SALMon: A SOA System for Monitoring Service Level Agreements

Marc Oriol, Xavier Franch, Jordi Marco

Universitat Politècnica de Catalunya, Spain
moriol@lsi.upc.edu
franch@essi.upc.edu
jmarco@lsi.upc.edu

Abstract. In this paper we present SALMon, a tool assessing the satisfaction of service level agreement (SLA) clauses by service-oriented systems. SALMon itself is organized as a service-oriented system that offers two kind of services: 1) the Monitor service that measures the values in execution time of dynamic quality attributes (like response time or availability), and 2) the Analyzer service that detects and reports violations of SLA clauses from the values obtained with the Monitor. The SALMon tool is highly versatile, allowing: 1) both active testing and passive monitoring as strategies, 2) different types of technologies for the monitored/tested systems (e.g., Web Services, RESTful services), 3) agile definition of measure instruments for new quality attributes. The service-oriented nature of SALMon makes it scalable and easy to integrate with other services that need its functionalities.

1. Introduction

Service Oriented Architecture (SOA) is one of the most popular architectural styles used nowadays. The systems built under this paradigm, named Service Oriented Systems (SOS), make use of several services in order to provide a fully-functional software system. The technology used to deploy these services may vary in a single SOS (e.g., Web Services, RESTful services, …), and dealing with this heterogeneity is a challenging matter.

Services are usually provided by third-parties, and therefore, both service clients and service providers must agree with Service Level Agreements (SLAs) specifying the Quality of Service (QoS) that must be achieved by the given services.

Knowing the QoS of the different services is mandatory in order to detect if an SLA is being (or going to be) violated. To retrieve QoS of SOS there are basically two approaches: passive monitoring and active monitoring (also known as testing).

- Passive monitoring: this approach obtains the QoS information by collecting data of the interaction between the service provider and the service client.
- Active monitoring (testing): in this approach an engine invokes the service in a systematic manner to ultimately obtain its QoS.

Both approaches have their own strengths and weaknesses and they should be used accordingly to the needs of the user and the capabilities of the approach.

By using testing, the system is able to detect a malfunction of the service before the client and it is able to have control on the structure and frequency of the invocations. However testing may have a significant impact on the performance of the service and some operations could or should not be tested (e.g., operations that change the state of the system in a manner that cannot be rollbacked). On the other hand passive monitoring has less intrusive effects on the system, but it loses the control on the invocations, and therefore on the QoS data that is obtained (e.g.: the QoS of a service that is rarely used but it is critical to fulfill its SLA when it is invoked).

In this paper, we present SALMon, a monitoring and SLA violation detection tool. SALMon combines monitoring and testing techniques to obtain the runtime QoS information of dynamic quality attributes (i.e., attributes whose value may change during execution, such as response time and availability). On the basis of this information, SALMon may inform about the violation of SLA clauses. SALMon may be considered itself a SOS that offers two kind of services, QoS monitoring and SLA violation detection. Although our current implementation is focused on Web Services, the architecture and interface of the tool have been developed in a technological-independent manner. Therefore, SALMon is able to retrieve (with both monitoring and testing approaches) the QoS of heterogeneous SOS.

The rest of the paper is structured as follows. Section 2 describes the related work in this field. Section 3 provides a framework for the definition of metrics to measure QoS. Section 4 is dedicated to the details of SALMon architecture. Section 5 describes how SALMon can be used. Section 6 provides some examples in which SALMon has been used. Finally, Section 7 provides the conclusions.

2. Related Work

There exists numerous contributions in the literature that focus on obtaining real-time QoS in Service Oriented Systems, either by monitoring or testing approaches [1][2][3][4][5][6]. However, most of the literature considers monitoring and testing methods as two disjoint approaches, and up to our knowledge, there is no framework that combines both of them to retrieve the QoS in a unique system.

Most of the monitoring or testing contributions focus on a concrete kind of services such as Web Services [6][1][2], or relies that the monitored/tested services uses a particular technology such as BPEL engines [4] or Java-Axis engines [1]. Nevertheless, some recent contributions studies the problem in SOS to support service monitoring [5] or service testing [3] that are not limited to Web Services.

Regarding technological dependent contributions, Zhou et al. [1] proposed an approach to install monitors for Web Services in the execution chain of Axis2 in an automatic manner.

