Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 321 (2016) 113–127

www.elsevier.com/locate/entcs

Quality of Service Conflict During Web Service Monitoring: A Case Study

Jael Zela Ruiz[1,2] Cecília M. Rubira[3]

Institute of Computing
University of Campinas
Campinas, Sao Paulo, Brazil

Abstract

Web services have become one of the most used technologies in service-oriented systems. Its popularity is due to its property to adapt to any context. As a consequence of the increasing number of Web services on the Internet and its important role in many applications today, Web service quality has become a crucial requirement and demanded by service consumers. Terms of quality levels are written between service providers and service consumers to ensure a degree of quality. The use of monitoring tools to control service quality levels is very important. Quality attributes suffer variations in their values during runtime, this is produced by many factors such as a memory leak, deadlock, race data, inconsistent data, etc. However, sometimes monitoring tools can impact negatively affecting the quality of service when they are not properly used and configured, producing possible conflicts between quality attributes. This paper aims to show the impact of monitoring tools over service quality, two of the most important quality attributes - performance and accuracy - were chosen to be monitored. A case study is conducted to present and evaluate the relationship between performance and accuracy over a Web service. As a result, conflict is found between performance and accuracy, where performance was the most affected, because it presented a degradation in its quality level during monitoring.

Keywords: Web Services, SOA, Quality of Service, Quality Attributes, Conflict, Performance, Accuracy, Monitoring Tools.

1 Introduction

In recent years, the Web service technology has become the most popular and used technology to build SOA applications [26]. Web services are based in a set of protocols and standards as SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), and UDDI (Universal Description, Discovery, and Integration). Web services are distributed components which are self-contained, discoverable, reusable, composable, and have a transparent location [3]. As a result

[1] This work was supported by the Laboratory of Distributed Systems and Software Engineering, at Institute of Computing, UNICAMP, Campinas, Brazil.

[2] Email: jael.ruiz@students.ic.unicamp.br

[3] Email: cmrubira@ic.unicamp.br

http://dx.doi.org/10.1016/j.entcs.2016.02.007

1571-0661/© 2016 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

of its popularity, an increasing number of functionally similar Web services can be found on the internet [7], which entails to the service consumer to ask the question: "what are the better services?" or "which of them better fit my needs?" [6]. Service consumers have a difficult task to choose an appropriate service for their requirements. Quality of Service (QoS) has become the most appropriate criterion to distinguish non-functional characteristics between equivalent Web services.

QoS is described as a number of properties, named quality attributes, which take in play the Web service quality. Some of these attributes are, for instance, availability, throughput, robustness, and integrity where a set of quality attributes compose a quality model. Currently, there are several quality models proposed in the academia and in the industry [3] [6] [19] [20]. Web services promise quality levels based on quality models. A negotiation between the service provider and the service consumer is carried out, in order to assure a specific level of QoS for Web services. A Service Level Agreement (SLA) is the result of this negotiation, where quality is defined, negotiated and tasks to assure quality are established [16]. Nevertheless, afterwards an SLA is arranged for both parties, a new question is asked by service consumers: How can we be sure that the supposed QoS defined in the SLA is really satisfied?. As a consequence, monitoring tools emerge to control the Web service quality levels. Monitoring tools are based on quality model. They are used to capture, collect, filter, and analyse information from the Web service during runtime [8]. Currently, there are many monitoring tools which come from the research and the industry, such as Dynamo [4], Cremona [14], SALMon [1], WebInject [11], SOAP Monitor [2], Webmetrics Web Services Monitoring [18], FlexMonitorWS [10].

However, because of the dynamic and unpredictable nature of Web services [12], quality attributes can suffer variations in their values during runtime. The relationship among quality attributes can produce conflicts between them when they are monitored at the same time. For example, in a response time and throughput scenario, response time can be a better quality value when it is monitored in isolation than in parallel with throughput. The reason is because the Web service receives a larger number of requests, producing that the Web service takes more time to respond to the user. On the other hand, throughput is also affected, because a small number of requests are attended by unit of time, due to the required time to respond each request. These conflicts are produced mainly for scalability reasons, Web services can have many service consumers sending many service requests at the same time. Monitoring tools become a factor for quality attribute conflicts.

Monitoring tools can become a double-edged sword, because they are a useful QoS control tool, but they can become the principal reason for conflicts when they are not properly configured. They can turn out to be an intrusive agent for the Web service, creating a stressful environment. This is important to know what is being measured, where you are monitoring, how it is being monitored, and how frequently it is monitoring. An active monitoring not properly configured can overload the Web service and produce a breakdown in the values for response time, throughput, or availability to current consumers. In order to demonstrate the monitoring tools effects over the Web service quality, we present a case study to measure the quality

of service for two important quality attributes: performance and accuracy. In this case study, a Web service to deliver clinical results is monitored by two different monitors generated from the FlexMonitorWS Tool [9]. Quality values are compared from three scenarios: in isolation, in parallel with the other monitor, and in parallel with fault injection. A statistical comparison of the results is presented in order to demonstrate the produced quality degradation level and the significant difference between them.

