An Architecture to Measure QoS Compliance in SOA Infrastructures

Alexander Wahl, Ahmed Al-Moayed, Bernhard Hollunder
Department of Computer Science
Hochschule Furtwangen University
Furtwangen, Germany
alexander.wahl@hs-furtwangen.de *ahmed.almoayed@hs-furtwangen.de* *bernhard.hollunder@hs-furtwangen.de*

***Abstract*—In the last couple of years Service Oriented Architecture (SOA) has gained in importance and became widely used. With increased acceptance the demand of non-functional requirements, so-called Quality of Service (QoS) attributes, arose. QoS attributes were applied to SOA environments, resulting in QoS-aware SOAs. Within the QoS-aware SOAs, compliance to the desired QoS in general is not easy to measure. In this work, we offer a solution architecture to measure actual data that relate to QoS attributes. Further, these data are compared to their target state. The aim is i) to evaluate compliance of the entire QoS-aware SOA to the desired QoS attributes and ii) to start suitable activities. In a proof of concept a solution architecture, based on the technique of Complex Event Processing, is implemented. Within this proof of concept selected QoS attributes are applied and compliance to the SOA is measured.**

***Keywords*-Service Oriented Architecture; Quality of Service; QoS Attributes; Complex Event Processing;**

I. INTRODUCTION

Service Oriented Architectures (SOA) are a design paradigm to compose and structure loosely coupled components to form distributed applications. SOA offers a way to map business processes from the business domain to the technical domain of computer systems. After a business process is analyzed and its single activities are identified, the individual activities are mapped to the technical domain by implementing corresponding services. To execute a business process, the services are called in corresponding sequences.

Web Services (WS) are the predominant technology to realize the services of a SOA. They are used to implement the functional aspects of business processes, which in brief define the input/output behavior of a component. Additional, in many business domains it is crucial to fulfill non-functional requirements. A non-functional requirement, or Quality of Service (QoS) attribute, specifies *how* a component is supposed to behave. Examples for QoS attributes are robustness, security, performance, scalability and accounting. More detailed descriptions of QoS attributes in SOA can be found in [1] and in [2]. Within a SOA equipped with QoS attributes, which we call QoS-aware SOA, desired QoS attributes are described in a formal manner. Therefore a policy language is used typically. These so-called service policies define the target state of the desired QoS attributes.

The crucial need to fulfill non-functional requirements is reflected by QoS attributes applied to SOA. For example, consider security aspects, like integrity and confidentiality, which are applied to Web Services using WS-Security [4]. When implementing a SOA from scratch, QoS attributes can be designed from the beginning. But many SOAs are grown, which means that they expanded over time. Such SOAs often integrate existing legacy applications and enhance them by QoS attributes they were not equipped with before. Also QoS attributes may have changed several times. So how can compliance to QoS attributes be measured? For example, assume a SOA of high complexity that has grown over time. For this SOA, a roles and rights model is specified. During runtime violations to the roles and rights specification are observed. In consequence, the entities that caused the violations are to be fixed. Further, the SOA is to be analyzed on compliance of all entities to the given roles and rights policy. But how can such an analysis be performed efficiently?

An efficient solution is to monitor and analyze the QoS attributes of a SOA at dedicated measurement points. Monitoring approaches were already elaborated in several publications. Berbner et al. [5] selected Web Services (WS) based on QoS properties guaranteed by Service Level Agreement (SLA). They ensured compliance to a given SLA using a monitoring component, which was not described in detail. Zeng at al. [6] introduced a high-performance QoS monitoring system. In their work they focus on service monitoring architecture and QoS metric computation. Artaiam and Senivongse [7] described a JMX-based monitoring extension of application servers for selected QoS attributes. Michlmayr et al. [8] integrated existing client-side and server-side monitoring approaches using Complex Event Processing (CEP). Finally, Oriol et al. [9] described the monitoring of adaptable SOA. We will go into more detail on these approaches and the differences to our work later in Section VI.

In this work, we propose a solution architecture that evaluates compliance of a QoS-aware SOA to the desired QoS attributes. The solution architecture monitors the SOA at dedicated measurement points. The thereby collected actual data are filtered according to a filter policy and compared to target states specified in the service policies of the SOA.