In [4] Moser et al. presented VieDAME, an aspect-oriented monitoring framework to monitor BPEL processes and ultimately perform service adaptation.

More flexible monitoring architectures are presented in [2] by Benharref et al. which considers a multi-observer architecture where monitors could be placed in both client and server sides to retrieve the QoS of Web Services.

Comuzzy et al. described in [5] an event-based monitoring approach of services to assess SLAs in the business, software and infrastructure layers.

In [6] Bertolino et al. presented a framework to assess SLAs for Web Services under testing approach that made use of WSDLs and SLAs for generating the tests.

In [3] Hielscher et al. introduced a framework, named PROSA, that exploits online testing techniques in order to provide proactive adaptations for services.

3. Quality Attributes and Metrics

Based on our previous works [7], we have identified a set of dynamic quality attributes for services from a quality model that extends the ISO/IEC 9126-1 standard [8]. Dynamic quality attributes are those whose values may change on run-time and therefore a monitoring mechanism is needed to assess them in a confident way. For these attributes, some quality metrics have been defined. Table 1 presents the set of dynamic quality attributes and quality metrics which SALMon is currently focused on. Attributes and metrics can be classified according to two criteria:

- Quality attributes may refer to the whole service (e.g., Availability and Server's Response Time) or to a particular service element. For example, in the case of Web Services, we focus on the operations of the service (e.g., Response Time, Execution Time and Accuracy).
- Quality metrics may be basic or derived. Basic metrics are those which must be monitored to obtain their values (e.g., Current Response Time and Current Availability), whereas derived metrics are those which can be calculated from a set of basic metrics (e.g., Average Response Time, defined as the Arithmetic mean of Response Times in a given interval; or Mean Time to Recovery, defined as the average time that the system takes to recover from any failure).

		Metric	Description
Service's attributes	Availability	Current availability	The current availability of a Service (may be available or unavailable)
		Accumulative availability time	The percentage of the time a Service has been available in a time interval
		Accumulative unavailability time	The percentage of time a Service has been unavailable in a time interval
		Average recovery time	The average time that takes for a Service to recover from an unavailability
	Server Resp. Time	Current Server Response Time	The time needed to have access to a Service. That is, Network time + Queue delay
		Miminum Server Response Time	The lowest server response time in a time interval
		Maximum Server Response Time	The maximum server response time in a time interval
		Average Server Response Time	The average server response time in a time interval
Operation's attributes	Response time	Current response time	The time between sending a service request and receiving a response
		Mininum response time	The lowest response time in milliseconds in a time interval
		Maximum response time	The maximum response time in milliseconds in a time interval
		Average response time	The average response time in milliseconds in a time interval
	Execution time	Current execution time	The time between receving a service request and creating a response
		Minimum execution time	The lowest execution time in milliseconds in a time interval
		Maximum execution time	The maximum execution time in milliseconds in a time interval
		Average execution time	The average execution time in milliseconds in a time interval
	Accuracy	Current functionality compliance	The current functionality compliance of a service.
		Parameter accuracy factor	The ratio of correct type parameters passed to a Service method in a time interval
		Result accuracy factor	The ratio of correct type results returned by a Service method in a time interval
		Fault factor	The ratio of faults that a Service method has generated in a time interval

Table 1: Metrics defined over the dynamic quality attributes measured in SALMon.

4. SALMon Architecture Overview

SALMon is a SOS by itself. It consists of two main Web Services: the Analyzer and the Monitor services. The first one is used to check the set of conditions stated in the SLA, whereas the latter is used to retrieve the QoS of the different monitored services.

Following the principles of SOA, SALMon can be easily integrated with other services in order to provide higher level QoS-based frameworks, such as self-healing SOA systems [9] or service selection systems, among others. See Section 6 for a couple of examples.

Figure 1 shows the architecture of SALMon, that is introduced in the rest of the section. We will skip the parts referred to the data base and to the authentication and authorization service.