This paper is organized as follows. In Section 2, we introduce to some underlying concepts about SOA, monitoring tools and FlexMonitorWS Tool [10]. In Section 3, we define a Quality of Service conflict and why it is produced. Section 4 reports a case study where a conflict scenario is evaluated. And finally, in Section 5, we provide the conclusions and future works.

2 Monitoring Tools and SOA

In this section we introduce some underlying concepts about SOA and monitoring tools, then we describe the effects of monitoring tools over Web services. Finally, we present FlexMonitorWS Tool [9], a monitoring tool for Web services used in this paper.

2.1 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural style widely used in distributed applications. Different functional units, services, are connected using standardized and well-defined interfaces [23]. SOA applications are dynamic, heterogeneous, distributed and autonomous.

SOA presents three primary roles: the service provider, the service consumer, and the service broker, as shown in Figure 1. The service provider defines a service in a WSDL file, it is published in the service broker using UDDI, so the service is discoverable to service consumers. Service consumers ask for the service to the service broker, this provides the WSDL file, and the service consumer consumes the service directly using SOAP [20].

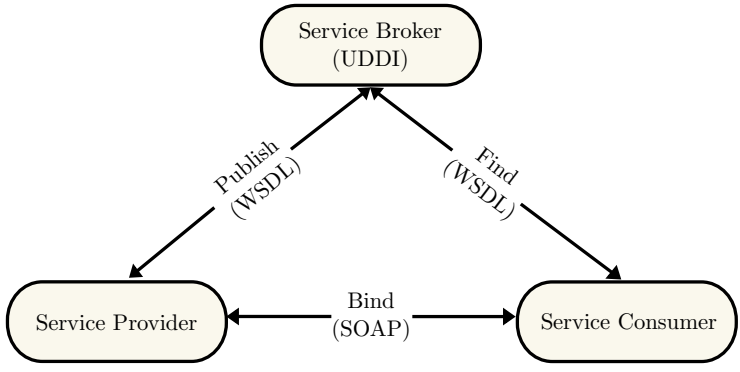


Fig. 1. SOA Architecture

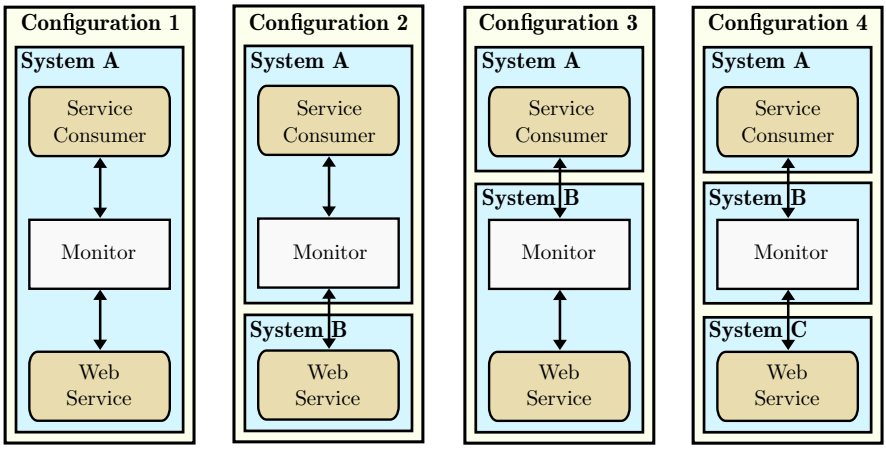


Fig. 2. Monitor configurations.

2.2 Web Service Monitoring Tools

Monitoring tools are systems that capture, collect, filter, and analyze information from a software system during runtime [8]. In Web services, monitoring tools are used for [9]: (1) improving the process of Web service selection and discovering, this enables QoS-based searches among functionally similar services. (2) self-healing technique such as dynamic adaptation and dynamic recuperation applied to some quality attributes (availability, scalability, capacity and reliability) when some of them not meet the desired level. (3) detecting violations in SLA, quality metrics based on SLA are used to evaluate and control the Web service.

Monitoring tools can be used according two strategies [6]. *Passive monitoring*, monitor is a sniffer and intercept the exchanged messages between the service provider and service consumer, with the aim of obtaining the QoS. In this strategy, a direct interaction with the service provider or consumer is minimized. *Active monitoring*, monitor sent service requests directly to the service provider, acting as a consumer. Monitoring systems can be configured differently according to three components [6], as shown in Figure 2.

