A framework for monitoring and runtime recovery of web service-based applications

René Pegoraro, Riadh Ben Halima, Khalil Drira, Mohammed Karim Guennoun, João Maurício Rosário

▶ **To cite this version:**

René Pegoraro, Riadh Ben Halima, Khalil Drira, Mohammed Karim Guennoun, João Maurício Rosário. A framework for monitoring and runtime recovery of web service-based applications. 10th International Conference on Enterprise Information Systems, Jun 2008, Barcelona, Spain. pp.6. hal-00438852

HAL Id: hal-00438852
https://hal.science/hal-00438852

Submitted on 4 Dec 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A FRAMEWORK FOR MONITORING AND RUNTIME RECOVERY OF WEB SERVICE-BASED APPLICATIONS

René Pegoraro
Computer Science Department, São Paulo State University at Bauru, UNESP, Brazil
Laboratoire d'Analyse et d'Architecture des Systèmes, LAAS-CNRS, Toulouse, France
pegoraro@fc.unesp.br

Riadh Ben Halima, Khalil Drira, Karim Guennoun
LAAS-CNRS, University of Toulouse, 7 avenue de Colonel Roche, 31077 Toulouse, France
riadh.ben.halima@laas.fr, khalil@laas.fr, kguennou@laas.fr

João Maurício Rosário
Mechanical Design Department, University of Campinas, UNICAMP, Brazil
rosario@fem.unicamp.br

Keywords: Web services, Self-healing, Service Oriented Architecture, Quality of Service.

Abstract: Service provisioning is a challenging research area for the design and implementation of autonomic service-oriented software systems. It includes automated QoS management for such systems and their applications. Monitoring, Diagnosis and Repair are three key features of QoS management. This work presents a self-healing Web service-based framework that manages QoS degradation at runtime. Our approach is based on proxies. Proxies act on meta-level communications and extend the HTTP envelope of the exchanged messages with QoS-related parameter values. QoS Data are filtered over time and analysed using statistical functions and the Hidden Markov Model. Detected QoS degradations are handled with proxies. We experienced our framework using an orchestrated electronic shop application (FoodShop).

1 INTRODUCTION

Service-Oriented Architecture (SOA) is a collection of services that communicate with each other to execute business processes. A service in SOA technology is a functional building block, dynamically discovered and composed, loosely coupled, and reusable. With this in mind, many companies, like Sun (Mahmoud, 2005), IBM (New to SOA, n.d.), Oracle (Oracle Application Server, n.d.), focus efforts to Web service-based SOA.

W3C (Booth et al., 2004) defines a Web service as a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-process format, specifically Web Services Definition Language (WSDL). Other systems interact with the Web service in a manner prescribed by its description using Simple Object Access Protocol (SOAP) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

We may implement a SOA by using orchestration, choreography and other approaches, including an invocation of Web services as service component part of other Web service.

Web services are a well established technology. We publish, discover, and use them in a standard interface. But Web services may change their functional and non-functional aspects during runtime. These changes may engender the QoS degradation that influences negatively the business process.

There are many dimensions to measure QoS of Web services, Maximilien and Singh (2004) describe many class of them. Besides, many efforts have been made to specify QoS for Web services (Garcia, 2006, Ludwig, 2004, WSLA, n.d.), but these are not yet accepted as standards. Nevertheless, SOA style processes need to provide

levels of QoS and ways to become them more stable and reliable. As QoS parameter, we compute and experiment our framework with the Round-Trip Delay Time (RTT). It is the more representative QoS value for Web services, since from client's point of view, the RTT is the total time used in a complete Web service operation call, starting from sending the request until receiving the response.

This paper describes how we use the proxy mechanism to intercept messages. This mechanism allows the measurement of QoS and the execution of repair actions in Web services-based applications. These mechanism, enable our Self-Healing Architecture (SHA). It is inserted between Web services that allows monitoring and carrying out repair plans. The diagnosis is achieved using statistical functions and the Hidden Markov Model to predict system state under partial observation and probabilistic hypotheses. When suspecting a deficiency, HMM signals it to decide about the appropriate reconfiguration actions.

