A QoS-driven reconfiguration management system extending Web services with self-healing properties

Riadh Ben Halima, Khalil Drira, Mohamed Jmaiel

A QoS-driven reconfiguration management system extending Web services with self-healing properties *

Riadh Ben Halima, Khalil Drira
LAAS-CNRS
7 avenue de Colonel Roche
31077 Toulouse Cedex 4, France
E-mail: {rbenhali, Khalil}@laas.fr

Mohamed Jmaiel
University of Sfax
National School of Engineers
B.P.W, 3038 Sfax, Tunisia
E-mail: Mohamed.Jmaiel@enis.rnu.tn

Abstract

This paper addresses QoS management for self-healing Web services. The objective is to provide a healing framework based on service monitoring and architectural-level repair actions. We address these topics in the context of the European project WS-DIAMOND. We implemented and assessed a connector-based healing layer capable of intercepting, analyzing and enhancing SOAP traffic and message contents with QoS data. Our framework supports Service Monitoring and dynamic run-time reconfiguration based on reflective programming.

key words: *Self-healing Web services, QoS management, reconfiguration, repair, service monitoring.*

1 Introduction

Internet progress has enabled data exchange between remote collaborators and the spread of on-line services. Today several platforms of services are provided to support the design, deployment and implementation of these services in reduced scales of time. The high dynamics of the Web requires new management infrastructures appropriate for Web services. This implies the necessity of deploying entities for supervising traffic between Web service providers and requesters in order to act for healing or preventing [3]. This includes also run-time QoS management in order to satisfy evolving process requirements and changing constraints. Research about self-managing and self-healing systems addresses such objectives [9, 1]. These systems are generally applied when the reliability and/or the QoS are required [5, 4]. A (QoS-centric) self-healing system inspects and changes its own behavior when the evaluation indicates that the intended QoS is not achieved, or when a better functionality or performance is required.

In the WS-Diamond project framework, three self-healing levels are addressed. The service-level repair which acts on manageable Web services having extended interfaces for management and applies to WSDM-compliant services. The flow-level repair which applies for orchestrated services and acts on the BPEL level using an extended manageable process execution engine. The communication-level repair which is the subject of this paper and acts on the exchanged messages. It proceeds by intercepting messages, extending headers with QoS parameters values, and substituting content's information at the body-level of the SOAP envelope. The communication-level repair relies on a connector-based architecture which interconnects WS requesters to WS providers. Some of the connectors process QoS management 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 or degraded Web service by another, functionally equivalent, Web service.

The connectors managing substitutions may be deployed on the requester-side and/or on the provider-side according to the available authorizations. Service monitoring and substitution may be conducted without any provider-related assumptions that may lead to restrictive constraints on the practicability of the process.

This paper is organized as follows. Section 2 describes our layered self-healing architecture. It represents an illustrative example of the reconfiguration layer action. In section 3, we describe the dynamic generation of the Reconfiguration Enactment Connector and associated operations. An overview of our current experimentations is presented in section 4. Section 5 discusses related work. The last section concludes the paper.

*This work was supported by EU Commission within the Project WS-Diamond (Web Services - DIAgnosability, MONitoring and Diagnosis, grant IST-516933).

