An Open Source Monitoring Framework for Enterprise SOA

Nabil Ioini, Alessandro Garibbo, Alberto Sillitti, Giancarlo Succi

An Open Source Monitoring Framework for Enterprise SOA

Nabil El Ioini[1], Alessandro Garibbo[2], Alberto Sillitti[1], Giancarlo Succi[1]
[1]Free University of Bolzano, Italy
[2]Selex-ES, Italy
{nelioini, asillitti, gsucci}@unibz.it, alessandro.garibbo@selex-es.com

Abstract. Web services monitoring is currently emerging as an effective way to trace faults in services at runtime. The lack of testing information provided by web services specifications was an indication that other methods need to be used to assess the quality of web services. This is mainly due to the fact that it is difficult to simulate the client infrastructure during testing of web services. Monitoring consists of inspecting services at runtime and taking adequate actions when unacceptable events occur. Monitoring could be performed by different stakeholders and could target different properties of services. Predominantly, monitoring is performed by service providers to manage their internal resources and balance their requests load. In our effort to improve the monitoring infrastructures, we propose a monitoring framework in which all the participants (services providers, services requestors) can contribute to monitoring and at the same time have direct access to the monitoring data. This paper describes a monitoring framework developed as part of NEXOF-RA[1] project. The framework offers a set of capabilities for a collaborative monitoring of web services. The paper presents motivations, system design, implementation and usage of the framework.

1 Introduction

To address the business needs of today's organizations, Web Services (WS) emerge as an enabling technology that allows building flexible systems that integrate multiple pieces of software into one system [12]. Contrary to traditional software development paradigms, where software vendors have to agree on the communication protocols and the interfaces to use, web services improve the collaboration of different software vendors to build the final product by using standardized protocols and interfaces. Nowadays, the effort to push the adoption of web services is twofold. From one side research communities and companies support WS by providing tools that help develop and manipulate web services, and on the other side standardization bodies supply standards to regulate the use of WS [6][7][8][13].

[1] NEXOF-RA project http://www.nexof-ra.eu/

The concept of trust in the domain of services is an important issue, since many services are bound at run time, and the correct behavior of the services is not known a priori [9]. Being able to give high confidence about the behavior and the quality of a service at run-time can have a great influence on service compositions.

A big challenge with WS is how to trust the claims of services' providers about the quality of their services [14]. In web services, we delegate part of our business logic to an external provider to do it for us. Thus, we have no control of what could happen during the execution of that part of the business logic. One solution to increase customers' trust in the provided services is to provide a monitoring framework where all the stakeholders that take part of a SOA infrastructure are involved in the monitoring process. This way all the stakeholders can collect and use the monitoring data to make decisions about which services to use.

The contribution of this work is twofold. First we are providing a monitoring framework in which the stakeholders have the possibility to specify their monitoring requirements and have access at runtime to the collected data, however, this requires them to adhere to certain practices and collaborate to monitor the used WS. The second contribution is that we are combining the monitoring from the service requestor and service provider point of views. This is done by collecting data from both sides and allow both parties to make use of the data collected by the other party.

The rest of the paper is organized as follows. In section 2 we presented the related work. Section 3 presents an overview of the system architecture. Section 4 discusses the implementation details. Section 5 and 6 show a demonstration scenario and experiments and section 7 concludes and presents future work.

2 Related Work

Existing monitoring techniques differ in many aspects. The main properties that distinguish them from each other are the stakeholders involved, data collection techniques, the degree of invasiveness and the monitoring requirement specification. The spectrum of approaches proposed in the literature spreads along all these dimensions, however, not all the approaches take in consideration all the dimensions.

In the literature many approaches resemble to ours such as SCALEA-G [10], which is a unified monitoring and performance analysis tool for the grid. SCALEA-G uses a different architecture but has similar functionality. IBM Tivoli [11] uses a special integration bus to collect monitoring data so that all the messages passing through the bus are captured. In [4, 5] an approach for monitoring BEPL workflows is presented. A central execution engine intercepts all the passing events and stores them in an event dataset. The probes to be monitored are defined using event calculus.