As an illustration, we deploy our SHA framework within the FoodShop application that is developed using BPEL orchestrated Web services (Web Services Business Process, 2007). The FoodShop is implemented in the framework of the European WS-Diamond Project.

The organization of this paper is as follows: Section 2 gives an overview of SHA. Section 3 discusses the architecture modules. Section 4 shows some experimental results. Section 5 concludes the paper.

2 SELF HEALING ARCHITECTURE FOR SOA

This architecture presented in figure 1, offers the resources for the monitoring, diagnosis and recovery for Web service-based applications. SHA monitors the interactions among Web services, identifies QoS degradation, plan for recovery actions and achieves them.

Figure 1 shows the component interactions within SHA. The Web service requester and providers are connected to the architecture through interfaces in Requester and Provider ports, respectively. Five main modules compose the SHA architecture:

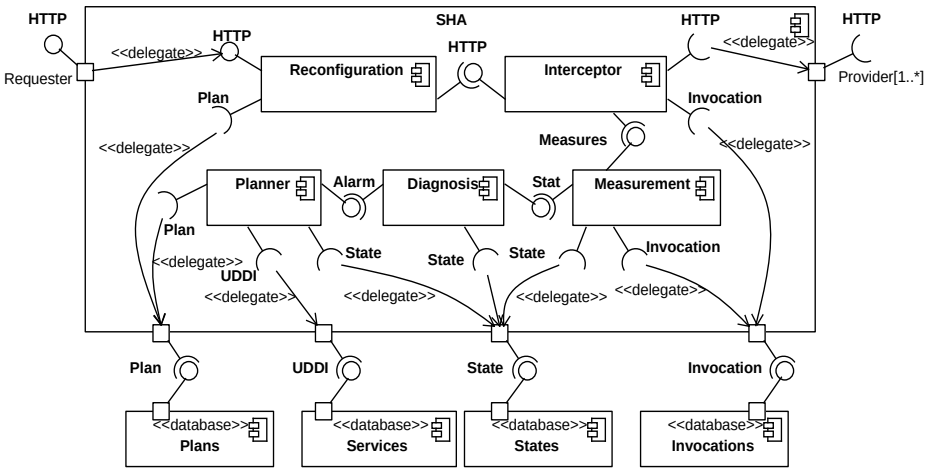

Figure 1 – Web service Self-Healing Architecture.

1. Interceptor – it intercepts messages between requester and providers to evaluate the QoS of each Web service operation;
2. Measurement – it computes statistical values from data generated by Interceptor for every executed Web service;
3. Diagnosis – it evaluates the data generated by Interceptor and Measurement modules to estimate the current state of Web services. If a state represents a misbehaviour QoS, Diagnosis sends an alarm to Recovery Planner Module;
4. Recovery Planner – it creates and stores recovery strategies, using the Diagnosis report;
5. Reconfiguration Manager receives requests and invokes Web services in agreement with plans created by Recovery Planner module.

To gather information from Web service interactions and to act over them, the communication links must be instrumented with

SHA. Ideally, instrumentation should be multi-platforms, simple to insert at Web servers, and with little side effects.

As enumerated in (Rud, 2006) many ways of instrumentations may be used in Web service environments. We choose HTTP proxy server because it is a technology well established in network, easy to implement, multi-platforms, and degradation is detected, all modules operate to perform the recovery strategy.

In Figure 2, *invoke* and *response* messages transport the packets HTTP with SOAP messages between requester and provider. These messages pass through Reconfiguration and Interceptor modules to provide reparability and diagnosticability to application.

