Monitoring Adaptable SOA-Systems using SALMon

Marc Oriol, Jordi Marco, Xavier Franch, David Ameller

Universitat Politècnica de Catalunya, Spain
{moriol,jmarco,franch,dameller}@lsi.upc.edu

Abstract. Adaptability is a key feature of Service-Oriented Architecture (SOA) Systems. These systems must evolve themselves in order to ensure their initial requirements as well as to satisfy arising new ones. In SOA Systems there are a lot of dependencies between services, but each service is an independent element of the system. In this situation it is necessary not only ensuring that the system fulfils its requirements but also that every service satisfies its own requirements, and dynamically adapting the system when some of them cannot be ensured. In this paper we propose a SOA system, named Service Level Agreement Monitor (SALMon), for monitoring and adapting SOA Systems at run time. SALMon is based on monitoring the services for detecting Service Level Agreement (SLA) violations. The SALMon architecture is composed of three types of components: Monitors, which are composed of measure instruments themselves; the Analyzer, which checks the SLA rules; and the Decision Maker that performs corrective actions to satisfy SLA rules again. These three types of components are mostly technology-independent and they act as services inside of a SOA system making our architecture very scalable and comfortable for its purpose.

1. Introduction

Service-Oriented Architecture (SOA) has become one of the most successful architectural styles used for the development of software systems. The main characteristic of this architecture is the construction of software solutions based on a group of services that communicate with each other. However, this high coupling also implies a strong dependency among the different components of the SOA system. A failure of a service could imply the malfunction or failure of the whole system.

The emerging research challenge, then, is how we can ensure that the components which are using these services are able to offer the same benefits and accomplishing the user's requirements in case that one of these services fails.

In this context, being able to build self-adaptive SOA systems is a major undertaking. Self-adaptive SOA systems are those which are able to change dynamically the services they use in order to keep fulfilling the Quality of Service (QoS) requirements stated in Service Level Agreements (SLA). Self-adaptive SOA systems demand having several alternative services to use in case that a service is not working properly.

The construction of this kind of SOA systems requires tool support for (1) monitoring services to continuously know their QoS, (2) determine when the SLA is

being violated, and (3) finally take the decision of using an alternative service with the same or similar functionality.

In this paper, we present SALMon, a SOA system itself that uses a monitoring technique to provide runtime QoS information that is needed to detect and eventually correct SLA violations. SALMon is still under development, therefore the prototype we present here should be considered as ongoing research. Notice that although the architecture and interfaces of SALMon are technologically independent so that the tool could be used to monitor any kind of services of a SOA system, we will focus in this first preliminary version on monitoring web services.

The rest of the paper is structured as follows: first we provide a framework for metrics definition based on previous works and the second part is dedicated to the details of SALMon architecture. Finally there is a section for the conclusions.

2. Quality Attributes and Metrics

In our previous work [1], we identified the quality attributes of services building a quality model [2]. Our approach is compliant with the ISO/IEC 9126-1 standard [4] and remarkably we have added some subcharacteristics related to non-technical issues following the advices given in [5]. This model was developed during our participation in a ITEA European project, SODA (Services Oriented Devices & Delivery Architectures, www.soda-itea.org), in which we had the responsibility of identifying and classifying the characteristics needed for defining the quality of Web services.

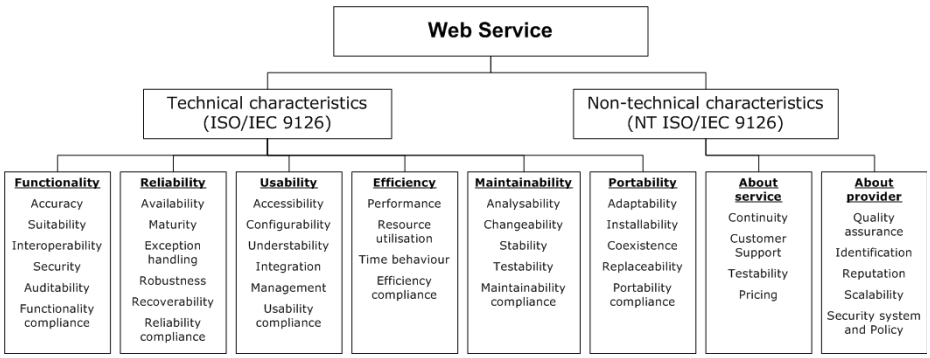

Figure 1: Web Service Quality Characteristics.