The main difference between these approaches and ours is that we are providing an approach conformant to a bigger infrastructure, which is the NEXOF-RA. Although the techniques we used might not be new but our goal is to have a framework compatible with the goal of NEXOF-RA, which is establishing strategies and policies to improve delivery of applications and enabling the creation of service

based ecosystems where service providers, service requestors and third parties easily collaborate.

3 System Overview

Traditionally system verification and validation (V&V) is a pre-deployment exercise designed to assess the capabilities of systems before putting them into production. In the paradigm of WS and SOA based systems [9], the dynamic nature of these systems requires the extension of the V&V setting to include run-time quality assessment that cannot be applied before deployment.

To ensure the quality of Web Services, we propose a monitoring framework that enables runtime data collection to automatically identify anomalous events defined by any of the stakeholders, aggregates related events and presents data to the interested parties with different levels of granularities.

The framework adopts a proxy-based strategy to collect monitoring data (Figure 1), where each participant in the interaction needs to use a proxy that allows the interception of the messages coming in and out. The framework relies on different receptors implemented inside the proxies to filter out the events of interest and report them to the monitoring server.

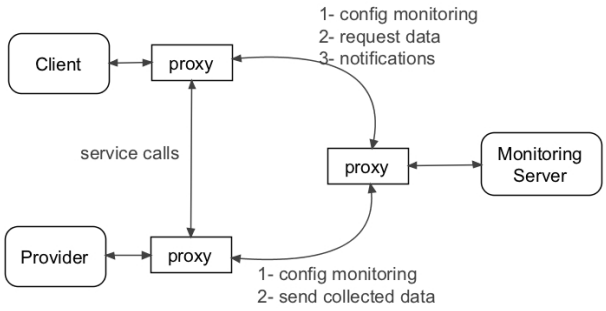

Fig. 1. Proxies for data collection

The framework provides an environment where registered services can be monitored automatically and according to users requirements. The monitoring model is described in the NEXOF-RA Monitoring SOA Enterprise pattern [2]. The main idea of the model is allowing all authorized users to perform two types of monitoring options: push and pull. The push monitoring allows users to register the events of interest to be monitored; if those events occur, all the registered users are notified. The pull monitoring instead, allows authorized users to request log data concerning a specific service.

1.1 Design choices

The design choices have been shaped to satisfy the requirements defined by the ESOA pattern [2]. Four attributes were given priority.

1. Maintainability: in an infrastructure that meant to be used in a SOA environment, maintainability is an important factor due the size and complexity of the interacting systems. By decomposing the framework into sub-components, maintainability becomes easier [15]. Each of the sub-components is responsible for a precise set of operations, which are exposed as interfaces to communicate with the rest of the framework. The framework itself looks as a component (a service) from outside, and it communicates with the other components (e.g. proxies collecting data) through a set of pre-defined interfaces (Web Services calls). Therefore, any part of the framework could be updated or modified without affecting the rest of the sub-components.
2. Availability: In a SOA environment, it is critical to understand the availability of the services, mainly because the usual everyday operations may depend on services, which for some reason could be unavailable at some time slots. To this end, the framework needs to be able to monitor all the services that take part of the ESOA environment including the framework itself. As we mentioned before, the framework is built as a set of services, and this allows the framework to monitor its availability.
3. Performance: A drawback of the framework in terms of performance is the overhead generated by the proxies. We have decided to use a proxy-based architecture for collecting data. However, before doing that we have considered other options mainly:
    a. Monitoring aware middleware [1], relying on a collecting data from interceptors implemented as part of the middleware. This technique has the advantage of being able to monitor detailed information about the services e.g. resources usage, since the interceptors are part of the infrastructure in which services are deployed. However, it imposes a limitation in that it requires all the participants to use the same middleware, in case their existing middleware does not have the monitoring capability, which could have a high cost.