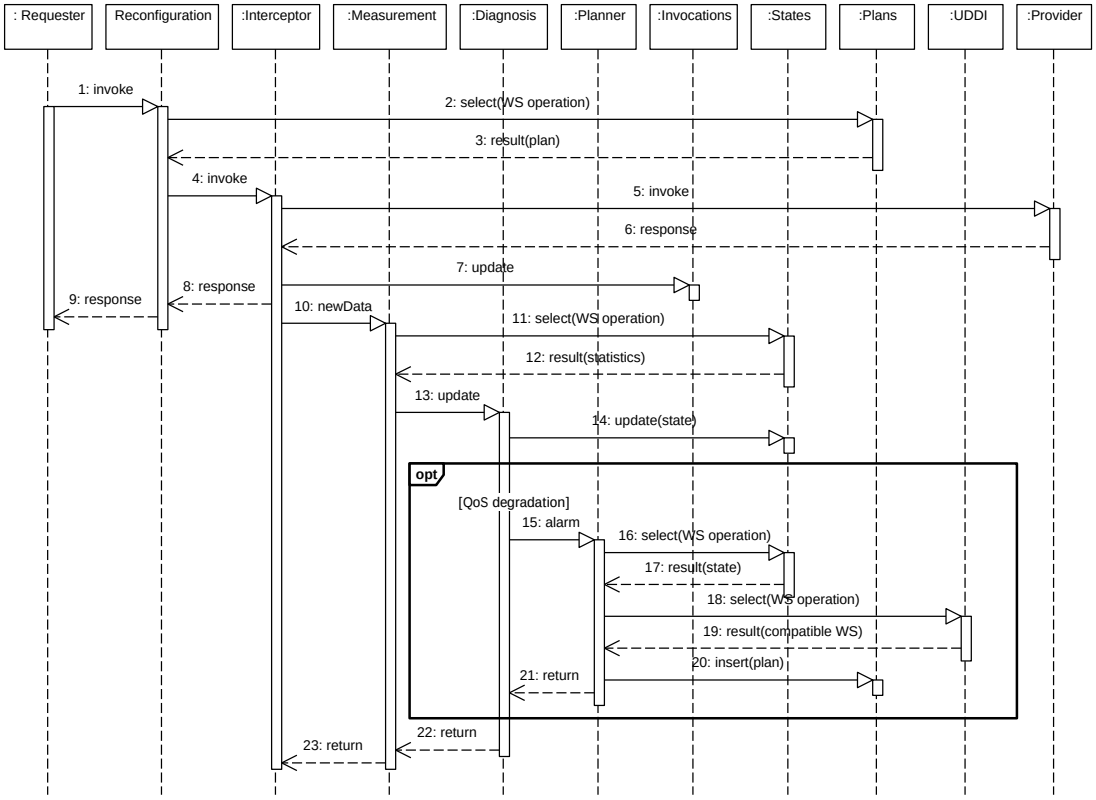

Figure 2 – Invocation of a Web service operation. The option combination fragment shows the steps to perform the recovery strategy.

possible to install in almost all Web application servers. Another motivation to implement SHA as a proxy server is that is also well situated to accomplish reparations plans, since it is capable to translate a request from a client in one or more different requests to different providers.

3 MODULES AND OPERATIONS

When the Web service-related QoS are computed and evaluated as acceptable values, the Repair Planner module remains stopped. The others modules work continuously to measure and verify the QoS parameters; and transmit the Web service messages. Figure 2 shows the communication inside the architecture. In the other situations, when QoS

3.1 SHA Modules

3.1.1 Interceptor Module

This module receives the request from a requester, measures the monitored QoS and sends it to Web service provider. When the Web service is finished, it receives the response from the provider, sends it to the requester, and finally updates *Invocation Log* with last QoS measures.