In the proposed quality model (Figure 1) we have identified several characteristics. Notice that, for instance, one characteristic is Efficiency and one of its subcharacteristics is the Time Behavior. But Time Behavior itself is not a single measurable concept, therefore we need to define attributes to decompose this subcharacteristic. The attributes are normally dependent on what we want to measure. In our case, since we are focusing on Web services, Response Time and Execution Time are good examples of measurable attributes for Time Behavior.

Time behaviour

Time behaviour is the capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions. In SALMon, we are interested in two particular measurable attributes:

- Response Time: It measures the time that a Web Service takes to give a basic response.
- Execution Time: It measures the time that a Web Service takes to execute a certain job (a method, a process...).

Availability

The availability is the degree to which the system is operable and in a committable state. The user might want to ensure that a service is available.

Accuracy

Accuracy is the capability of the software product to provide the right or agreed results or effects with the needed degree of precision. In this case, the user could want to monitor a concrete functionality of the web service, in this way, we talk about functionality test.

The attributes Response Time and Availability are attributes that belong to the web service since they are related to the whole service. If a web service is not available, none of all its operations will be available. If the response time of the web service is increased (e.g., because of a high demand on the service), the execution time of all the operations of the web service will be affected accordingly. On the contrary, Execution Time and Accuracy are attributes that belong to concrete operations of the web service.

Once these four attributes have been identified, the next step is determining their concrete metrics that will be subject of measure. In Table 1 we present these metrics.

		Metric	Description
Web Service's attributes	Availability	Current availability	It measures the current availability of a Web Service all the time or in slots of time.
		Accumulative availability time	It measures in percentage how much time a Web Service has been available since it has been monitored for a first time.
		Accumulative unavailability time	It measures in percentage how much time a Web Service has not been available since it has been monitored for a first time.
		Average recovery time	It measures in seconds the average time that a Web Service needs to be available again (It considers unavailability).
	Response time	Current response time	It measures the current response time in milliseconds to access to a Web Service.
		Minimum response time	It measures which is the lowest response time in milliseconds to access to a Web Service.
		Maximum response time	It measures which is the maximum response time in milliseconds to access to a Web Service.
		Average response time	It measures which is the average response time in milliseconds to access to a Web Service.
Operation's attributes	Execution Time	Current execution time	It measures the current execution time in milliseconds that a Web Service takes to execute a job. (response+execution)
		Minimum execution time	It measures which is the minimum execution time in milliseconds that a Web Service takes to execute a job.
		Maximum execution time	It measures which is the maximum execution time in milliseconds that a Web Service takes to execute a job.
		Average execution time	It measures which is the average execution time in milliseconds that a Web Service takes to execute a job.
	Test functionality	Current functionality compliance	It measures the current functionality compliance of a Web Service.
		Parameter Accuracy factor	It divides the number of accepted correct type parameters passed to a Web Service method by the number of expected parameters for this method (1 is better).
		Result Accuracy factor	It divides the number of correct type results returned by a Web Service method by the number of expected results of this method (1 is better).
		Fault factor	It divides the number of faults that a Web Service method had generated by the total times that this method had been executed (1 is worse)

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

An important distinction is between basic metrics and derived metrics. Basic metrics are those which must be monitored to obtain their values. Examples of basic metrics are Current Response Time or Current Availability. Derived metrics are those which can be obtained from a set of basic metrics. For example, the Average Response Time is a derived metric since it can be obtained through the set of Current Response Times in an interval of time. Another example is Recovery Time Failure which is also a derived metric from the basic metric Current Availability. This distinction is important since given the values of a basic metric, there is no interaction with the monitored service to obtain the values of the corresponding derived metrics.

3. Platform Architecture Overview

The architecture of SALMon is a Service Oriented Architecture. Our platform is composed by the following services (see Fig. 2): Analyzer, Decision Maker and Monitor. Another type of module relevant to the SALMon architecture are the Measure Instruments, which are part of the Monitor service. SALMon also uses a service to store the monitoring data and a service for authentication and authorization.