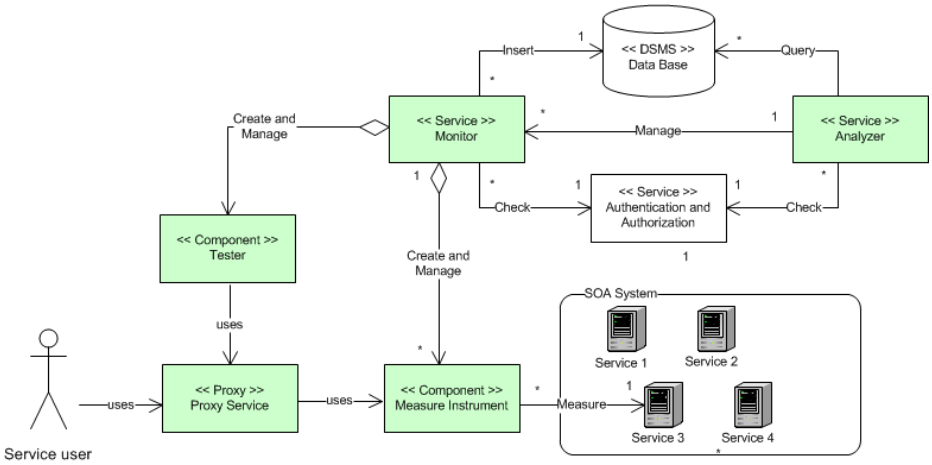


Figure 1: SALMon Architecture

4.1. The Analyzer Service

The Analyzer service is responsible for checking if the set of Service Level Objectives (SLOs) stated in the different SLAs are fulfilled. Notice that the Analyzer is able to manage several SOS. Each SOS might be composed of several services, eventually supplied from different providers, and therefore with different SLAs, and for each SLA there are several SLOs to be checked. The goal of Analyzer is to assess these SLOs on run time and report violations if these objectives are not fulfilled.

WS-Agreement specification, the current standard to specify SLAs, does not define the syntax or semantics to express SLOs, and therefore, different WS-Agreement compliant documents might express SLOs in completely different ways. For such a reason, and in order to avoid mismatches, the set of conditions to be checked in SALMon are stated through the usage of the Analyzer WSDL interface, particularly, with the method SetCondition().

We consider an SLO as a condition composed of the evaluated metric, a relational operator and a value for the comparison (i.e. "Current Response Time < 100ms").

To check on run-time if these conditions are fulfilled, Analyzer makes use and manages several monitors. Each monitor is responsible for a particular SOA System and they retrieve the QoS needed to check the conditions.

The conceptual model in Figure 2 summarizes all the concepts and relationships stated above.

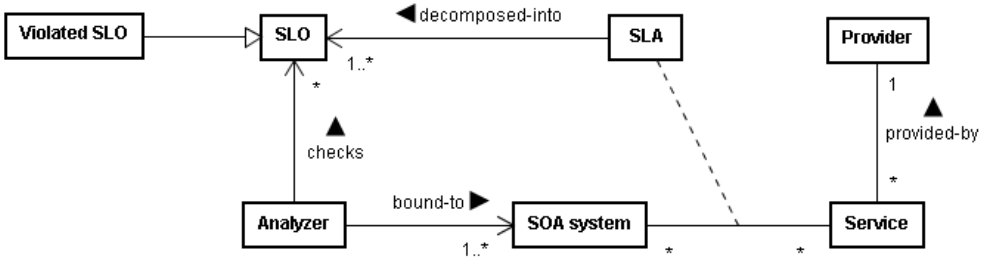


Figure 2: Concepts and relationships implied in the Analyzer service

Since Analyzer does not interact with the monitored technologies, it is a technologically-independent tool enabled to define conditions for any kind of service (e.g., Web Services, RESTful services).

4.2. The Monitor Service

The Monitor service is responsible to retrieve the values of the quality metrics of the different services in a single SOS.

A monitor can be used by the Analyzer or independently by the user (either human or an external component). In the latter case, the user is able to monitor the QoS of a SOS without being interested on checking the fulfillment of the SLOs.

4.2.1 Retrieving Quality Metrics

The monitor manages several measure instruments (see Figure 1). A Measure Instrument is a component that implements the logic needed in order to obtain the value of a concrete basic quality metric (e.g., Current Response Time, Current Availability, Accuracy…) of a service or operation.

Derived quality metrics are calculated from the set of basic quality metrics retrieved from the measure instruments using an aggregator function in a defined time interval (maximums, minimums, averages…).