3.1.2 Measurement Module

In this section, we consider the RTT QoS parameter. We use the algorithms presented by Jacobson &

Karels (1988) and Karn & Partridge (1991) to TCP messages. Our implementation computes individually in each Web service the RTT average using the "smoothed" round-trip time estimate (SRTT). When a Web service operation is executed, the Interceptor module measures how long it takes, which represents the computed RTT values. With each new operation invocation, the Monitoring module computes the new $SRTT_i$ from equation (1).

$$SRTT_i = (1-\alpha).SRTT_{i-1} + \alpha.RTT_i \quad (1)$$

$$\begin{cases} \alpha = 1/i; & i \le N \\ \alpha = 1/N; & i > N \end{cases} \quad (2)$$

Where: RTT_i denotes the last RTT measured, i denotes the number of the last invocation of the Web service; $SRTT_i$ denotes an average approximation of RTTs; N denotes a constant that controls how rapidly the SRTT adapts to change.

To help us to identify QoS degradation, we consider two thresholds: Acceptable Round-Trip Time (ARTT) and Retransmission Timeout (RTO).

RTO is the maximum time that SHA expect for a response. To calculate the RTO we use the expression (3).

$$RTO_i = SRTT_i + K.\sqrt{\sigma_i^2} \quad (3)$$

$$\sigma_i^2 = (1-\alpha).\sigma_{i-1}^2 + \alpha.(SRTT_i - RTT_i)^2 \quad (4)$$

Where: i denotes the number of the last invocation; K denotes the constant that defines the premature timeout proportion; σ^2 denote the variance; and α as in equation 2.

The ARTT threshold, specified in equation (5), defines if the Web service QoS values is in concordance with its historic relevant values. Hence, if the RTT is greater than the ARTT, the Web service may be in trouble.

$$ARTT_i = (SRTT_i + RTO_i)/2 \quad (5)$$

3.1.3 Diagnosis Module

If this module diagnoses a state of QoS violation, it sends an *alarm* signal to Recovery Planner in order to prepare a plan to repair the process.

We consider three hypothetical states to model QoS behaviour of Web services:

1. Working: the service is normally working
2. Partially Working: the service shows some disagreements with the expected QoS. The service is still being used. However, at random, some requests will be duplicated to find other candidates that may offer better QoS.
3. NOT Working, the service does not work or frequently disagrees with expected QoS. This service will be substituted as soon as possible.

Normally Web services do not provide direct information about QoS states, but the variations observed on QoS suggest their current states; hence, we needed a model to estimate these states from observed QoS. We choose Hidden Markov Model (HMM) to model QoS behaviour of Web services. A HMM is a discrete-time stochastic model in which the system being modelled is assumed to be a Markov process with non-observable states, but variables influenced by the states are observable.

The HMM is defined as $< S, A, V, B, \pi >$, where:

- S is the set of states; S = {Working, Partially Working, NOT Working};
- A is the transition probability distribution among the states s_i to s_j,
  a_{ij} = Pr[s_j at $t+1$ | s_i at t];
- V is the set of observable variables; $V = \{v_{to}, v_a, v_{ok}\}$;
- B is the probability distribution of observe v_k being in s_j.
  $b_j(k)$ = Pr[observe v_k | s_j]; and
- π is the initial state distribution $\pi = \{\pi_1, ..., \pi_N\}$.

Where: the observable variables are: v_{to} if time-out is observed (RTO < RTT); v_a if RTT is acceptable, but is higher than expected (ARTT < RTT ≤ RTO); and v_{ok} if the RTT has QoS expected (RTT ≤ ARTT). We assigned estimated values for A and B based in empirical observations and expected state after specific behaviour of Web service invocations. State diagnosed and statistics information are updated in the *State log* in each Web service invocation.

3.1.4 Recovery Planner Module

This module uses the data brought by *alarm* signal and collected from *Invocation* and *State logs*. Recovery Planner gathers information to try to identify the better reparation strategy, and then, if possible, it creates a recovery plan and inserts it in the Plans database, as show in Figure 2. Each plan in database relates a Web service with a plan to correct it. After a plan is stored, every execution of the Web service will use the specific plan.