A SOA application landscape consists of several compo-

 ISBN: 978-1-61208-105-2

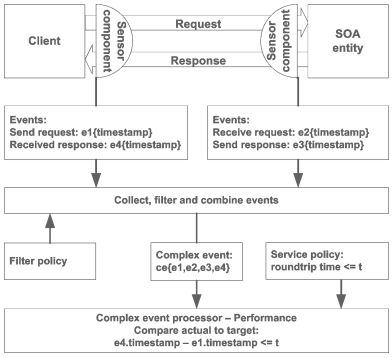

Figure 1. Mechanism for the exemplary QoS attribute performance

nents like services, processes, application servers, hardware platforms, etc. we refer to as SOA entities (SE). Dedicated measurement points at the SOA entities are equipped with sensor components (SC). The characteristics of the sensor components differ depending on the measurement point. A sensor component may be some source code attached to a services source code, a JMX client component, or even a GUI element, like buttons, sliders, etc. In common, these sensor components collect actual data from the SOA entities.

A sensor component emits events that include the information needed for further analysis. The included information as well as the necessary number of events strongly depends on the desired QoS attribute. The latter depends, among others, on the number of measurement points. If several events are needed they are combined, which generates an abstract event, also called a complex event [10].

Figure 1 visualizes the mechanism for the QoS attribute performance:

1) In the sensor components events including timestamps are emitted.
2) The events are collected and filtered based on filter policies. The filter policies thereby describe *what* events emitted by *which* sensor components are combined to a complex event.
3) A complex event is generated including the collected events needed.
4) The complex event is processed by a complex event processor. Incoming complex events are analyzed on compliance to QoS attributes defined in the service policy of the SOA.

In our example target roundtrip time is compared to the calculated actual roundtrip time. Therefore two of four measurement points (send request, receive request, send response, receive response) are used.

In summary: With a grown SOA it is desirable to evaluate compliance to specified QoS attributes. The combination of SOA and CEP results in a highly flexible approach to detect compliance to or violation of QoS attributes constraints by target-actual comparison. This work offers a solution architecture that is able to perform such a target-actual comparison. The comparison is not limited to information extracted from single services only, but also from whole business processes, the application server and/or the system platform. By filter policies analysis can be controlled to single QoS attributes or SOA entities. The solution architecture is able to analyze QoS attributes from technical domain as well as from business domain. Exemplary QoS attributes are performance, roles and rights, reliability, schedule and cost. The solution architecture also is able to react in various ways, reaching from display on a dashboard towards automatic anatagonization using dedicated escalation applications.

The paper is organized as follows: The next section gives a brief description of the requirements this architecture has to address. Afterwards our solution architecture is described in detail, including a statement on coverage of the given requirements. A realization of the solution architecture and exemplary implemented QoS attributes are described in the proof of concept section. We then provide a discussion on related work. Finally, a description of potential future work and our conclusion is given.

II. ARCHITECTURAL REQUIREMENTS

In a SOA application environment, there are several situations where it is desirable to support QoS attributes. Remember the QoS attributes performance, schedule and cost, which relate to an ordering process with a due time for shipment. But how can compliance of a SOA to its QoS attributes be shown? A flexible and powerful solution architecture is required to measure compliance of the SOA to its QoS attributes.

The aim is to evaluate conformance to desired QoS attributes. Thereby, QoS attributes may relate to technical domain, like performance, or business domain, like cost and schedule. Ability to handle QoS attributes of both domains is a requirement.

The solution architecture

1) needs to be able to determine the actual state of a SOA concerning its desired QoS attributes and
2) to compare this actual situation to the defined QoS attributes.

To sum up, the architecture performs a target-actual comparison on QoS attributes described in the service policies.

The actual situation concerning QoS attributes is determined based on relevant data captured from SOA entities. The solution architecture therefore needs to provide an appropriate capturing mechanism. Relevant data are to be captured by sensor components at dedicated measurement points located at specific SOA entities. The SOA entities thereby may be of different type (service, process, application server, etc.), possibly distributed and under diverse governance. The need of source code change at the SOA entity to apply the sensor component is to be kept to a minimum to increase

 ISBN: 978-1-61208-105-2

acceptance, applicability and interoperability. The aim is to minimize necessary modifications to the SOA.