    b. Central proxy server relying on a central proxy, which can be used by all the participants. This technique represents several challenges. The first one is that it cannot capture precise information about the quality of service. For example, the response time that is captured by the proxy will represent the response time from the proxy to the service and not from the client to the service. The second challenge is that internal quality attributes of the services cannot be captured such as the time needed by the service to process the request internally. For this reason we have decided to choose a proxy-based approach,

which does not require the participants to alter their environments as well as it has access to more accurate information, since the proxy is considered to be part of the participants' infrastructures. Of course, there is a cost to pay in terms of the overhead generated by the proxy, however, compared to the advantages we gain, we consider it to be acceptable. Furthermore, the framework offers two strategies for monitoring to deal with how much overhead is generated. The push approach requires much more overhead since each component is required to actively report any event of interest to a specific client. For example, a client requirement might be to be notified if the throughput of the monitored service exceeds 5 ms. In this case the services actively filter out all the incoming requests and in case of an event that exceeds 5 ms, it needs to report it to the monitoring server. The second approach is called the pull approach, and this is a more relaxed approach, in which the service collects the data locally and it is the monitoring server that asks for it when needed.

4. Adaptability: By having a well defined interfaces and separation of concerns, the framework could be adapted and reused in different contexts. Our framework is used to monitor Web Services, however, other types of systems could be used as well. For example, the proxies can be adapted to monitor different types of components instead of Web Services, and communicate the monitored data to the monitoring server.

4 Implementation

The framework architecture features two main components, namely, the monitoring server and the monitoring proxy (Figure 2). The monitoring server provides facilities to store the collected data and serve it to the users when they need it. It also handles notifing the users in case of anomalous events. The current implementation of the monitoring server is done as a web application, and all the interactions with the outside components are done through web services. The monitoring proxy is a Java application that is used by all the participants of the monitored environment. We have extended an existing open source project called membrane router[2] as the monitoring proxy.

4.1 Monitoring Server

It is the central component of the monitoring server. It is implemented as a J2EE application running on top of Apache Tomcat container. The monitoring server

[2] Membrane Router, http://www.membrane-soa.org/soap-router-doc

offers four operations and requires four operations from the components that need to be monitored.

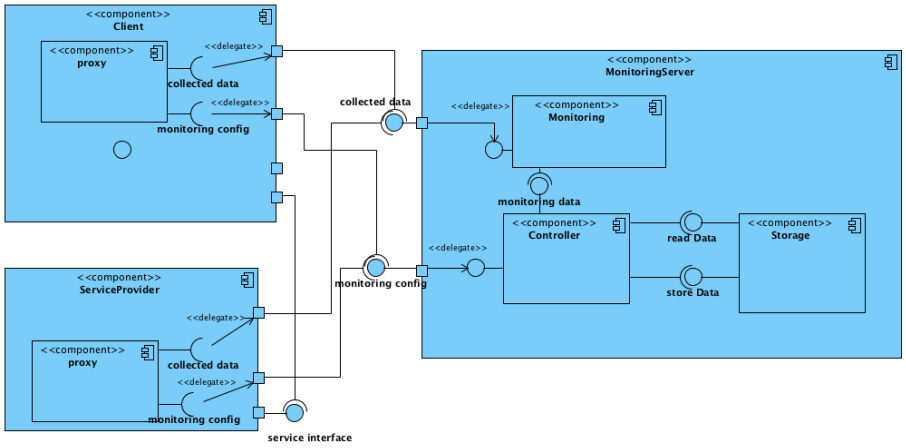

Fig. 2. Architecture overview

- Required operations:
  - ConfigureMonitroingPolicy: sets the monitored component configuration policy such as time slot before sending data to the server;
  - GetLoggedData: pulls all the data from monitored component;
  - IsAuthorized: requests the monitored component if the user who is trying to access monitoring data has authorization to do so;
  - SubscribeEvent: subscribes a new event to be monitored;
  - NotifyEvent: notifies the users who have subscribed to the different events.
- Offered operations:
  - ConfigureMonitringPolicy: enables the users of the monitoring server to set their policy preferences;
  - GetMonitoringData: provides the user with the possibility to pull the monitoring data from the server;
  - SubscribeEvent: allows to specify the events they are interested in;
  - notifyEvent: allows every user (customers and providers) to send a notification to the monitoring server using this interface.

4.2 Monitoring Proxy

The role of the proxy is to collect data from the different participants including service providers, service users and the monitoring server. The data is sent to the

monitoring server for storage and analysis. In the current implementation we are using membrane router as the monitoring proxy.