Since measure instruments are the core components that actually retrieve the values of the basic metrics, these components are technologically dependent on the kind of service they are monitoring. In this sense, the Monitor service stays without the technological details and just creates the different measure instruments to obtain the QoS.

These decisions provide high extensibility to the monitor. Adding new kind of services or basic quality metrics can be achieved simply by implementing the suitable measure instrument.

Measure instruments need to have knowledge regarding the usage of the service to compute the metric. This is reduced to have access to the messages sent between client and provider on real time. To do so, we make use of the Man-in-the-middle approach for monitoring [10]. This approach is based on putting an agent between the client and the service's communication, enabling the agent to notify to the measure instruments about the messages exchanged between both parties.

SALMon implements the Man-in-the-middle approach by the usage of a proxy. The location of the proxy is important in order to obtain an accurate value of the metrics that are related to time. If the proxy is located at the client side, SALMon is able to obtain the response time (configuration 2 in Figure 3), whereas if the proxy is located at the server side, SALMon is able to obtain the execution time (configuration 3 in Figure 3). Configuration 4 obtains a time which value is between both metrics, and configuration 1 supposes that both service clients and services are installed in the same system.

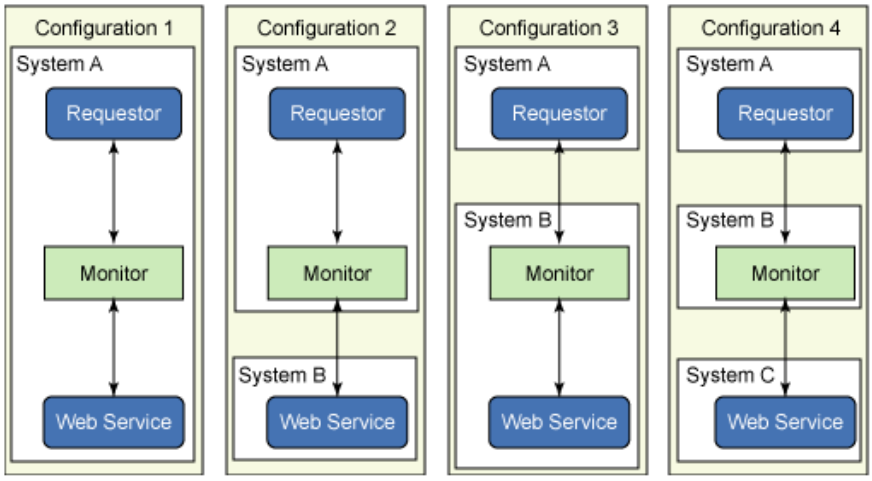


Figure 3: Configurations of monitor allocations as presented in [9]

4.2.2 Passive Monitoring and Testing Approaches

Both passive monitoring and testing approaches have their own advantages and disadvantages. Providing a framework with both capabilities can take advantage of both techniques to assess the QoS. To do so, the monitor has a component, named Tester, responsible of invoking the services (see Figure 1 and Figure 4). Once a user sets a condition to be checked, she also specifies if the condition should be checked through passive monitoring or testing.

Since the tester interacts with the monitored services, it is also a technological dependant component. Currently we have testers for Web Services (WSTester) and for operations of Web Services (OpTester). The first ones just ask for the Web

Service (particularly for its WSDL) in order to test metrics that are related to the whole Web Service (Availability and Server Response Time). The second one is used in order to invoke operations of the Web Service to obtain metrics related to the operations.

The information needed to perform the test depends on the kind of service or service element we are interested on. For WS testers, only the URL and the time interval between measures is needed. For WS-Operation testers, we also need a SOAP request (which may include the user and the password in the header if authentication is required) and a pattern to identify a valid SOAP response.

In our current Tester component, we are considering Stateless Web Services, so we do not undertake the issue of workflow-based mechanism to invoke them.

These invocations are performed through the usage of the proxy, providing therefore both passive monitoring and testing mechanisms.

Figure 4 presents the class diagram that includes the concepts presented above.