Next, the solution architecture needs to provide a configurable filter component to filter the data according to defined filter policies. The filter policies define i) on *which* QoS attributes target-actual comparison is to be performed ii) *what* data are to be captured and iii) *where* the data are to be captured. For a filter policy a declarative language ,e.g., WS-Policy [3], is to be used to enable modification the filter behavior without the need of recompilation.

The relevant data captured at the SOA entities can be seen as a kind of events that contain the appropriate data for further analysis. The solution architecture needs to be able to combine desired events to a more abstract event, as described before in the performance example.

Another requirement is to offer a flexible mechanism that allows to react on specified conditions. Appropriate activities thereby include execution of applications for active antagonization (e.g., cancellation of request execution) as well as the compilation of statistics (e.g., display of statistics on QoS attribute conditions).

Finally, the solution architecture should be based on standards and well-known frameworks. By using standard frameworks and products the applicability to and the interoperability of different environments is increased.

III. Solution Architecture

A. Description of solution architecture

In this section, an architecture that meets the requirements in the previous section will be presented. Figure 2 gives a basic overview of the solution architecture, which consists of four components: i) the QoS-aware SOA, ii) the filter component, iii) the analysis and statistics component and iv) the escalation component. The QoS-aware SOA is already existing. It is to be enhanced and monitored by the other three components to enable a target-actual analysis on desired QoS attributes.

In a QoS-aware SOA, the desired QoS attributes are defined by service policies, which describe the target states. To determine the actual state of QoS attributes, sensor components are applied to SOA entities at specific measurement points. The sensor components are responsible for emitting events with collected actual data. The data was read and composed from the SOA entities and sent to the monitor and filter component.

The monitor and filter component has two tasks: i) to observe the SOA environment and to collect the emitted data from the sensor components; ii) to filter the received data according to filter policies.

The term policy is used in both, QoS-aware SOA and monitor and filter component. It is to be interpreted depending on the context: Within a SOA, the term policy refers to service policies, for example W3C standard WS-Policy [3], that specify the target non-functional behavior of a certain

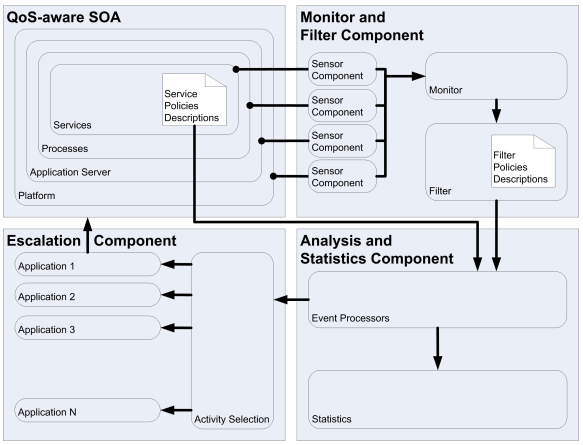

Figure 2. Basic overview on the solution architecture

Web Service. Within the monitor and filter component, the term policy defines the behavior of data filters.

The analysis and statistics component compares actual data to service policies of the SOA environment. To do that, the filtered data are compared with the service policy. This component also creates statistics on QoS attribute conformance or violation. If an escalation is desired, this component will trigger the next component to perform the appropriate escalation.

Finally, the escalation component provides desired activities, like solutions to solve service policy violations on the SOA environment. The activity strongly depends on the objectives, as will be shown later by example in Section V.

B. Why does the architecture meet the requirements?

First, the solution architecture is able to collect data from the SOA entities. For example, with performance it is able to collect timestamps data whenever a SOAP message is initiated or received (see Figure 1). The architecture attaches sensor components to SOA entities like e.g., Web Services. These components collect data and send them as an event to the filter component. Once an event is received, the filter component checks its filter policy in order to decide what to do with such an event.

Second, our architecture keeps code modifications to SOA entities to a minimum. Ideally, none of their source code will be changed. As mentioned before, the monitor and filter component must be able to collect data from the SOA environment. Therefore, data from the SOA entities needs to be sent to the monitor and filter component. There are different ways to put this into effect. Either new code fragments must be added to the SOA entities, or realized by sensor components, like SOAP message handlers, that work as proxies for the incoming and outgoing SOAP messages. The sensor component could be based on different technologies, such as JMX or even ESB events components. Also, a network sniffer could be used as a sensor component

 ISBN: 978-1-61208-105-2

to analyze network traffic for a certain request or response. In summary: Several options do exist to implement the handler approach. The decision, which one to use, strongly depends on different factors; for example, to which extend the SOA entities are allowed to be modified.