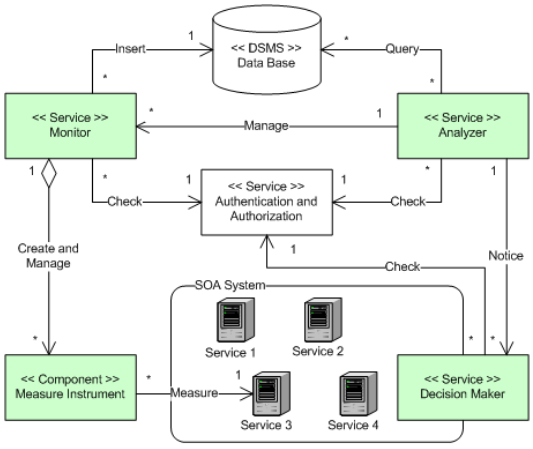

Figure 2: Platform Architecture

In the rest of the section we will present this services and components in detail:

- Monitor service. It uses Measure Instruments to get the information about QoS. Measure Instruments are components generated by the Monitor to communicate with the services in order to get the monitoring information that is relevant for computing the chosen metrics. This information is stored in a data base and is also rendered to the Analyzer.

- Analyzer service. It is responsible of checking for SLA violations in concrete SOA Systems: when a violation is detected, it is notified to the Decision Maker service of the affected SOA System. To attain its goal, the Analyzer manages a Monitor service.

- Decision Maker service. It selects the best treatment to solve the incidences detected by the Analyzer in a concrete SOA system. Each Decision maker is related with one (and only one) SOA System.

3.1. Monitor

The Monitor service is composed of several Measure Instruments for the same SOA System. Measure Instruments are components used to get all the *basic metrics* of the selected quality attributes. Derived metrics will be obtained from them. These components are responsible of bringing the measures to the Monitor, which has the responsibility of maintaining this information updated. The update process is an iterative call to each Measure Instrument in different intervals of time, saving the results in a database.

Since our current approach on monitoring services is intrusive[1], Measure Instruments have the responsibility to minimize the number of interactions performed with the monitored service.

The Monitor needs the information of the service to monitor service's metrics (Response Time and Availability) and also information of the operations to monitor operation's metrics (Execution Time and Functionality Test).

To monitor the metrics of a Service, service details such as *url* and *port* are needed. The time interval between measures to the service is also needed and some services might require a user and a password.

In order to minimize the interaction with the monitored service, all service-metrics share the same time interval between calls and use the same measure instrument. Therefore, if we want to measure the basic metrics current availability and current response time of a service, the same measure instrument is in charge of measure both metrics in the same call.

To monitor the metrics of an operation, further information details are needed. In particular, the name of the operation and at least one valid SOAP message request. To test the accuracy of the operation, we need also to have the knowledge to determine whether if a response message is valid or not. To do so, we use patterns of correct SOAP message responses. If the message response meets the pattern, we say that the response is a valid message, otherwise we say that it's invalid. This approach however, cannot state if the data given in the response is reliable but if it is well-structured and consistent accordingly to the request.

3.2. Analyser

The Analyzer manages Monitors and checks for SLA violations in concrete SOA systems. When a violation is detected it is notified to the Decision Maker of the affected SOA system. In general an Analyzer can handle multiple SOA systems using one Monitor and one Decision Maker for each one. The use of Decision Maker services is optional but if they are not used, the SALMon user is limited to monitoring and SLA violation detection.

The SLA can be configured manually with the interface provided by the Analyzer or automatically with a SLA standard document for each service (e.g., WSLA [3] for

[1] Intrusive (or Active) monitoring approach is also known as Testing.

the case of Web services). We understand SLA as a set of conditions that must be true in some time interval. A condition is composed of the evaluated metric, a relational operator and a value for the comparison (i.e. “Current Response Time < 100ms” is a condition that must be true for the specified service during the specified time interval).

Defining time intervals is important since some conditions are relevant in a specific interval of time or date. For instance, it could be possible that a service is required to be available in a specific timetable, but we could agree that this service can be temporally unavailable in a scheduled time for maintaining purposes.