In the Recovery Strategies section, we present more details about recovery strategies plans.

3.1.5 Reconfiguration Manager Module

This module receives the requests and invokes requested Web services through Interceptor module.

Reconfiguration Manager Module is in charge of offering the healing capability allowing, when necessary, carry out the reconfiguration actions planned by Planner module. When a requester invokes a Web service, this module is the first to execute in SHA. Reconfiguration module receives

the request, and if there is a plan in database for this service, executes it. More specifically, Reconfiguration module receives the HTTP/SOAP message; translate all reference from original destination – server, service, and operation – to destination specified in plan and send this message to specified destination.

3.2 Recovery Strategies

In the context of this paper, recovery strategies are plans to try to sustain QoS level of a SOA process, using two types of recovery plans: *substitute* and *duplications*.

3.2.1 Substitution

The substitution recovery approach is suitable for recovery when exist a compatible Web service with acceptable QoS to substitute a Web service that presents QoS misbehaviours or QoS degradation that may lead the service to misbehaviour.

3.2.2 Duplication

When a Web service has not been showing acceptable values of QoS, but works even in a precarious fashion, we can use duplication to discover the QoS of others compatible Web services Ludwig (2004). In this scenario, we could invoke the current service and invoke in parallel a compatible service with unknown QoS. The Web service showing the best QoS after some invocations will be chosen to substitute the original. SHA implements duplication as double substitution, in fact, Reconfiguration module creates two threads, one to invoke the original service and another to invoke the new candidate to substitution. This mechanism uses the results of the fastest as response to requester; the other results are used just to update the statistical and probable states of each Web service.

4 EXPERIMENTAL RESULTS

To experiment SHA, we carry out two experiments, one that concern to time consuming by SHA and another in a Web service orchestration.

4.1 Time Consuming Inside SHA

When we insert SHA into a Web service application, the times consumed in message exchange between Web services enlarges. To identify the impact of SHA in an application, we made time measures in a simple Web service environment. Three similar computers in a local LAN composed the test: one client, one with the Web service, and one to accommodate SHA. The Client and SHA are implemented in Java 5. The client Web service just sums of two integers. It was deployed in Apache Tomcat 5 with Axis 1.4. Table 1 shows the RTT average of Web service invoked 5000 times in each situation. In Table 1, execution time increases when the SHA is hosted in the client computer and even more when SHA is alone.

Table 1: Average of RTT measures and their relations. The second line shows a deployment without SHA (base for relations).

Comput. 1	Comput. 2	Comput. 3	Time (ms)	Relation
Client		Web service	8.5	1.0
Client + SHA		Web service	15.6	1.8
Client	SHA	Web service	19.6	2.3

4.2 FoodShop

The food shop prototype used has become the standard test bed in the frame of Ws-Diamond Project and it involves characteristics, as asynchronous and synchronous invocations, compositions using BPEL, and simple Web services. The description following comes in Deliverable 1.1 from European WS-Diamond Project (IST-516933).

The food shop example is concerned with a FoodShop Company that sells and delivers food. The company has an online *Shop* and several *Warehouses* located in different areas that are responsible for stocking imperishable goods and physically delivering items to customers.

Customers interact with the FoodShop Company to place their orders. In case of perishable items, that cannot be stocked, or in case of out-of-stock items, the FoodShop Company must interact with several *Suppliers*.

4.3 Execution Environment, Deployment and Results

To illustrate and improve the developed framework, we deploy SHA within the FoodShop application. We deploy the FoodShop Web services in four virtual machines: one for the Shop, one for the Warehouse, and two for the Suppliers. The first Supplier is the currently used by the application process while the second is the substitute one. We deploy the client in a fifth virtual machine, and the

SHA in a sixth one. For this case, we choose to centralize the SHA, which centralizes the interaction management.