- Configuration 1: The service consumer, the monitor, and the Web service are in the same system (System A).
- Configuration 2: The service consumer and the monitor are in the same system (System A), and the Web service is in another system (System B).
- Configuration 3: The service consumer is in a system (System A), and the monitor and Web service are together in another system (System B).
- Configuration 4: The service consumer is in the System A, the monitor and the Web service are in the System B and System C, respectively.

2.3 FlexMonitorWS Tool

FlexMonitorWS [10] is a Web service monitoring tool based on Software Product Lines (SPL). This tool is based on the creation of a family of monitors to monitor

different points in a Web service application and different quality attributes using different modes of monitoring. It was developed in Java language using FeatureIDE. FlexMonitorWS tool exploits the flexibility property by means of the creation of monitoring profiles which serve to a specific target and user requirements [9].

Monitoring profiles are built according to a feature model (Figure 3), the main features are (a) monitoring target, (b) quality attributes, (c) operation mode, (d) monitoring frequency, and (e) notification mode [9] [10]. The monitoring target specifies where the monitoring will take place: one can choose a *Web service*, *server application*, *server*, and/or *network*. Quality attributes indicate what needs to be monitored confirming what is sought, like *availability*, *performance*, *reliability*, *accuracy*, *robustness*, *hardware state*, *failures in the log file*, and/or *network QoS*. Operation mode establishes the strategy to be used, a passive monitoring by means of message *interception*, or active monitoring over *invocation* the service directly or by means of *inspection* of log files. Monitoring frequency can be *continuously* or *periodically*. Notification mode sets up the method to notify about the generated monitoring results, by sending a *message* or writing in a *log file*. According to the selected features is generated a product, a monitor (jar file), this is executed using a property file containing the Web service specification.

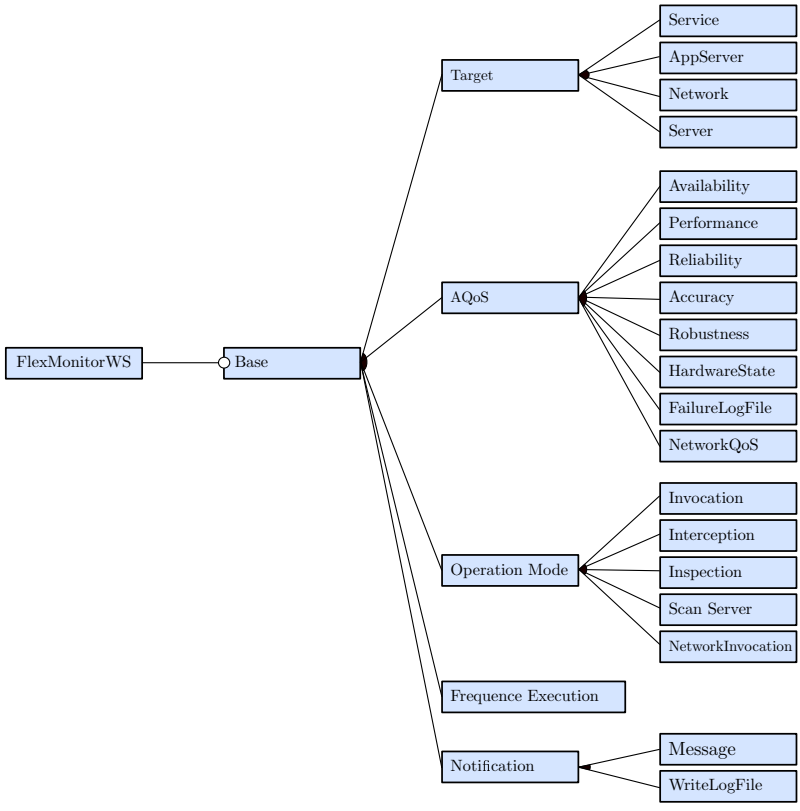


Fig. 3. FlexMonitorWS Feature Model

3 Quality of Service Conflict

The term 'conflict' has been defined as a common and inevitable phenomenon that arise in different contexts and levels [15]. In Requirements Engineering, Mairiza [15] defines conflict between Non-Functional Requirements (NFR) to determine interference, inconsistency or interdependence among requirements, existing a negative contribution of one NFR on another one [17], producing that two or more NFR cannot be satisfied at the same time.

Following the Mairiza's definition for conflict in NFR and since QoS attributes are NFR. We define QoS conflict when exists a negative contribution between two or more attributes, producing a degradation in the quality value for one or more attributes.

A degradation in the quality level is produced for different reasons, i) The Web services are dynamic and unpredictable, Web services are connected and disconnected in running time, QoS attributes can suffer variations in their values. ii) QoS attributes are subjective, relative and interacting [15]. Subjective because they can be viewed, interpreted, and evaluated differently by different people. Relative because the interpretation and the importance of the QoS attributes can vary depending on the system. Interacting means they tend to interfere, conflict or contradict with each other. iii) Insufficient resources. Constraints on the resources as CPU, memory usage, or network bandwidth [25]; these resources do not support the overload on the Web service. iv) Web service evolution, the constant evolution of the Web service to improve its functionalities can produce a possible conflict in its quality levels. v) Monitoring Tools. Depending on how monitoring tools operate over monitored systems, they can produce risk and problems over this last one. Many researchers have reported an intrusion problem of monitoring caused by monitoring tools.