The Analyzer is also responsible to compute the desired derived metrics from basic metric values stored by the monitor service.

3.3. Decision Maker

The Decision Maker service has a repository of treatments and alternative services for a concrete SOA system. It will automatically select and execute the best treatment for the reported incidences.

Because the kind of job of this service and for security reasons, it is preferred to place the service in the concrete SOA system where it is working.

The Decision Maker has the following responsibilities:

- To take actions when something goes wrong in the SOA system.
- To write reports of the incidences with the taken actions.

3.4 Users

There are two kinds of user in the SALMon architecture, normal user and administrator user. Note that the kind of user is set for each service, for example one user could be administrator of an Analyzer and a Decision Maker but only with normal access to a Monitor service.

The normal users will be limited to the finality of each service while the administrators have extra functionalities: management of the access to the service, establishment of restrictions in services (e.g.. set the maximum number of Measure Instruments in a Monitor), and set the interconnection between services.

The user of the Analyzer service must be authenticated before start working with it, this user must be authorized to use the Monitor services. If the user wants to use Decision Maker services, he/she must also have enough rights in each of the monitored SOA systems. Measure instruments are property of one monitor so they don’t need authentication.

To be able to use SALMon the user will need to have at least a normal user level in one Analyzer and in one Monitor. The Decision Maker is optional but if the user has no access to the Decision Maker or the SOA system has no Decision Maker service, it will be limited to monitoring.

3.5 Use cases

As we can see in Fig. 3, SALMon is composed of 12 groups of use cases. We may distinguish them whether if they belong to a normal user or to the administrator. In the diagram we have also two virtual actors: Analyzer Engine is responsible of checking SLA while Monitor Engine is responsible of measuring the metrics of the web services. A normal user will interact directly with Analyzer service, which will in turn, communicate with the appropriate monitor.

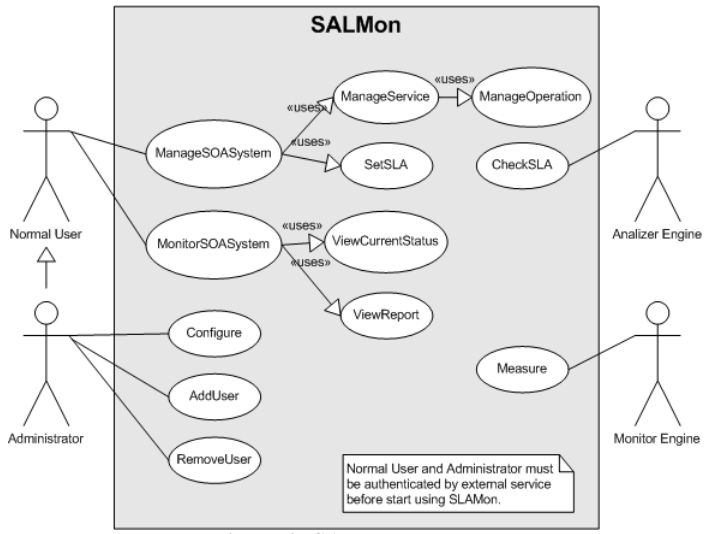

Figure 3: SALMon use cases

3.6 Data Model

In Figure 4 we show the data model of SALMon. To build this data model we have extracted some ideas from the State of the Art report of monitoring web services developed by the NESSI group [6]