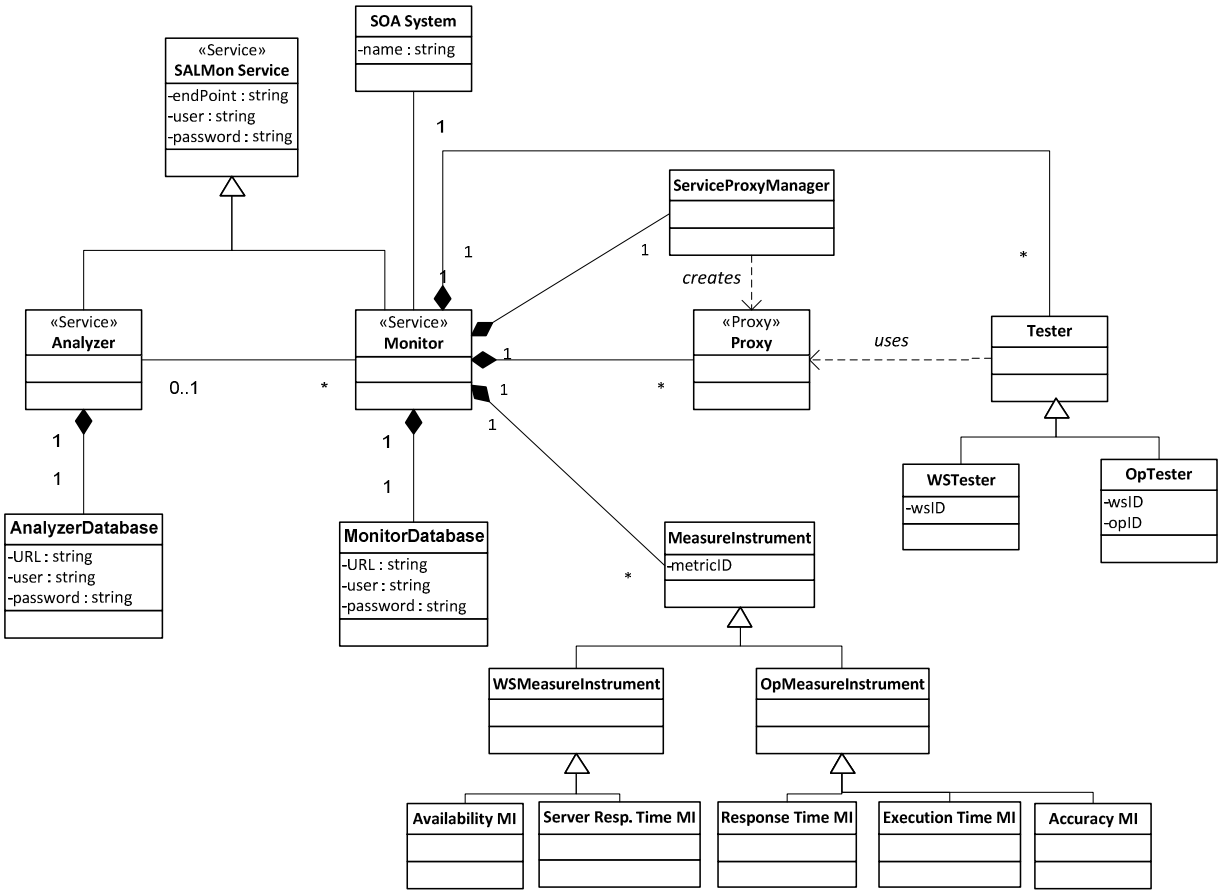
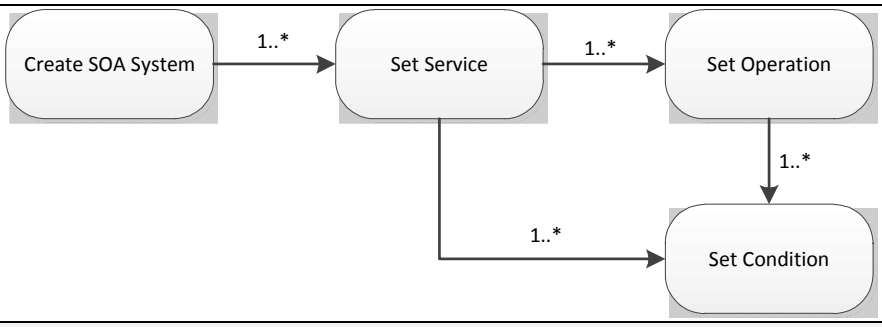