Membrane router is composed of three main parts: 1) the EndpointListener, which waits for the incoming messages 2) the EndpointSender, which sends the messages to their destination and 3) a set of interceptors in between the two end points. The part that we are mostly interested in is the interceptors. In the current implementation we have two interceptors to capture response time and throughput (Figure 3).

The interceptors implement two functionalities:

1. Intercept input messages: this function captures the input messages and adds new headers to them such as the sender id and timestamp
2. Intercept output messages: when a message passes through the router and is processed by the service provider, the provider sends back the response. The response message is captured again and, the headers that have been set in the input message are checked. A practical example that we have used is to calculate the throughput by checking the time elapsed between receiving the input message and receiving the output message.

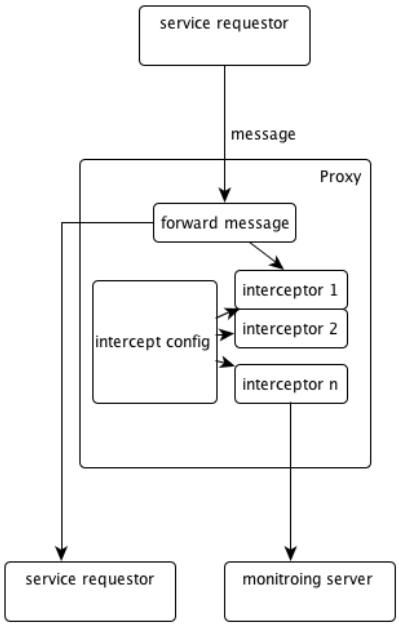

Fig. 3. The proxy architecture

4.3 Data Collection

As we mentioned before the monitoring framework offers two monitoring strategies with a different level of invasiveness and time to react to the events recorded.

The push approach is a timely monitoring strategy, which relies on external data source about the data to collect. The external data source is the set of configurations submitted by the stakeholders interested in monitoring the service. These configurations take the form of a simple triple of the form:

<Property, condition, value>

where the property means the property to monitor such as response time, throughput, the condition are Boolean expressions (e.g. >, <, =). In the current implementation the Boolean expressions are used to compare the monitored values against. However, to avoid imposing on any specific format for defining the monitoring requirements, it is up to the interceptors' developers to decide how their interceptors receive the monitoring requirements.

The degree of invasiveness has a great impact on the quality of the collected data. The more invasive the technique is, the more data can be collected. Our current implementation is limited in this side, because everything that we collect starts when the service requestor or provider submit the request/response to the proxy. So everything that happens before that such as the internal state of services is not captured. This limitation can be addressed by increasing the level of invasiveness such as the case of [13], but a tradeoff needs to be made between the data collected and the implementation cost.

5 Demonstration Scenario

Several scenarios have been considered and tested. In this demonstration, we focus on the two monitoring strategies, namely, the push and the pull. To use the framework some assumptions are needed. We assume that every participant has built the interfaces required to interact with the monitoring server. We also assume that all the participants have deployed the monitoring proxy in their environments.

In the pull strategy, the process is initiated by a user or a group of users setting their configuration policy for some specific service. Once this is done, the monitored service starts collecting the required events. At any point of time, users who have registered to that specific service can make an implicit call asking the monitoring server to pull the data that has been collected so far. This approach has the advantage of instantly showing how the service is performing. Figure 4 presents a sequence diagram of this approach.

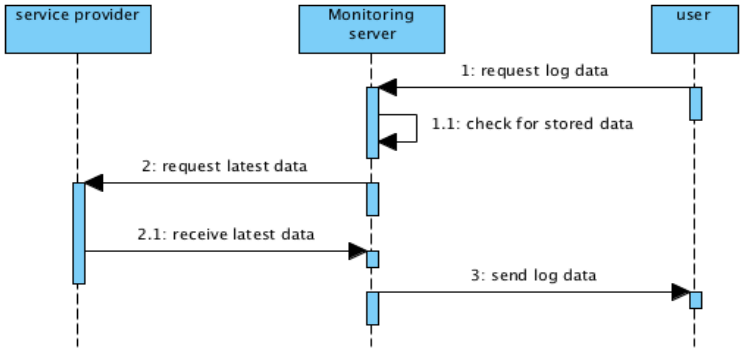

Fig. 4. The pull approach

The push approach on the other hand is initiated by the user registering to a specific event. The monitoring server forwards the registration to the service of interest. The service keeps track of all the users and their events of interest, and once the event occurs, the monitoring server is asked by the service to notify the respective users. The monitoring server is then responsible for notifying the users. The sequence diagram in Figure 5 shows the sequence of activities of the push approach.