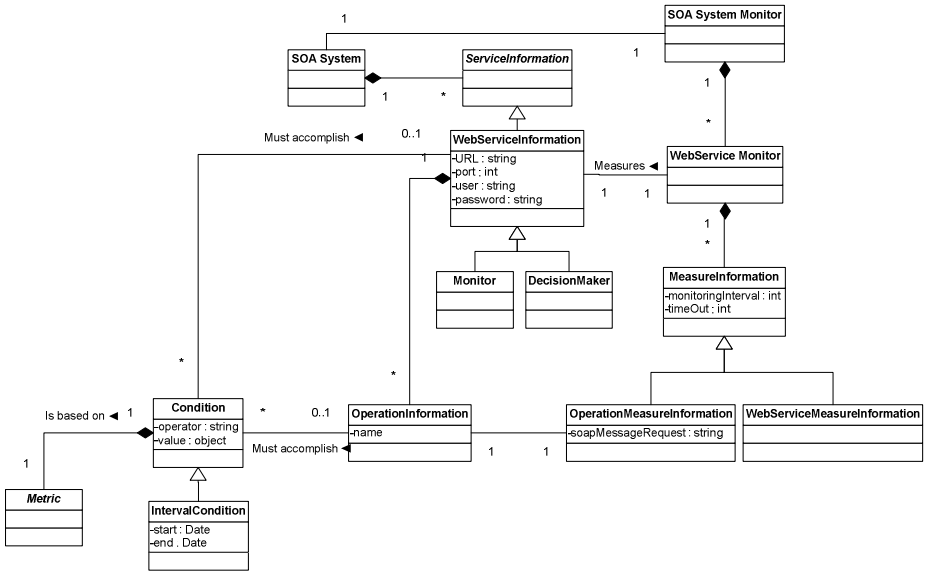

Figure 4: Data Model of SALMon

4. Conclusions

In the context of SOA systems, dynamic changes are needed in order to keep fulfilling the QoS requirements stated in SLAs. We have presented SALMon, a SOA System which is able to monitor services, check their SLA and eventually take decisions in order to support self-adaptation of SOA systems.

The SALMon architecture has been designed to support any kind of services. To do so, SALMon have general interfaces which can be applied to any type of service. However, we are focusing our first implementation in monitoring web services since they are the most common type of service used in a SOA system. As future work we plan to support monitoring of multiple types of services using the same monitor with different kinds of Measure Instruments, so we will be able to monitor an entire heterogeneous SOA system.

We have identified that some metrics are easy to monitor (e.g., response time), but others (e.g., accuracy) need a more complex method to measure them. In this area, we are focusing in how to determine accuracy metrics precisely. Our approach presented is using patterns of correct responses.

Finally, our current monitoring strategy can be labeled as active measurement, it means that we are establishing a connection to the monitored service. This method has its benefits but it is not always the best choice because it could interfere with the obtained QoS measurements, for this reason we plan to build measure instruments capable to work according to conservative strategies which won't need to establish connections but require to be placed nearer in the client or the service network.

Acknowledgements

This work has been supported by the research projects ADICT, TIN2007-64753, MCyT, Spain, and SODA FIT-340000-2006-312. Marc Oriol has a FPI grant bound to the project TIN2007-64753.

References

[1] D. Ameller, X. Franch. "Service Level Agreement Monitor (SALMon)". Composition-Based Software Systems, 2008. ICCBSS 2008.
[2] International Organization for Standarization. ISO Standard 8402: Quality management and quality assurance-Vocabulary, 1986.
[3] M.P. Papazoglou. "Service-Oriented Computing: Concepts, Characteristics and Directions". In Proceedings of the Fourth International Conference on Web Information Systems Engineering, p.3, December 10-12, 2003.
[4] International Organization for Standarization. ISO/IEC Standard 9126: Software Engineering – Product Quality, part 1. 2001.
[5] J. P. Carvallo, X. Franch, C. Quer. "Managing Non-Technical Requirements in COTS Components Selection". RE 2006.
[6] NESSI Group. "NESSI Open Framework – Reference Architecture, State of the art report". 2008

A quality model for service monitoring and adaptation *

Cinzia Cappiello[1], Kyriakos Kritikos[1], Andreas Metzger[2], Michael Parkin[4], Barbara Pernici[1], Pierluigi Plebani[1], and Martin Treiber[3]

[1] Politecnico di Milano – Dipartimento di Elettronica e Informazione, Italy
{kritikos, pernici, cappiell, plebani}@elet.polimi.it
[2] University of Duisburg-Essen – Software Systems Engineering, Germany
Andreas.Metzger@sse.uni-due.de
[3] Vienna University of Technology – Distributed Systems Group, Austria
m.treiber@infosys.tuwien.ac.at
[4] Tilburg University – Department of Information Systems and Management, Austria
m.s.parkin@uvt.nl

Abstract. Adaptive Service Based Applications (SBAs) can become a reality with the advent of sophisticated monitoring and adaptation mechanisms. In this paper, the main focus is on defining quality and how it can be exploited by these monitoring and adaptation mechanisms. To this end, we propose a quality model for SBAs and their infrastructure, new techniques for predicting quality and different types of quality-based adaptation actions for SBAs.