Figure 4: Data Model of SALMon

5. Using SALMon

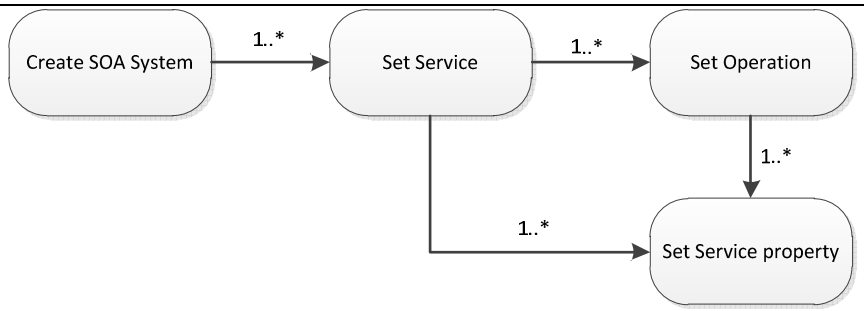
Both Analyzer and Monitor services offer a set of methods defined in their respective WSDLs, the services should be used as indicated in the workflows presented in Figure 5 and Figure 6.



Analyzer WSDL

CreateSOASystem(String name): IdentifiedResult
SetService(int soaID, ServiceInformation service): IdentifiedResult
SetOperation(int soaID, int serviceID, OperationInformation service): IdentifiedResult
SetCondition(int soaID, int wsID, Condition c, AssessmentApproach ap): IdentifiedResult
SetCondition(int soaID, int wsID, int opID, Condition c, AssessmentApproach ap): IdentifiedResult

Figure 5: Analyzer Workflow and WSDL



Monitor WSDL

CreateSOASystem(String name): IdentifiedResult
SetService(int soaID, ServiceInformation service): IdentifiedResult
SetOperation(int soaID, int serviceID, OperationInformation service): IdentifiedResult
SetServiceProperty(int soaID, int wsID, Metric m, AssessmentApproach ap): IdentifiedResult
SetServiceProperty(int soaID, int wsID, int opID, Metric m, AssesmentApproach ap): IdentifiedResult

Figure 6: Monitor Workflow and WSDL

A user may either use Analyzer or Monitor services accordingly to his or her interests. In this section, we explain briefly just the Analyzer service workflow since the Monitor service workflow is quite similar.

The user first registers the SOA System into Analyzer service. Analyzer then assigns this SOS to a concrete available Monitor and returns an identifier for that SOS.

For each SOS, the user may set some services using the setService method. This method has a parameter named ServiceInformation with information of the service the user is interested to monitor (i.e.: name, description, URL,…). In case of Web Services, there is a subclass WebServiceInformation with specific Web Service concepts (such as WSDL).

For the operations of the Service the user should call setOperation, which in the same way, has a parameter named OperationInformation.

Finally the user can set the conditions. As stated above, a condition can be given for an operation of the service or to the whole service (depending on the kind of metric). Also, a parameter indicates if we want to perform monitoring or the testing approach.

6. SALMon as a SOS: Two Scenarios of Use

Since SALMon is a SOS, both Analyzer and Monitor services have been used in other systems as external services to support QoS-based frameworks. Particularly SALMon has been used in a framework for ranking services named WeSSQoS and an ongoing framework to support self-healing SOS named MAESoS.

6.1 WeSSQoS

WeSSQoS [11] is a framework which ranks Web Services accordingly to the user's non-functional requirements and the QoS of the different Services that belongs to the same domain.

WeSSQoS has been implemented under the SOA principles and therefore is able to obtain the data and QoS of services from different sources, named repositories. These repositories must have a WSDL interface with a common operation to extract the QoS data. To this effect, WeSSQoS is able to combine the given information through the QoS Selector Service, and once the data is collected, it invokes the QoS Selection Model, which is the service that performs the actual ranking.

In this framework, SALMon is used as a repository to obtain real-time dynamic quality attributes of the services. To do so, WeSSQoS makes use of the Monitor service under the testing approach (see Figure 7).