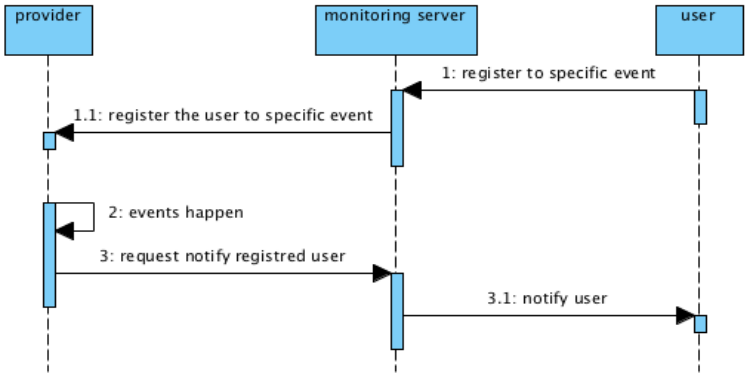

Fig. 5. The push approach

6 Experiments

Starting from the scenarios above we have performed different tests to show how the framework could be used in practice. We have setup a testing environment as shown in Figure 6.

The testing environment is composed of the monitoring server, a service requestor and a service provider. As shown in Figure 6, each one of components has a proxy deployed as part of its infrastructure to collect the monitoring data. The service provider is a web application serving simple web services such as arithmetic operations. The service requestor is a web service-testing tool called SoapUI[3]. SoapUI allows generating SOAP requests to make use of the services exposed by the service provider. The monitoring server uses also a proxy to capture all the requests and responses of the monitoring server. This allows the activities of the server to be monitored also.

[3] http://www.soapui.org/

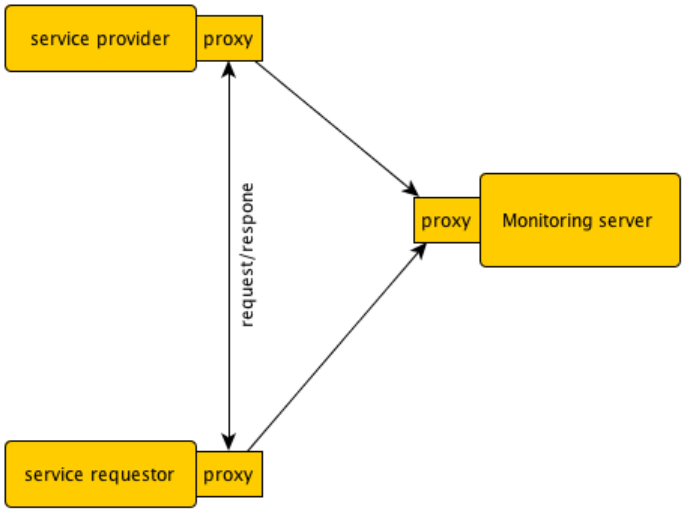

Fig. 6. Testing environment

In our testing we have primarily focused on the two monitoring strategies, the push and the pull. However, other types of tests were explored. In the following the tests and their results are presented in Table I

Test	Description	Result
Push	The service requestor registers a new event "response time > 5 ms". This means that if the service response time is greater than 5 ms, the service requestor wants to be notified. The monitoring server forwards this information to the service provider proxy to add it to the list of properties to monitor. For testing purposes we have implemented a simple service which takes an integer value as parameter and use it to delay the service response time, for instance if we pass as parameter number 5 the service will have 5 ms as response time. Using SoapUI we have generated 10 SOAP calls with different response times.	The service requestor has been notified every time a request violates the condition specified. The notification was sent by e-mail.
Pull S1	The service requestor can request at anytime the monitoring data collected so far by the proxies. Using SoapUI we generate 10 random requests to the service. On the monitoring server side the database is still empty. The service	The monitoring server checks its local database, but since no data is available, it requests the data from the proxy. The proxy returns 10 events, which are forwarded to the service