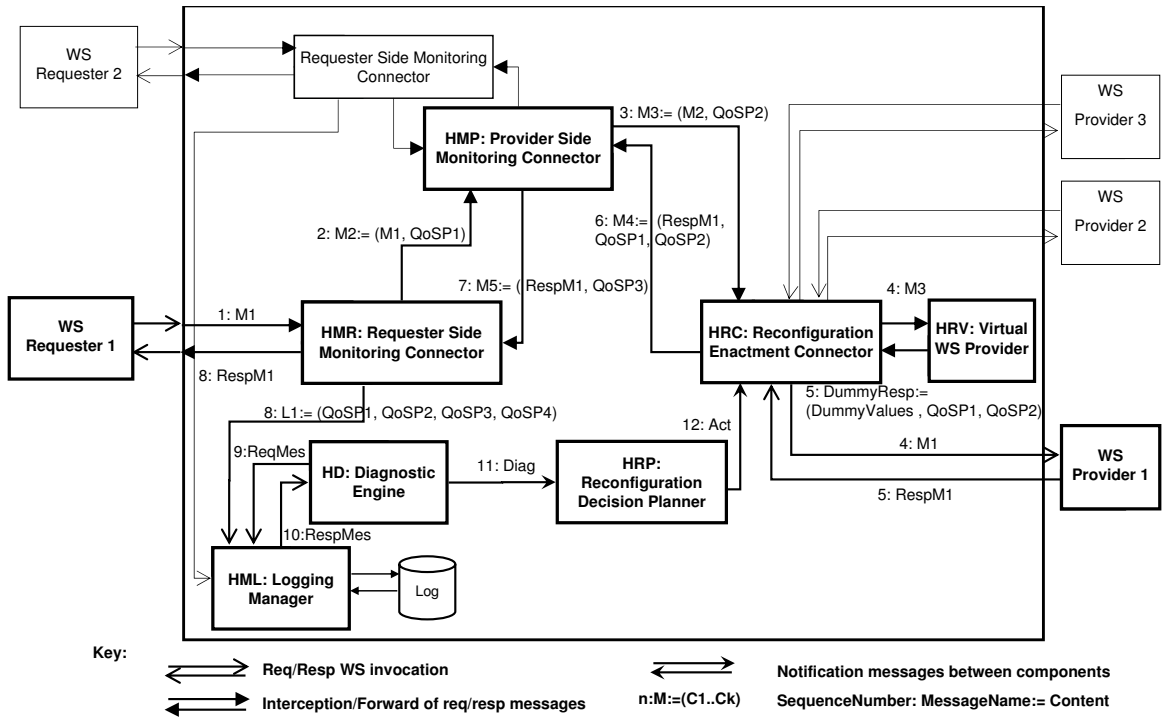

Figure 1. QoS-driven, connector-based proactive healing system architecture

2 Connector-based self-healing architecture

Three main steps are distinguished in the self-healing process [6]: *Monitoring* to extract information about the system health (using knowledge about the system configuration), *diagnosis* to examine and analyze them, and *repair* to heal the system by executing actions. In our architecture, we implement these three steps in three software layers. Each layer is composed of several components (see table 2). Messages exchanged are presented in table 1.

In this paper, we present the reconfiguration enactment step in the self-healing process in the framework of the WS-DIAMOND European project.

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 second time. M2 is extended by the QoS parameter value (QoSP2) in the output message M3. To illustrate this, 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. The functional data is extracted from M3. This corresponds

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. Description of service-level and healing-level messages.

Self-healing layers and components	Description
Monitoring and Measurement Layer	
HMR: Requester Side Monitoring Connector	Implements the Healing Monitoring at the requester-side (in short: HMR). Intercepts requests (M1) and responses (M5), inserts QoSP1 and QoSP4 values
HMP: Provider Side Monitoring Connector	Implements the Healing Monitoring at the provider-side (in short: HMP). Intercepts requests (M2) and responses (M4), inserts QoSP2 and QoSP3 values
HML: Logging Manager	Implements the Healing Monitoring data Logging management (in short: HML). A Web service used remotely to store QoS parameters
Diagnosis and Planning Layer	
HD: Diagnostic Engine	Implements the Healing Diagnostic (in short: HD). Interrogates periodically HML, analyzes statistically QoS values, and sends alarms and diagnostic reports to HRP
HRP: Reconfiguration Decision Planner	Implements the Healing Reconfiguration Planning algorithms (in short: HRP). Analyzes diagnostic reports, and decides to enforce a reconfiguration action sequence appropriately to the current context
Reconfiguration Layer	
HRC: Reconfiguration Enactment Connector	Implements the Healing Repair Connector (in short: HRC). Intercepts requests adressed to HRV, duplicates parameters for a concrete WS provider (CWS), substitutes HRV response values by CWS's response values
HRV: Virtual WS Provider	Implements the Healing Repair Virtual WS (in short: HRV). A dummy Web service (automatically generated from CWS WSDL description), it always responds with a dummy message to HRC

Table 2. Description healing layer components.

to the initial content of message M1. This latter is used to dynamically create an invocation request with the same content towards the concrete WS being bound to HRC. It is also forwarded to the Virtual WS Provider. Responses of these two services are collected by HRC. The HRC substitutes HRV response values by CWS's response values. In other terms, it replaces the DummyValues by RespM1 in the message DummyResp. As a result, we obtain the Message M4 as a response for the request. Message M4 is intercepted by the Provider-Side Monitoring Connector for a third extension by the QoS parameter value (QoSP3) in the output message M5. For example, QoSP3 may represent the execution time associated with the request. Message 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 requester-side. The QoS data is extracted at this connector-level, and sent to the Logging Manager (HML), a Web service which saves the data in a MySQL Database.