All the sensor components have in common that they are attached to measurement points within the SOA. The measured data then need to be transferred to the monitor and filter component. Therefore, events are created and emitted at the sensor components. All incoming events at the monitor and filter component are filtered according to the filter policy. The events that the policy allows are forwarded to the analysis and statistic component. Other events will be ignored. Optional, all incoming events are saved permanently.

At the analysis and statistics component the events that passed the filter are further processed. If needed, the events are combined to complex event. Analysis is performed on the complex events. The solution architecture described here is able to handle both, single events as well as complex events.

Compliance to desired QoS attributes are detected either based directly on the events or on complex events. This method enables the system to measure compliance of a SOA to a single quality attribute. Moreover, several complex events again can be combined with events or complex events. With this mechanism, the solution architecture is also able to measure compliance to combinations of several QoS attributes. For example, if several QoS attributes in combination have impact on other QoS attributes.

The architecture is based on standards, frameworks and products. However, some components, like sensor components, need to be implemented from scratch.

The escalation component is an important part in this architecture. It provides a way to initiate appropriate escalation measures as well as a way to start a certain activity in case of a service policy violations. The functionality of this component strongly depends on the kind of violation and predefined objectives of the escalation component.

In a nutshell: The provided solution architecture fulfills all the requirements specified in Section II. It is able to monitor diverse entities of a SOA application landscape. Changing the source code within the SOA landscape is kept to a minimum by using sensor components to the monitored component. Events emitted by the sensor components are collected by a monitor and filtered by an adjustable filter. The filtered events are further analyzed to measure compliance of the SOA to QoS attributes, which are described in a services' policy. Also based on these events statistics are generated. This architecture is based on standards and well-known frameworks. Finally, the architecture offers an escalation component, which is used to trigger desired activity in case of compliance to or violation to a services' policy.

IV. Solution Architecture Details

For each of the four systems of the solution architecture we will in detail describe the input and output data, the performed tasks and the entities within the systems.

A. SOA Application Landscape

The SOA application landscape consists of several entities that we named SOA entities. But what are these entities? Obviously, there are the different kinds of services, like component service, composite service, workflow service, etc. In addition to that there are processes, realized by appropriate combinations of services. An entity of the SOA application landscape is also the application server itself. Finally, the platforms (operating system, hardware components, etc.) are such entities, too. Typically, these SOA application landscapes are huge and grown distributed systems. In consequence, these systems are highly complex, and so is the communication structure within.

B. Monitor and Filter Component

All the SOA entities are to be interlinked with a monitor and filter system to determine the actual states of the QoS attributes of the entire SOA entities. The applied sensor components collect actual data, encapsulate these data in events and finally emit these events. The sensor components are situated within the SOA application landscape, but they are part of the monitor and filter component. The sensor components detect and indicate changes in state of the SOA entities. As a simple example: At the time a service receives a request, for example a SOAP message, its state changes. This change in state is detected by the sensor component, which is situated prepending to the service. With rights and roles the sensor component will then determine the user principal of the SOAP message. Afterwards an event that contains (besides other information) the principal is generated and emitted. At a more global view each SOA entity equipped with such a sensor component emits a corresponding event once a SOAP message is received. Received events are optionally stored before the further processing, like the filter mechanism. Because of the storage the system is enabled to perform retrospective analysis.

The received events are filtered according to the desired filter behavior. The filter behavior is described by the filter policies in a declarative manner, for example using XML. For example: Suggest a SOA equipped with sensor components for performance and for rights and roles. In the filter policy, the desired QoS attribute (rights and roles) and its corresponding sensor component IDs are described. For each received event the sensor component ID is compared to the ones specified within the filter description. If matching, the event is forwarded to the corresponding subsequent processing unit, as described later. So in our example events related to rights and roles pass the filter, events related to

 ISBN: 978-1-61208-105-2

performance do not pass. The output of the monitor and filter component are events.