1 Introduction

Based on the Service Oriented Architecture (SOA), Service Based Applications (SBAs) can be built from simple or complex services based on functional and non-functional requirements usually provided by a user. This construction of SBAs is usually performed either at design-time or run-time with the help of a service composition engine. As it is already known, the design-time construction of SBAs has the limitation that the constructed complex service can fail in many ways: one of its main component services may not be available as it has reached its capacity limit or it has produced erroneous output or it has failed or the network connecting this service with the outer world is not available. For the above reasons, the run-time construction of SBAs is preferred as it can solve the above problems, for instance, by substituting the faulty service with another one offering the same functional and similar quality capabilities of the faulty one. However, substituting a faulty service with a new one does not always solve the problem. Sometimes it is preferable to re-execute the faulty service with the same or new input parameters or to compensate this service with a compensation action defined within the service management interface in case we are talking about transactional services. Moreover, in case the faulty service is substituted with a new one, maybe the remaining execution plan has to be changed in order to still satisfy the user functional and quality requirements or violate the quality requirements in the smallest possible way.

*The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube) and the Italian FIRB Project TEKNE

1.1 Quality

QoS research gained a lot of attention during the last years in the field of Service Oriented Computing (SOC). This is due to the role that QoS can play in the life-cycle of services [1]. In particular, QoS can be used for filtering and selecting between functionally equivalent services [1], for the design time and runtime selection of component services of a composite service and for adapting services when their promised service level is violated [2]. As QoS is actually a set of non-functional properties, several QoS taxonomies along with their QoS attributes have been proposed. Sabata et. al. [3] present a taxonomy for the specification of QoS in distributed systems. In their approach, the taxonomy is a hierarchical structure that is divided into two major classifications: *metrics* and *policies*. Metrics such as performance (divided into *timeliness*, *precision*, and *accuracy*) measure quantifiable QoS attributes. Policies provide strategies to cope with changing situations and define renegotiation strategies. The work presented in [4] proposes a quality extension to UDDI that encapsulates QoS information which distinguishes four basic classes of QoS attributes, namely *runtime related* QoS, *transaction support related* QoS, *configuration management and cost related* QoS and *security related* QoS. Truong et. al. [5] introduce a framework for monitoring Grid services with respect to QoS attributes. The authors define an extensive, hierarchical QoS classification schema that consists of four main categories, namely *cost*, *dependability*, *configuration* and *performance*.

Basic sets of QoS attributes are discussed in several papers. In [6] a basic set of QoS attributes include *availability*, *accessibility*, *integrity*, *performance*, *reliability*, *regulatory*, and *security*. Zeng. et. al. [7] introduce five major quality criteria for atomic services: *execution price*, *execution duration*, *reputation*, *reliability*, and *availability*. The authors use these criteria in a linear programming approach to select optimal execution plans for composite services. In PAWS [2], the needed functionalities for selecting services, negotiating and maintaining quality through adaptation are discussed.

As it can be seen from the above analysis, there are many service quality models that focus on SOC or Grid Computing but there is no standard and rich service quality model that can be used across multiple scientific disciplines. Moreover, all quality-based service composition and adaptation approaches, that take advantage of these quality models, stay only at the service level and neglect the infrastructural one, while they also perform re-active adaptation trying to repair the problem after it is created and sensed.

1.2 Self-Adaptation

In addition to run-time responses to failures of the service-based application (see above), the highly dynamic environment under which services operate imposes new challenges for engineering and provisioning SBAs. SBAs have to be *flexible* and *adaptable*. By *flexible* we mean that the SBA should be able to change its behavior according to variable execution contexts, while by *adaptable* we mean that the SBA should be able to execute even if the conditions at runtime differ from those assumed during the SBA's initial design. Flexibility can be achieved if SBAs are designed in such a way that are able to self-adapt to timely-respond to changes in their context or their constituent services or the user preferences and context. A necessary condition for achieving this property is