In order to test our framework, we inject delays in the execution time of a Supplier Web service. The Measurement module computes the new QoS values, and informs the Diagnosis module about the system state. The HMM-based diagnosis detects the QoS degradation and sends a report to the Planner Module, which generates a recovery plan. Reconfiguration module will use the plan and reroute next requests to the new Supplier thanks to the proxy.

In our experiments, we fix $K = 2$ for two raisons. The first is the Chebyshev Inequality (equation 6).

$$\Pr[RTO_i < SRTT_i + K\sigma] \geq 1 - \frac{1}{K^2} \qquad (6)$$

With K=2, we conclude that more than 75% of responses are valid responses (non-timeout). The second is on the basis on an already large scale monitoring experiment on the French Grid'5000 (www.grid5000.fr). The statistical study of the logged QoS values, shows that 96% of responses are valid with K=2.

5 CONCLUSIONS AND FURTHERS WORKS

In this paper, we presented a self-healing architecture that manages QoS in a Web service-based application. The presented approach relays on different modules for monitoring, diagnosis and recovering QoS degradation. We illustrate our approach with the FoodShop application. The first experiment is achieved while using virtual machines and we are working on a large-scale experiment under the Grid5000. We will focus on the distribution of the SHA while centralizing and coordinating the diagnosis and the recovery actions, based on the knowledge of the structural architecture of the applications and the dependencies between service invocations.

The recovery action requires a Web service with offer the same functionalities as the deficient one. Actually, we deal with a predefined list of similar and equivalent services. We need to improve this component while using ontology for specifying and searching the substitute services.

ACKNOWLEDGEMENTS

This work was supported by CAPES – Brazilian Council of Research and LAAS-CNRS, France, through collaboration research project CAPES-COFECUB.

REFERENCES

Booth, D., Haas , H., McCabe , F., Newcomer , E., Champion, M., Ferris , C., Orchard, D. (Eds.) (2004, February 11). Web Services Architecture. *W3C Working Group Note.* Retrieved March 22, 2007, from http://www.w3.org/TR/2004 /NOTE-ws-arch-20040211/.

Garcia, D. Z. G., & Toledo, M. B. F., A (2006). Web Service Architecture Providing QoS Management. In *Proceedings of the 12th Brazilian symposium on Multimedia and the Web*, Natal, Rio Grande do Norte, Brazil, 35–44.

Jacobson, V., & Karels, M. (1988, August). Congestion Avoidance and Control (revised). In *Proc. ACM SIGCOMM'88*, 314-329.

Karn, P., Partridge, C. (1991). Improving Round-Trip Time Estimates. In *Reliable Transport Protocols. ACM Trans. Comput. Syst.* 9(4), 363-373.

Ludwig, H. (2004). Web services QoS: external SLAs and internal policies or: how do we deliver what we promise? In *Proc. of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03)*, 115-120. Springer.

Mahmoud, Q. H. (2005, April). *Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI).* Retrieved November 21, 2007, from http://java.sun.com /developer/technicalArticles/WebServices/soa/.

Maximilien, E. M., Singh, M.P. (2004, September-October). A framework and ontology for dynamic Web services selection, In *Internet Computing, IEEE*, 8(5), 84- 93.

New to SOA and Web services (n.d.). Retrieved November 3, 2007, from http://www.ibm.com /developerworks/webservices/newto/.

Oracle Application Server 10*g* (n.d.). Retrieved November 12, 2007, from http://www.oracle.com /technology/products/ias/.

Rud, D., Schmietendorf, A., Dumke, R. (2006). Performance Modeling of WS-BPEL-Based Web Service Compositions. *IEEE Services Computing Workshops (SCW'06)* 140-147.

Web Services Business Process Execution Language v2.0 (2007, April 11). Retrieved November 20, 2007, from http://docs.oasis-open.org/wsbpel/2.0/OS /wsbpel-v2.0-OS.html.

Web Service Level Agreements (WSLA) Project (n.d.). Retrieved November 12, 2007, from http://www.research.ibm.com/wsla/.