C. Analysis and Statistics Component

The output events of the monitoring system is the input for the analysis and statistics system. Within this component the input event vectors are further processed. For each event vector, respectively the corresponding QoS attribute, an event processor is provided. With the example stressed before two event processors are provided - one for roles and rights and one for performance.

Within the event processors the events are combined to complex events and analyzed according to the QoS attributes requirements. In a first step, the service policies (located in the SOA) of the SOA entities are read by the event processors. As described above, the service policy contains the description of the target state for a QoS attribute. Next, from the event vector the events that correspond to the SOA entity are extracted and combined for further analysis. Based on these complex events the actual state concerning the QoS attribute is determined. In the performance example stressed before, the actual message transfer times and the processing time of a SOA entity are determined by four events. The filtered events of the same SOA entity ID and message ID, which corresponds to receiving a request and sending the response, are combined to a complex event. The processing time can now be determined from the complex event by subtraction of the timestamps. Finally, the result (actual state) is compared to the target state. The result indicates compliance to or violation of target state. In either case, statistics can be generated, like performance violation per time unit or a list of principals for each SOA entity.

In a nutshell, the analysis and statistics system is a collection of event processors and generated relations. For each quality attribute a dedicated event processor is needed, since combination of used events and attached additional information is individual for each quality attribute. From the results of the event processors desired relations are generated. The outputs of these components are QoS attributes compliance or violation vectors and the generated relations.

D. Escalation Component

The final component of our solution architecture is the escalation component. The component in essence is a collection of individual application that establishes certain activities based on the output of the analysis and statistic component. These activities are highly individual. For example, on performance violation an application that issues a ticket to a ticketing system might be started. Or a kind of management application that upscales resources for the SOA application landscape. Another option is an information cockpit application. On compliance of actual states to target states the cockpit indicates green condition, on violation red condition. Additional information, like performance status of the last hour, may also be displayed by gaining access the corresponding statistics.

Relationship among event and escalation activity can be 1-by-1 or 1-by-n. This means that for an individual result of the analysis and escalation system, like a QoS attribute violation, several escalation activities may be issued. For example, on performance violation a ticket is issued and the resources available to the SOA application landscape are upscaled. By these examples it also becomes obvious that the escalation system does not necessarily influence the SOA application landscape. Issuing a ticket does not directly influence the SOA, but resource upscaling does.

V. Proof of Concept

In the following, we will describe our proof of concept implementation of the prior described solution architecture.

A. System Overview

The solution architecture in general is realized based on several established frameworks and standards. For the SOA application landscape we used the Enterprise SOA (eSOA) showcase by q-ImPrESS [11]. For the monitor and filter component as well as for the analysis and statistics component GlassFishESB with IEP runtime component [12], [13] is used. IEP includes an implementation of CEP. The event processors are implemented using NetBeans and IEP design time component. At runtime the event processors are hosted at the GlassFishESB application server. To store events the default setting of the IEP component, Apache Derby, is used.

The eSOA showcase is a set of exemplary applications from the domain of order and supply chain management forming a non-trivial service oriented system. It implements simulators for customer-relationship-management (CRM), product data management (PDM), pricing, inventory, order and shipment. The showcase is based on Web service technology and Java. For communication SOAP messages are used.

Sensor components are applied to the Web services of the eSOA showcase. In detail, we implemented SOAP message handlers for some exemplary QoS attributes, as we will describe later. On server-side the sensor components are positioned before the individual Web services, as can be seen in Figure 1. Before, in this case, means that the SOAP message handler is positioned between the client and the Web service. In consequence, a SOAP request first passes the SOAP handler before it is received by the Web service. And a response of the Web service first passes the SOAP handler before it is received by the client. In addition, subsequent means that a request message first passes the SOAP handler and is then sent to the Web service. A response message first passes the SOAP message handler and is afterwards forwarded to the client. On client-side the situation is vice-versa, if a client-side sensor component is needed.

 ISBN: 978-1-61208-105-2

All sensor components have in common that they emit events. The characteristics of the events depend on the QoS attributes. The emitted events are collected by the IEP component of GlassFishESB. Further, these events are filtered using corresponding event processors. For the filter policy description XML is used. The filter policy thereby consists of the QoS attribute to pass the filter.