This paper is focused on how Monitoring tool can generate conflict between quality attributes. Monitoring tools use different methods are used to extract and collect information from Web services. These methods are divided into three types: instrument method, interceptor method, and agent approach [25]. Instrument methods are used by testing techniques. In this method, monitoring code is embedded into the Web service implementation and it is inserted manually by the programmers (e.g. Javassist, AspectJ). Interceptor methods are used in the middleware, they get details about all sent and received messages to the Web service (e.g. Interceptor in CORBA, Handler in AXIS, JVMTI in JVM). This method is more independent, but it is executed in the same process with the Web service consuming the same resources. Agent methods are totally independent from the Web service, running in its own process consuming its own resources [25].

Methods bring intrusive effects to Web services in different degrees. When there are multiple monitors inserted in the target system, this becomes more complex. Instrument mechanisms make the target code difficult to understand and maintain [25]. While the interceptor mechanism can lead to performance decrease, because the monitor runs with shared resources. On the other hand, the agent approach

is the least intrusive method compared with the others, because it runs separately from the Web service.

4 A Case Study: DCTR System

In this section, we describe a case study to evaluate the relationship between performance and accuracy properties when they are monitored in a Web service over a Delivering Clinical Test Results System (DCTR System).

4.1 Object of Study

DCTR System is a Web service-based system developed in a clinical laboratory to deliver clinical test results to its patients. Web services were developed using Java language, JAX-WS API. The system offers several features encapsulated as Web services, such as *PatientService*, *DoctorService*, *ResultService*, etc. Among these Web services, we select the *PatientService*, because it is some of the Web services most used by the clinical staff when a new test is introduced.

The *PatientService* provides many features, between them, we have these two operations:

1. *getPatientName()*, which, given a patient code composed of an alphabetic part and a numerical part separated by a hyphen (e.g. PAT-0321), returns the corresponding patient's name.
2. *getPatients()*, which, given an integer number n, returns a list of the n recent attended patients.

Figure 4 shows a segment of the WSDL for *PatientService* to the operations *getPatientName()* and *getPatients()*.

4.2 Purpose and Goals

The aim of this case study is to identify the potential quality conflict between performance and accuracy during a Web service monitoring. For this study, a conflict is identified as a degradation in the quality value for one or both quality attributes. At the same time, it is pretending to discover the negative effects of the monitoring tools in the quality of service as the main cause of the conflict. We have selected the two operations in the Web service, *PatientService*, namely *getPatientName()* and *getPatients()*. Our research question has been formulated as follows:

RQ1: What are the performance and accuracy quality levels measured in isolation?.

RQ2: Is the accuracy quality level degraded when it is monitored in parallel with performance?