The Diagnostic Engine (HD) questions periodically HML (message ReqMes/RespMes), analyzes statistically QoS values, and sends alarms and diagnostic reports to HRP (message Diag). When a QoS degradation is detected, HRP plans a reconfiguration and solicits HRC for enactment (message Act). For example, HRP can ask for leaving WS Provider 1 and binding requesters to WS Provider 2.

Figure 2. SOAP Examples

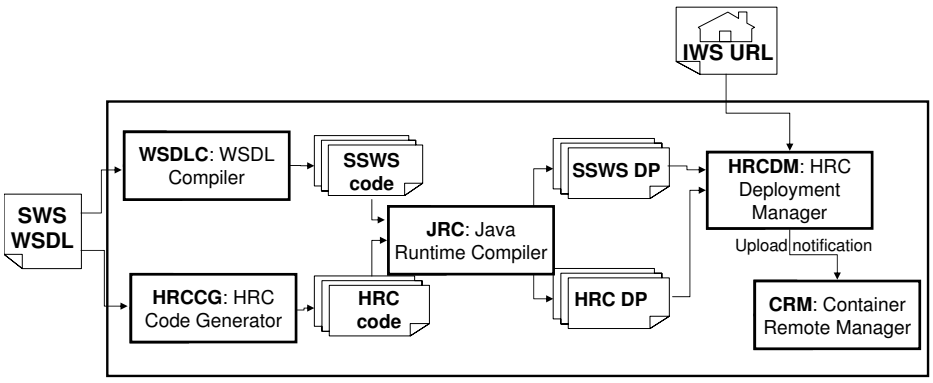

Figure 3. Generating the Reconfiguration Enactment Connector (HRC)

Consequently, requests will be routed to WS Provider 2 instead of WS Provider 1.

As an illustration, a first experience is done with a simple Web service which adds two integers and returns the sum (Web Service Provider 1). The Virtual WS Provider (HRV) and the Reconfiguration Enactment Connector (HRC) will be dynamically generated and deployed.

Figure 2 shows the SOAP content of exchanged messages of this example. Message M1 represents the initial SOAP message sent by the requester. It contains requester parameters ("1" and "4"). Message M3 is extended with two QoS parameters values that indicate the communication between the two sides.

In figure 2, message M5 represents a SOAP message request enriched with QoS parameters values. It shows the WS Provider invocation result bound to the Virtual WS SOAP message Request.

3 Implementation and experiments

3.1 Substitution policies

The substitution of service functionalities (that do not respect the required QoS) may be done through one or many other services. We distinguish three different cases:

The complete substitution: The substitute Web service (WS2) includes an over-set of the offered services by the faulty one (WS1). It will be used when the service WS1 is declared definitely a non-compliant service and has to be substituted. So, we redirect all requests to another service (WS2) which replaces entirely the WS1.
Functionalities (WS1)$\subset$ Functionalities (WS2).

The partial substitution: We substitute only the faulty methods. The connector will act on a part of the requester messages for the faulty methods, and redirect them to another service (WS2) which provides the same functionalities of the faulty WS1 methods.
Functionalities (WS1) $\bigcap$ Functionalities (WS2) = Functionalities(Requester).

The selective substitution: The offered functionality of the faulty Web service will be substituted by a set of Web services (WS2, SW3, ..., WSN). Each WSi will replace a part of the erroneous Web service.
Functionalities (WS1) = Functionalities (WS2) $\bigcup$ Functionalities (WS3) $\bigcup$ ... $\bigcup$ Functionalities (WSN).

We plan to extend the HRC for supporting all the substitution strategies while specifying substitution rules like work in [8].