After passing the filter the events are processed by the analysis and statistics component. The analysis is performed by event processors. An event processor basically consists of an input stream and some processing entities that end in an output stream. For each QoS attribute a corresponding event processor is implemented. The individual event processors are designed and executed using the IEP design time and runtime components.

B. Exemplary quality attribute implementations

In our proof of concept, we implement selected QoS attributes. In a first step, we implemented the QoS attribute of performance. In more detail: Based on SOAP message handlers (client-side and server-side) we determined request and response transmission time, service calculation time and roundtrip time.

Also from the technical domain, roles and rights are implemented. In brief: The individual Web Services of eSOA are called by clients. Each client has an individual identification. The right and roles model defines which Web Service may be called by a certain client. The motivation for this scenario is the analysis of a grown SOA on conformance to given right and roles model. The task is to generate a statistic on principals of service requests.

Different sensor components are implemented for this task. A first kind of sensor component is a SOAP message handler at server-side prepending to a Web service. Within this SOAP message handler the principal of the incoming request is determined from the SOAP message context. Then, an event including this information on the principal is emitted to the corresponding event processor. In brief, an IEP component is a JBI module, that is added to and deployed with a composite application. The event processor appears as a Web Service that, in our case, uses SOAP for communication. At the sensor component, respectively our SOAP message handler, the created event, in essence, is a SOAP message. The structure of the SOAP message is defined in the JBI modules WSDL. Emitting the event means, that this SOAP message is sent to the Web Service exposing the event processor.

With regards to righs and roles, one concrete example would be the SOAP request from the client is processed by the SOAP handlers handleRequest() method. From the MessageContext the principal of the SOAP request is determined using method getUserPrincipal(). Next, a SOAP message including the principal is generated and sent to the event processors Web Service. At the event processor the principal is compared to a database that contains the roles and rights model, and any violation is indicated.

An alternative kind of sensor component uses the JMX interface of the application server to extract the information on principal of request on hosted services. Both kinds of sensor components do solve the task, and either can be used depending on existing restrictions. The advantage of the second is that it does not touch services at all, but accesses to application server management console is needed.

Another exemplary QoS attribute is from the business domain: schedule. With this a due date for shipment is agreed. Motivation for this scenario is, for example: Shipment of a placed order is guaranteed within 24 hours, otherwise a certain discount is allowed. The task is to ensure this in due time. If the due time is exceeded, statistics on time overruns are to be generated and discount is to be given out. Again, the Web services are equipped with SOAP message handler as sensor components. From the SOAP messages the information on order ID, actual time and due time is extracted. With that information time overruns are detected and statistics are generated at the analysis and statistics component. On violation an application within the escalation component automatically allows a discount.

VI. Related work

In 2005, an architecture based on Web services including comprehensive QoS support was described by Berbner et al. [5]. Within it particular Web services are composed to workflows. Thereby, the selection of Web service is based on their QoS properties that they guarantee in Service Level Agreement (SLA). To ensure compliance to given SLAs a monitoring component is mentioned, but not described in detail.

The design and implementation of a high-performance QoS monitoring system was presented by Zeng et al. [6]. Their two main issues on the monitoring system are the service monitoring architecture and the QoS metric computation. Within their work a *QoS observation metamodel* with three types of monitoring context, one on processes and two on services, was developed. So the measurement points for QoS monitoring are limited to the services and processes. Our work is a more general approach, since we do not limit ourselves to services and processes, but support any SOA entity as described before.

A JMX-based monitoring extension of Java system application server for the QoS attributes availability, accessibility, performance, reliability, security and regulatory was described by Artaiam et al. [7]. They also give a detailed description of QoS attributes metrics. However, they are limited to service-side monitoring, which means QoS monitoring of services within the application server (GlassFish). Client-side QoS monitoring is not included. Our approach explicitly enables both, server-side and client-side monitoring.

 ISBN: 978-1-61208-105-2

The integration of an existing client-side monitoring approach and a server-side monitoring using CEP to monitor SLA was elaborated by Michlmayr et al. [8]. For CEP the open source implementation ESPER is used. For monitoring at client side so-called QoS monitoring schedules are used that specify that certain services are monitoring in certain time intervals. On server side a .NET technology is used, which i) is a limitation to certain server infrastructure and ii) also limits the service implementation to .NET technology, as is mentioned by the authors. The described solution architecture of the authors also includes a notification mechanism to subscribers on detected SLA violations. Their approach is similar to our approach. However, they use ESPER, focus on .NET technology and monitor at dedicated points in time. In contrast, our work uses the IEP component. Although our proof of concept uses Java, it is also applicable to other technologies, like .NET. Next, we use continuous monitoring rather than dedicated points in time. And finally, we provide a mechanism to trigger activities.