RQ3: Is the performance quality level degraded when it is monitored in parallel with accuracy?

```
<xs:complexType name="getPatientName">
  <xs:sequence>
    <xs:element name="patientCode" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="getPatientNameResponse">
  <xs:sequence>
    <xs:element name="return" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="getPatients">
  <xs:sequence>
    <xs:element name="max" type="xs:int"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="getPatientsResponse">
  <xs:sequence>
    <xs:element name="return" type="tns:patients" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="patients">
  <xs:sequence>
    <xs:element name="patient" type="tns:patient" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="patient">
  <xs:sequence>
    <xs:element name="birthDate" type="xs:dateTime" minOccurs="0"/>
    <xs:element name="emailAddress" type="xs:string" minOccurs="0"/>
    <xs:element name="firstName" type="xs:string" minOccurs="0"/>
    <xs:element name="icPatient" type="xs:long" minOccurs="0"/>
    <xs:element name="lastName" type="xs:string" minOccurs="0"/>
    <xs:element name="mobileNumber" type="xs:string" minOccurs="0"/>
    <xs:element name="monthOld" type="xs:int" minOccurs="0"/>
    <xs:element name="patientCode" type="xs:string" minOccurs="0"/>
    <xs:element name="phoneNumber" type="xs:string" minOccurs="0"/>
    <xs:element name="sex" type="xs:string" minOccurs="0"/>
    <xs:element name="status" type="xs:string" minOccurs="0"/>
    <xs:element name="yearsOld" type="xs:int" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Fig. 4. A segment of WSDL for *PatientService*.

The first question deals with the measurement of quality attributes separately; by answering this question we can know the quality level of the Web service, which will be used as a basis of comparison for our experiment. The second and third questions deal with the measurement in parallel of the quality levels for performance and accuracy; by answering these questions, we will be able to identify the potential conflict between these quality attributes during monitoring.

4.3 Quality Attributes

In this case study, we present two quality attributes, performance and accuracy. While performance is concerned with how quickly a service request can be completed, accuracy is concerned with whether the service response is correct. But, correct responses are not always produced quickly.

4.3.1 Performance

Performance of a Web service represents how fast a service request can be completed [13]. Performance can be measured in terms of throughput, response time, latency, execution time, and transaction time. Response time was selected for this work, because it is the main concern for both service consumers and service providers. It is a critical quality attribute because if a service consumer perceives a long delay after send a request to the service, the consumer is likely to change to another faster Web service [22].

Response Time is the required time to complete a Web service request [19] [13]; the time between sending a request to the Web service and receiving the response. The response time depends primarily on two factors: network delay and server side latency. Response time is measured by the following equation:

$$ResponseTime = T_{response} - T_{request} \tag{1}$$

Where $T_{request}$ is the time (timestamp) when the service request is sent to the Web service, and $T_{response}$ is the time (timestamp) when the service response is received from the Web service.

4.3.2 Accuracy

Accuracy is the level of accurate results that Web service can give to services requests [23]. It is measured by the number of errors (error rate) produced for the Web service over a period of time [9] [13]. Accuracy is concerned about the correctness of the service response, when accuracy value is close to one, it said to be accurate, if it is close to zero, the Web service is not accurate; so it loses credibility of its service consumers.

Accuracy is measured by the following equation:

$$Accuracy = 1 - \frac{nFaults}{totalRequest} \tag{2}$$

Where $nFaults$ is the number of errors returned for the Web service, and $totalRequest$ is the number of service requests sent to the Web service.

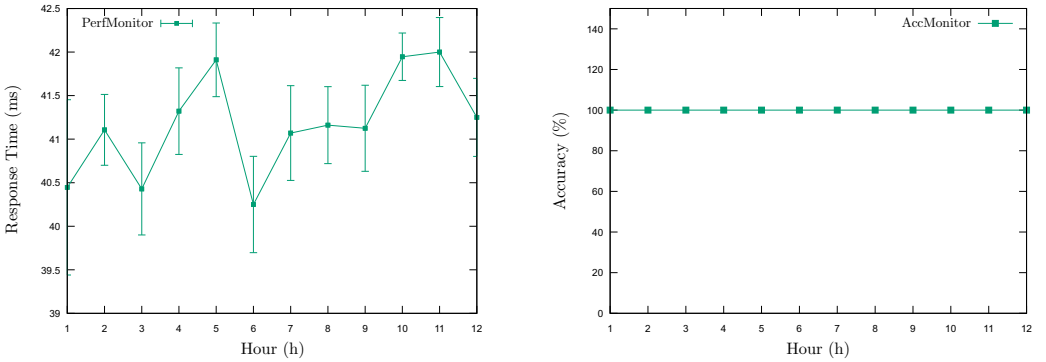
4.4 Monitors Generation

Two products (monitors) were generated using FlexMonitorWS Tool, with a different set of features. The first monitor called "PerfMonitor" was configured to monitor the performance attribute, selecting the following features of the feature model in Figure 3:

