS parameter value (*QoSP3*) in the output *M5*. For example, *QoSP3* may represent the execution time associated with the request. *M5* is intercepted by the *Requester Side Monitoring Connector*. It is then extended by the fourth QoS parameter value (*QoSP4*). For example, *QoSP4* may represent the time the response took to reach the provider side. The QoS data is extracted at this connector-level, and sent to *Logging Manager* (*LM*), a web service which saves data in a MySQL Data Base. The *Diagnostic Engine* (*DE*) interrogates periodically *LM* (message *ReqMes/RespMes*), analyzes statistically QoS values, and sends alarms and diagnostic reports (message *Diag*) to the *Reconfiguration Decision Planner* (*RDP*). When QoS degradation is detected, The *RDP* plans a reconfiguration and solicits *REC* for enactment (message *Act*). For example, *RDP* can ask for abandoning *WS Provider 1* and binding requesters to *WS Provider 2*. Consequently, requests will be redirected to the *WS Provider 2* instead of *WS Provider 1*.

5 Concluding Remarks

The WS-Diamond approach to self-healing web services provides a reference architecture and a self-healing platform: (1) for monitoring orchestration and choreography and detecting misbehaviour symptoms in complex WS-based applications, (2) for executing the diagnosis of functional faults and QoS degradation, and (3) for selecting and executing repair plans to recover from or to prevent QoS degradation.

References

[1] D. Ghosh, R.Sharman, H. R. Rao, and S. Upadhyaya. Selfhealing systems: survey and synthesis. *Decision Support Systems*. 42(4):2164-2185, 2007

[2] Keller A. and Ludwig H. "The WSLA Framework: Specifying and Monitoring Service Level Agreements for web services", Journal of Network and Systems Management Springer 11(1), 2003 pp. 57-8

[3] G. Wang, et al., "Service Level Management using QoS Monitoring, Diagnostic, and Adaptation for Networked Enterprise Systems". In EDOC Proc., 2005

[4] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. Journal of Web Semantics. (2004).

[5] Menascé, Daniel A. Response-Time Analysis for Composite web services. IEEE Internet Computing. January-February 2004. Pages 90-92.

[6] C. Zhou, L.T. Chia, B.S. Lee. DAML-QoS Ontology for web services. In ICWS Proc.,2004

[7] S. Modafferi, E. Mussi, B. Pernici. SH-BPEL - A Self-Healing plug-in for Ws-BPEL engines. In Workshop MW4SOC proc. (Melbourne, Au), 2006

[8] R. Ben Halima, M. Jmaiel, K. Drira. A QoS-driven reconfiguration management system extending web services with self-healing properties. In WETICE, 2007, Paris.