Monitoring of adaptable SOA was described in Oriol et al. [9]. The focus is on dynamic adoption of a QoS-aware SOA. Within the QoS-aware SOA QoS attributes are stated using SLA. The current QoS values are monitored by a monitor component and compared to the stated SLA at an analyzer component. SLA violations are shown to a decision maker component, which is able to perform adjustments within the SOA. In contrast to our work, this approach focuses on adjustments within the SOA. Our approach is more general. In our solution, architecture adjustments are just one aspect of possible escalation activities.

VII. Conclusion

By the combination of SOA system, monitor and filter component, analysis and statistics component and escalation system a versatile and powerful tool is available to analyze SOAs on compliance to the defined QoS attributes. Using this solution architecture compliance analysis is not limited to services and processes, but also includes other SOA entities, like application server and platform. This enables for several QoS attributes yet not supported, especially QoS attributes, like in our schedule example. With the escalation component a variety of activities can be carried out. These activities may be of more passive nature, like to issue a ticket. Or of active nature, like enabling for a self-scaling SOA. The boundaries of the given approach have not been yet explored.

Our perspective is to enrich the existing systems with additional QoS attributes that are not yet supported. Therefore, it is necessary to determine which QoS attributes are requested from both, the technical and the business domain. And which of these QoS attributes can be formalized and further supported. A related question is the use of alternative description languages for QoS attributes.

We will also implement additional tools to support developers with an interest for our approach. The generated tools are to be added to different IDEs. A tool chain to define additional QoS attributes, to equip Web Services with these, and to deploy such Web services is implemented. Also additional components to ensure compliance to these QoS attributes will be provided.

Acknowledgments

We would like to thank the anonymous reviewers for giving us helpful comments. This work has been partly supported by the German Ministry of Education and Research (BMBF) under research contract 17N0709.

References

[1] "Web services quality factors v1.0," http://www.oasis-open.org/committees/download.php/38611/WS-Quality_Factors_v1.0_cd02.zip, accessed at 27. Aug 2010

[2] L. O'Brien Lero, P. Merson, and L. Bass, "Quality attributes for service-oriented architectures," in *Systems Development in SOA Environments, 2007 (SDSOA'07)*

[3] "Web Services Policy 1.5," http://www.w3.org/TR/ws-policy

[4] "Web Services Security v1.1," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[5] R. Berbner, O. Heckmann, and R. Steinmetz, "An Architecture for a QoS driven composition of Web Service based Workflows," in *Networking and Electronic Commerce Research Conf. (NAEC'05)*

[6] L. Zeng, H. Lei, and H. Chang, "Monitoring the qos for web services," in *Proceedings of the 5th Int. Conf. on Service-Oriented Computing (ICSOC'07)*

[7] N. Artaiam and T. Senivongse, "Enhancing service-side qos monitoring for web services," *ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'08)*

[8] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Comprehensive qos monitoring of web services and event-based sla violation detection," in *Proceedings of the 4th Int. Workshop on Middleware for Service Oriented Computing (MWSOC'09)*

[9] M. Oriol, J. Marco, X. Franch, and D. Ameller, "Monitoring Adaptable SOA-Systems using SALMon," in *Workshop on Service Monitoring, Adaptation and Beyond*

[10] D. Luckham, *The power of events*, 5th ed. Boston, Mass. [u.a.]: Addison-Wesley, 2007.

[11] Q-ImPrESS, "Enterprise SOA Showcase." http://www.q-impress.eu/wordpress/software/, accessed at 27. Aug 2010

[12] S. Microsystems, "Glassfish application server." https://glassfish.dev.java.net/, accessed at 27. Aug 2010

[13] Oracle, "Intelligent event processing (iep)." https://open-esb.dev.java.net/IEPSE.html, last accessed at 27. Aug 2010

 ISBN: 978-1-61208-105-2