- Target: *Service* (operation: *getPatients()*)
- Quality attribute: *performance*
- Operation Mode: *invocation*
- Frequency: *30 seconds*
- Notification mode: *WriteLogFile*

The second monitor called "AccMonitor" was configured to monitor the accuracy attribute, selecting the following features of the feature model in Figure 3:

- Target: *Service* (operation: *getPatientName()*)
- Quality attribute: *accuracy*
- Operation Mode: *invocation*
- Frequency: *30 seconds*
- Notification mode: *WriteLogFile*



(a) Performance quality level executing "PerfMonitor".

(b) Accuracy quality level executing just "AccMonitor".

Fig. 5. Monitors executed in isolation.

4.5 Experiment Execution and Results Discussion

The experiment was executed in the following environment: Intel(R) Core(TM)2 Duo CPU 2.66 GHz processor, with 4.00 GB main memory, Windows 7 + SP1 system operating, and JDK 1.7. The monitors were hosted in this environment, whilst the Web service was hosted in an Apache Tomcat 7.0 server installed on a machine with the following configuration: AMD Phenom(tm) II P920 Quad-Core 1.60 GHz processor with 6,00 GB main memory, Windows 7 + SP1 as system operating, using JDK 1.8.

In order to respond to our research questions, the experiment was executed in these three different cases:

(i) "PerfMonitor" monitoring performance quality level over *getPatients()* operation.

(ii) "AccMonitor" monitoring accuracy quality level over *getPatientName()* operation.

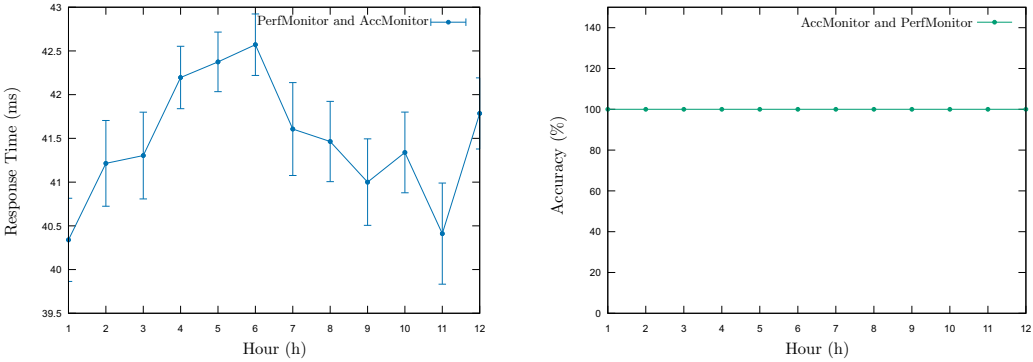
(iii) "PerfMonitor" monitoring performance and "AccMonitor" monitoring accuracy over *getPatients()* operation and *getPatientName()* respectively at the same time.

4.5.1 PerfMonitor Execution

"PerfMonitor" was executed over *getPatients()* operation of the *PatientService* Web service to measure its performance quality level during 12 hours. Figure 5(a) shows the average time by monitoring hour, time taken for the Web service to response a request from the service consumer. Responding to our research question **RQ1**, we notice that the response time has suffered an increase from 40 to 42 milliseconds, with an average of 41.467 milliseconds.

4.5.2 AccMonitor Execution

"AccMonitor" was executed over *getPatientName()* operation of *PatientService* Web service, to measure its accuracy quality level, during 12 hours. Figure 5(b) shows the average accuracy percentage calculated by monitoring hour. The accuracy



(a) Performance quality level.

(b) Accuracy quality level.

Fig. 6. "PerfMonitor" and "AccMonitor" executed in parallel.

remains 100% accurate over the entire time without suffering any fault, answering our question **RQ1**.

4.5.3 PerfMonitor and AccMonitor: Parallel Execution

"PerfMonitor" and "AccMonitor" were executed over *getPatients()* and *getPatientName()* operations, respectively, on *PatientService* Web service, in order to measure its performance and accuracy quality levels, during 12 hours. Figure 6 shows the average time to response a request and the average accuracy percentage by hour, when both response time and accuracy are monitored in parallel. It is observed that the response time attribute increases with regard to the previous execution. The time to response was between 40.4 and 42.5 milliseconds.

Responding to our research question **RQ2**, one can argue that Accuracy quality level remains unchanged during the isolated monitoring and with performance monitoring in parallel. In both cases the Web service was 100% accurate. This guarantees its correct operation.

On the other hand, the performance quality level is affected when it is executed with accuracy monitor in parallel. Responding to our research question **RQ3**, we found a decrease of the quality value in 0.259 milliseconds in the performance. Figure 7 shows the comparison by hour between these two executions. In order to support the difference in the results, we have assessed the statistical significance between "PerfMonitor" in isolation and "PerfMonitor" with "AccMonitor" results by means of a per-query paired t-test with 95% of confidence. The results of paired t-test confirms that the difference in performance is statistically significant. The Web service presents statistically a better performance when it is monitored just for "PerfMonitor".

4.5.4 AccMonitor with Fault Injection: PerfMonitor and AccFaultMonitor

In order to perform some dependability measures and collect evidences our assumption about the strong accuracy of *PatientService*, A third monitor was generated using fault injection, "AccFaultMonitor". XML injection [24] was used to generate interface faults [5]. We have used two kinds of injection: Parameters