3.2 Automated design of the Reconfiguration Enactment Connector (HRC)

The HRC is dynamically created and automatically deployed (see figure 3). Table 3 describes the generation process. In case of permanent QoS violation, the system deploys HRC in order to reroute requests towards a suitable Web service (SWS).

We use axis [http://ws.apache.org/axis/](version 1.4) as a middleware for Web services, tomcat (version 5.5.17) as a container and Java (version 1.5) as a programming language.

The routing process, implemented by HRC, is composed of five main operations as shown in figure 4. It is able to parse SOAP messages towards IWS (Using Axis and Jdom[1] APIs), and to dynamically create requests to SWS (Using Java Reflection[2]).

[1] http://www.jdom.org/: XML parser in Java
[2] http://java.sun.com/j2se/1.3/docs/guide/reflection/

Internal components	Description
IWS: Initial Web Service	The faulty Web service to be substituted
SWS: Substitute Web Service	The new Web service selected to solve QoS degradation generated by IWS
WSDLC: WSDL Compiler	Adapted WSDL compiler with configurable options for JVM management. It compiles SWS WSDL
HRCCG: HRC Code Generator	Generates the Java classes of the Reconfiguration Enactment Connector (HRC)
HRCDM: HRC Deployment Manager	Deploys all the connector files on the WS container
CRM: Container Remote Manager	A Web service that invoked remotely by HRCDM for reloading the new context and enacting HRC
Generated code for runtime built healing entities	**Description**
HRC code	Java classes implementing the Reconfiguration Enactment Connector (HRC) behavior
HRC DP: HRC Deployment Package	Dynamically compiled and packaged Java classes of HRC
SSWS code: Stub for SWS code	Dynamically generated stub for SWS invocation by HRC
SSWS DP: Stub for SWS Deployment Package	Dynamically packaged Java classes for SWS stub deployment

Table 3. Description of HRC generation and deployment.

Reconfiguration Enactment Connector Behavior

```
for each IWS call :
{
   Param[]= GetParamfromHRVSOAPRequest(SOAPEnvelopeReq);
   // extract functional parameters from intercepted request messages
   ParamTypes[]=GetParamTypefromSWSWSDL(SWSWsdlUrl);
   //  extract functional parameter types from WSDL
   Stub= BuildStubforSWS();
   // Create stub for the currently bound concrete WS Provider
   ResSWS= InvokeSWS(Stub, Param[],ParamTypes[]);
   // invoke the currently bound concrete WS Provider
   SetResult2HRVSOAPResponse(ResSWS, SOAPEnvelopeResp);
   // substitute content of SOAP response message
}
```

Figure 4. The Reconfiguration Enactment Connector Operations

3.3 Experiments

In order to experiment our system in real scale conditions, we implemented the Conference Management System (CMS), a Web service-based networked application for the management of the "cooperative review" process. This Web service-based architecture aims at ensuring data exchange flexibility between system components, namely: Committee Chair, Reviewer, and Author. We developed a service-oriented architecture composed of 24 services, 18 tables to manage data, and about 9000 code lines. We are currently conducting large scale experiments for measuring QoS under the French Grid5000.

We have deployed our architecture on a set of 382 nodes of the Grid5000. After more than a day of application running, requesters have sent about 30 millions requests and the monitoring system has saved about 3 Gigabytes of QoS parameters values. In the following, we give an idea about measured values (in millisecond):

Execution Time	Communication Time
Min= 0	Min= 10
Max= 1044	Max= 45742
Avg= 1.217	Avg= 38.355

The execution time ranges between few ms and 1000 ms with an average of 1.2. The communication time ranges between few ms and several dozen of thousands (from 10 to 45700) with a big average of 38.3. With a first analysis, this shows the sensitiveness of such systems to communication context and demonstrates the importance of the communication-level preventive and reactive healing.

A second experiment is carried out while varying requester numbers. The measurement results show a real sensitiveness to the variation of requester numbers. The communication time varies between about 50 for 10 requesters to 1000 for 100 requesters. It increases highly with the number of requesters showing the importance of this parameters in response time observed by the requesters. Such information is being analyzed and modelled to support a correct monitoring and diagnosis for this application. Measurement details are summarized in table 4.