	requestor calls the getMonitoringData service of the monitoring server.	requestor and also stored in the database.
Pull S1	We performed the same scenario as S1 but this time we have some existing data in the monitoring server database.	The monitoring server sends back the existing data in the database. Additionally, it sends also the newly collected data by the proxy.
Availability	Once the configuration is set. We shutdown the service provider service.	The provider proxy could not receive any response from the service, so it notifies the monitoring server, which notifies the service requestor.
Overhead	Compare the response time of services with and without proxies. 1000 requests have been sent by the service requestor with and without the proxies	Without the proxies the average response time was 0.12 ms while the ones with the proxies was 0.16 ms.

7 Conclusions

In this paper, we have presented a comprehensive framework for simplifying services monitoring. Our framework adopts the model proposed in the NEXOF-RA Monitoring SOA Enterprise Pattern, which defines the architecture of the monitoring components. The framework has been implemented by integrating different open source components and techniques to increase the level of functionality to the final user. The main advantage of the framework is that it has been implemented as a set of services, which gives it the ability to monitor itself. We have tried to be less invasive as possible to avoid adding extra costs for existing infrastructures. However, this limits the number of events we can monitor. For the future we are working to add more interceptors to the proxies to collect more information. The current implementation is a prototype that we have used to test the different components. The next step we are considering is to perform a larger case study in which real services infrastructures will be used to study the effect of the monitoring on the relations between service requestors and service providers.

References

1. Zheng Z., Lyu M. R. (2008) A qos-aware middleware for fault tolerant web services. In *ISSRE*, pages 97–106, Seattle, USA
2. Monitoring in Enterprise SOA pattern http://www.nexof-ra.eu/sites/default/files/Monitoring%20in%20ESOA%20Pattern_v0_7.pdf
3. Baresi L. Nitto E. D. (2007) Test and Analysis of Web Services. Springer-Verlag New York, Inc., Secaucus, NJ, USA
4. Mahbub K., Spanoudakis G. (2004) A Framework for Requirements Monitoring of Service Based Systems. In Int. Conf. on Service-Oriented Computing (ICSOC)

5. Mahbub K., Spanoudakis G. (2005) Run-Time Monitoring of Requirements for Systems Composed of Web-Services: Initial Implementation and Evaluation Experience. In Int. Conf. on Web Services (ICWS)
6. Web Services Description Language (WSDL 1.1). W3C Note, http://www.w3.org/TR/WSDL/, 15 March 2001.
7. Simple Object Access Protocol (SOAP 1.2), W3C Recommendation, http://www.w3.org/TR/soap12, 27 April 2008.
8. Di Penta M., Bastida L., Sillitti A., Baresi L., Maiden N., Melideo M., Tilly M., Spanoudakis G., Gorroñogoitia Cruz J., Hutchinson J., Ripa G. (2009) SeCSE - Service-centric System Engineering: An Overview. In At Your Service (Eds.: Di Nitto E., Sassen A.M., Traverso P., Zwegers A.), MIT Press
9. Damiani E., El Ioini N., Sillitti A., Succi G. (2009) WS-Certificate. IEEE International Workshop on Web Services Security Management (WSSM 2009), Los Angeles, CA, USA
10. Truong H. L., Fahringer T. (2004) SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid. *Scientific Programming*, vol. 12, no. 4, pp. 225–237, axGrids 2004 Special Issue
11. IBM Tivoli Composite Application Manager for SOA, 2006.
12. Predonzani P., Sillitti A. (2001) Components and data-flow applied to the integration of web services. Electronics Society
13. Petrinja E., Nambakam R., Sillitti A. (2009) Introducing the OpenSource Maturity Model. In Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS '09). IEEE Computer Society, Washington, DC, USA
14. El Ioini N., Sillitti A. (2011) Open Web Services Testing. IEEE World Congress on Services (SERVICES), DC, USA
15. Granatella G., Predonzani P., Sillitti A., Succi G. (2004) Selecting components in large COTS repositories, Journal of Systems and, Elsevier