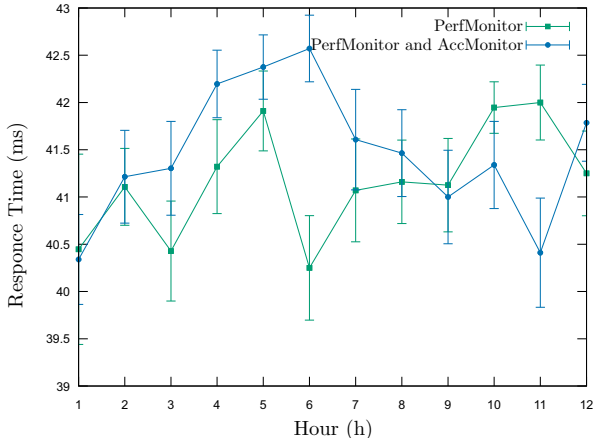


Fig. 7. Comparison of Performance quality level .

corruption injection and structure corruption injection. For example, we have corrupted the patient code sent to *getPatientName()* operation as follows:

Fault 1:
`<arg0>PAT-0239</arg0>` to `<arg0>9320-TAP</arg0>`

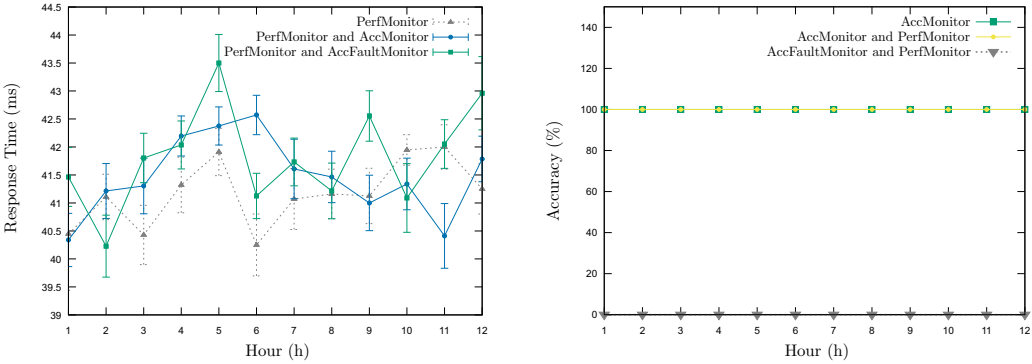
We have also corrupted the XML structure of the request, inverting opening and closing XML tags as follows:

Fault 2:
`<arg0>PAT-0239</arg0>` to `</arg0>PAT-0239<arg0>`

"AccFaultMonitor" was executed by 12 hours in parallel with "PerfMonitor". The parameters corruption injection was executed in the first 4 hours, structure corruption injection for the next four hours, and in the last 4 hours, both parameter and structure corruption injections were executed.

Figure 8(a) shows the average time by hour executed in parallel with "AccFaultMonitor". Performance continue suffering a degradation in its quality level as in the previous experiment. Paired t-test between "PerfMonitor" in isolation and "PerfMonitor" with "AccFaultMonitor" in 95% confidence confirms that the Web service present statistically a better performance when performance is monitored in isolation. This is because the Web service takes more time trying to interpret a corrupted request, holding the processor for more time. This situation can reduce the amount of resources needed for "PerfMonitor", and therefore taking longer to respond.

Figure 8(b) shows a comparison of the accuracy quality level in all tested scenarios. The calculated percentage for accuracy with fault injection is also displayed by monitoring hour. Corruption of the parameter values produced `java.lang.NullPointerException`. This can be a reasonable behavior because invalid data was sent to the Web service, but, on the other hand, it is not a good



(a) Performance quality level in all scenarios.

(b) Accuracy quality level in all scenarios.

Fig. 8. Performance and Accuracy quality levels in all scenarios

response, because it is not an adequate response to the service consumer. Structure corruption faults were all detected by the Web service, and it replied immediately rejecting as a malformed SOAP message.

5 Conclusions

The quality of a service is out of control of whoever use the service and the behavior of a Web service can change at runtime. Service consumer can not be aware about these changes. Monitoring tools become an important mechanism for aware quality and functional changes in a Web service. At the same time, monitoring tools can become an intrusive agent for the quality of Web service. When it is pretended to monitoring a Web service, we need to identify: what is our monitoring target? what do we need to monitor?, how monitor it?, how often monitor it? and how notify the monitoring results?.

It is necessary to pay attention to the monitor configuration for Web service monitoring, because it can be the main reason for quality level degradation of Web services. Instrumentation methods can bring more negative effect in quality degradation, and agent methods are less negative in the quality of service. On the other hand, Active monitoring (invocation) is an operation mode which produces quality level degradation in Web services.

Our study has shown the relationship among response time and accuracy, and conflict has been found between these two quality attributes when they are monitored at the same time. Performance is the most affected quality attribute, because a greater time is needed to response a service request when the accuracy is monitored at the same time, the Web service server receives a higher number of requests. Statistic tests confirmed this scenario. On the other hand, accuracy was not affected, it remained unchanged all the time in both cases, monitoring in isolation and with performance monitor in parallel.

Injection faults were added to the accuracy monitor to confirm the accuracy of *PatientService*. Parameter and structure corruption injections were not accepted by the Web service, although the exceptional responses were not adequate to the service

consumers, it shows that our case study is highly accurate. However, performance needed more time to respond to every request, decreasing, even more, its quality level.

References

[1] Ameller, D. and X. Franch, "Service level agreement monitor (salmon)", Seventh International Conference on Composition-Based Software Systems (2008), 224–227.

[2] Apache Software Fundation, "Web/HTTP Test & Monitoring Tool" (2011), URL:http://axis.apache.org.