Client number	Request number	Communication Time	Execution Time
10	4953	Min= 15, Max= 553, Avg= 53.00	Min= 8, Max= 387, Avg= 29.45
50	5065	Min= 23, Max= 48167, Avg= 686.74	Min= 9, Max= 6472, Avg= 46.24
100	5386	Min= 32, Max= 70053, Avg= 848.99	Min= 11, Max= 7775, Avg= 108.98

Table 4. Experimental results.

4 Related Work

Different approaches propose reconfiguration to implement self-healing [7, 2, 8]. Some of them act at the communication level. They offer a third party for reconfiguration between the WS requester and provider, as mediator [2], or community [8]. Other works like [7], act at both: component level and communication level.

Authors of [7] propose to include a healing layer in components and connectors. Therefore, the self-healing connector contains two layers: a communication layer which manages exchanged messages, and a healing layer which reconfigures stub at this level. The self-healing component consists of a service layer and a healing layer which launches reconfiguration after an anomaly detection.

In [8], authors provide a solution for Web service substitution. They propose a "WS community" which regroups Web services having similar functionalities. This community is represented with an "Abstract WS Interface" which is a common interface for all alike Web services. The mapping interface between the Concrete WS Interface and the Abstract one are ensured dynamically by the Open Source Connectivity while respecting mapping rules.

Work in [2] proposes to group one or more Web services with QoS features inside a unique wrapper and to publish it as a standard Web service. Clients invoke this wrapper (called also virtual Web service) which is responsible for invoking real providers.

However, in these works, if the concrete Web service crashes, they can not perform the substitution, because they can not maintain a SOAP canal for response. In our work, we address requests to a Virtual WS which does not implement the WS interface. This may lighten the Virtual WS and minimize its crashes, and give us the possibility to re-route requests on the fly from one concrete WS to another.

5 Conclusion

In this paper, we presented a self-healing framework for QoS management in Web service-based applications. The framework has been implemented on top of reflective programming libraries. It relies on intercepting and handling the content of SOAP-level communication messages.

Our implementation has been assessed on top of the GRID5000 infrastructure with ad hoc application scenarios. It is now being used within the European WS-DIAMOND project for two case studies. The first case study addresses collaborative activities. It implements WS support for authoring and reviewing activities with discovering and publishing capabilities. We have already implemented this scenario and measured its QoS-enhanced traffic for complex interactions with multiple Web services. The analysis of Logs is now being conducted to tune the measuring and diagnosis processes that act upstream of the repair level. We are now working on the deployment of a second Web service-based application of the family of supplier chain management. This constitutes the second scenario of the WS-DIAMOND project. Our approach will be applied for communication-level healing management associated with service-level and flow-level healing management.

References

[1] S. A. Gurguis and A. Zeid. Towards autonomic web services: achieving self-healing using web services. In *DEAS '05: Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, pages 1–5, New York, NY, USA, 2005. ACM Press.

[2] J. A. J.F. Vilas and A. Vilas. An architecture for building web services with quality-of-service features. In *In 5th International Conference on Web-Age Information Management (WAIM 2004)*, 2004.

[3] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[4] D. A. Menascé. Qos issues in web services. *IEEE Internet Computing*, 6(6):72–75, 2002.

[5] D. A. Menascé. Response-time analysis of composite web services. *IEEE Internet Computing*, 8(1):90–92, 2004.

[6] P. Robertson and B. Williams. Automatic recovery from software failure. *Commun. ACM*, 49(3):41–47, 2006.

[7] M. E. Shin and J. H. An. Self-reconfiguration in self-healing systems. In *EASE '06: Proceedings of the Third IEEE International Workshop on Engineering of Autonomic & Autonomous Systems (EASE'06)*, pages 89–98, Washington, DC, USA, 2006. IEEE Computer Society.

[8] Y. Taher, D. Benslimane, M.-C. Fauvet, and Z. Maamar. Towards an Approach for Web services Substitution. In IEEE, editor, *10th IEEE International Database Engineering and Applications Symposium (IEEE IDEAS 2006)*, dec 2006.

[9] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma. Management applications of the web service offerings language (wsol). In *Advanced Information Systems Engineering, 15th International Conference, CAiSE 2003, Klagenfurt, Austria, June 16-18, 2003, Proceedings*, volume 2681 of *Lecture Notes in Computer Science*, pages 468–484. Springer, 2003.