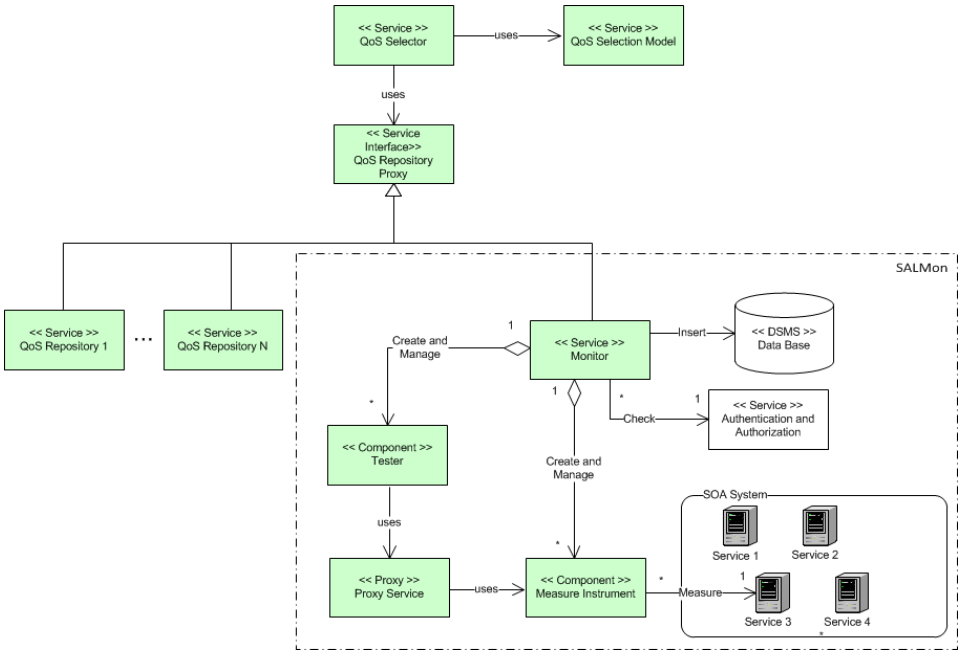


Figure 7: WessQoS Architecture integrated with SALMon

6.2. MAESoS

MAESoS [12] is an ongoing framework which integrates into a unique model the components of the following modeling languages:

- ***i**:** to determine the goals of the user and their decomposition to actual services
- **Variability Models*:*** to express the variability points of services to support the adaption.
- **Quality Models:** used to define a hierarchical set of quality Attributes and Metrics grouped in categories.

MAESoS is used to support self-healing SOS. It's also implemented as an SOA and among other components it has a Decision Maker service which is responsible for dynamic rebinding for the recovery of the system in case of a service malfunction. Decision Maker makes use of the Analyzer Service to check for SLA violations, when a violation is detected it is notified to the Decision Maker of the affected SOS. The Analyzer can be used in both passive and active approaches in order to achieve reactive and proactive adaptations respectively.

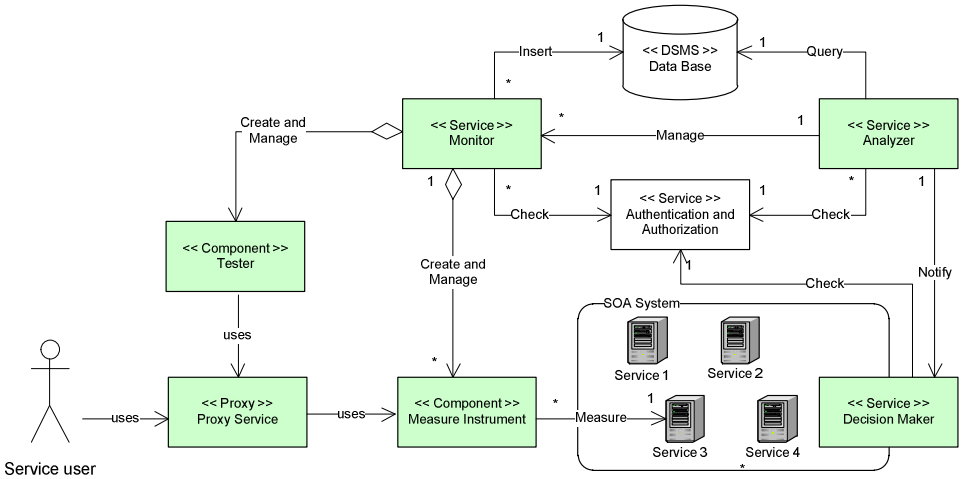


Figure 8: Partial MAESoS Architecture integrated with SALMon

7. Conclusions

In the context of SOS, retrieving the QoS of the different services is mandatory in order to detect the accomplishment of the different Service Level Objectives stated in SLAs. We have presented SALMon, a SOS itself which is able to retrieve QoS to ultimately check the conditions stated in these SLAs. The main advantages of SALMon are:

- It supports both active and passive monitoring. Providing both assessment techniques in a unique framework offers to the final user the benefits of each approach (which might be selected accordingly to his/her needs).
- It is designed to support any kind of service technology. To do so, SALMon provides a technological independent interface and has an infrastructure where the only components that are technologically dependent are those which directly interact with the monitored services (i.e.: measures instruments and tester engines) whereas the whole framework remains in an upper technologically independent layer. Nevertheless, our first implementation has been realized for web services but we plan to implement monitoring of multiple types of services in the same monitor with different kinds of Measure Instruments and tester engines.
- It is open for integration into other systems. The service-oriented nature of SALMon makes it scalable and easy to integrate with other systems, as we have shown in two currently running systems. Integration with other two systems is currently under scheduling.
- It is open for extension with respect to measures. Since Measure Instruments are responsible to obtain a concrete basic quality metric, extending the tool

with new kinds of basic quality metrics is achieved by implementing just the suitable Measure Instrument.

Acknowledgements

This work has been supported by the research project ADICT, TIN2007-64753, MCyT, Spain. Marc Oriol has a FPI grant bound to the project TIN2007-64753. We also would like to thank the work done by, and discussions made with, other members of the group, remarkably David Ameller, Lidia López and Marc Rodríguez.

References

[1] C. Zhou, L. T. Chia and B. S. Lee, "QoS measurement issues with DAML-QoS ontology" in Proc. IEEE International Conference on e-Business Engineering (ICEBE'05), 2005.

[2] A. Benharref, R. Dssouli, M. A. Serhani and R. Glitho, "Efficient traces collection mechanisms for passive testing of Web Services", Information and Software Technology, vol. 51, pp. 362-374, 2009.

[3] J. Hielscher, R. Kazhamiakin, A. Metzger and M. Pistore, "A framework for proactive self-adaptation of service-based applications based on online testing" in 1st International Conference of the Future of the Internet of Services (ServiceWave 2008), 2008.

[4] O. Moser, F. Rosenberg, S. Dustdar, "Non-intrusive monitoring and service adaptation for WS-BPEL", Proceeding of the 17th international conference on World Wide Web (WWW 2008),2008.

[5] Comuzzi M, Kotsokalis C., Spanoudakis G., Yahyapour R.: Establishing and Monitoring SLAs in complex Service Based Systems, IEEE 7th International Conference on Web Services (ICWS '09), 2009.

[6] A. Bertolino, P. Inverardi, P. Pelliccione and M. Tivoli, "Automatic synthesis of behavior protocols for composable web-services," in ESEC/FSE 2009, 2009.

[7] M. Oriol, X. Franch, J. Marco, D. Ameller. "Monitoring Adaptable SOA-Systems using SALMon". Mona+ Workshop in ServiceWave conference, 2008

[8] ISO, International Organization for Standarization. ISO/IEC Standard 9126: Software Engineering – Product Quality, part 1. 2001.

[9] C. Dabrowski, K. Mills."Understanding self-healing in service-discovery systems". Proceedings of the first workshop on Self-healing systems, 2002.

[10] P. Brittenham "Understanding the WS-I Test Tools". Downloaded from www-128.ibm.com/developerworks/webservices/library/ws-wsitest/. 2003.

[11] O. Cabrera, M. Oriol, J. Marco, X. Franch, O. Fragoso, R. Santaolaya. "WeSSQoS: Un Sistema SOA para la Selección de Servicios Web según su Calidad", JSWEB'09, 2009.

[12] B. Burgstaller et al. "Monitoring and Adaptation of Service-oriented Systems with Goal and Variability Models". Research Report LSI-09-8-R, Universitat Politècnica de Catalunya, 2008.