[3] Balfagih, Z. and M. F. Hassan, "Quality model for web services from multi-stakeholders perspective", Proceedings of the 2009 International Conference on Information Management and Engineering, ser. ICIME 09 (2009), 287-291.

[4] Baresi, L. and S. Guinea, "Towards dynamic monitoring of ws-bpel processes", Proceedings of the Third International Conference on Service-Oriented Computing, ser. ICSOC05 (2005), 269–282.

[5] Bessayah, F., A. Cavalli, W. Maja, E. Martins, and A. Valenti, "A fault injection tool for testing web services composition", Testing Practice and Research Techniques **6303** (2010), 137–146.

[6] Cabrera, O. and X. Franch, "A quality model for analysing web service monitoring tools", Research Challenges in Information Science (RCIS), 2012 Sixth International Conference (2012), 1–12.

[7] Choi, C. R. and H. Y. Jeong, *A broker-based quality evaluation system for service selection according to the qos preferences of users*, Information Sciences **277** (2014), 553–566.

[8] Delgado, N., A. Gates, and S. Roach, *A taxonomy and catalog of runtime software-fault monitoring tools*, IEEE Transactions on Software Engineering **30** (2004), 859–872.

[9] Franco, R. J., "FlexMonitorWS: uma solução para monitoração de serviços Web com foco em atributos de QoS", Masters thesis, Institute of Computing, University of Campinas, Campinas, Sao Paulo, Brazil, 2014.

[10] Franco, R. J., C. M. Rubira, and A. S. Nascimento, "FlexMonitorWS: uma solução para monitoração de serviços Web com foco em atributos de QoS", Congresso Brasileiro de Software: Teoria e Pratica, CBSoft2014, 21th Sessão de Ferramentas **2** (2014), 101–108.

[11] Goldberg, W., "Web/HTTP Test & Monitoring Tool", (2011). URL:http://www.webinject.org.

[12] Ladan, M. I., "Web services metrics: A survey and a classification,", 2011 International Conference on Network and Electronics Engineering IPCSIT **11** (2011), 93–98.

[13] Lee, K., J. Jeon, W. Lee, S.-H. Jeong, and S.-W. Park, "QoS for Web Services: Requirements and Possible Approaches", W3C Working Group Note 25 (2003), URL:http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/.

[14] Ludwig, H., A. Dan, and R. Kearney, "Cremona: An architecture and library for creation and monitoring of ws-agreements", Proceedings of the 2nd International Conference on Service Oriented Computing, ser. ICSOC 04 (2004), 65–74.

[15] Mairiza, D., D. Zowghi, and N. Nurmuliani, "Managing Conflict among Non-Functional Requirements" 12th Australian Workshop on Requirements Engineering (AWRE 2009), Sydney, Australia, **12** (2009).

[16] Metzger, A., S. Benbernou, M. Carro, M. Driss, G. Kecskemeti, R. Kazhamiakin, K. Krytikos, A. Mocci, E. Di Nitto, B. Wetzstein, and F. Silvestri, "Analytical Quality Assurance" Service Research Challenges and Solutions for the Future Internet **6500** (2010), 209–270.

[17] Moreira A., Rashid A., and J. Araujo, "Multi-dimensional separation of concerns in requirements engineering,", 13th IEEE International Conference on Requirements Engineering (RE '05), Paris, France, 2005.

[18] Neustar Webmetrics, "Web/HTTP Test & Monitoring Tool" (2011), URL:http://www.webmetrics.com.

[19] Oriol, M., J. Marco, and X. Franch, *Quality models for web services: A systematic mapping*, Information and Software Technology **56** (2014), 1167–1182.

[20] Oskooei, M. A. and S. M. Daud, "Quality of service (QoS) model for web service selection", Computer, Communications, and Control Technology (I4CT), 2014 International Conference (2014), 266–270.

[21] Oskooei, M. A., S. B. M. Daud, and F. F. Chua, "Modeling quality attributes and metrics for web service selection,", AIP Conference Proceedings **1602** (2014), 945–952.

[22] Rajamony, R. and M. Elnozahy, "Measuring client-perceived response times on the WWW", Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems **3** (2001).

[23] Rajendran, T. and P. Balasubramanie, *Analysis on the study of qos-aware web services discovery*, Journal of Computing **1** (2009), 119–130.

[24] Salas, M. I. P. and E. Martins, *Security testing methodology for vulnerabilities detection of xss in web services and ws-security*, Electronic Notes in Theoretical Computer Science **302** (2014), 133–154.

[25] Wang, Q., Y. Liu, M. Li, and H. Mei, "An online monitoring approach for web services", Computer Software and Applications Conference COMPSAC 2007. 31st Annual International **1** (2007), 335–342.

[26] Zheng, Z., Y. Zhang, and M. Lyu, *Investigating qos of real-world web services*, IEEE Transactions on Services Computing. **7** (1959), 32–39.