Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection

Anton Michlmayr, Florian Rosenberg, Philipp Leitner, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Wien, Austria
lastname@infosys.tuwien.ac.at

ABSTRACT

In service-oriented systems, Quality of Service represents an important issue which is often considered when selecting and composing services. For receiving up-to-date information, non-functional properties such as response time or availability can be continuously monitored using server- or client-side approaches. However, both approaches have strengths and weaknesses. In this paper, we present a framework that combines the advantages of client- and server-side QoS monitoring. It builds on event processing to inform interested subscribers of current QoS values and possible violations of Service Level Agreements. These events can trigger adaptive behavior such as hosting new service instances if the QoS is not as desired. We describe our QoS monitoring approach in detail, show how it was integrated into the VRESCo service runtime environment, and evaluate the accuracy of the presented monitoring techniques.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

QoS Measurement, Performance, Reliability

1. INTRODUCTION

In the past few years, Service-oriented Architecture (SOA) and Service-oriented Computing (SOC) [9] have emerged as a paradigm for addressing the complexities of distributed applications, and have finally gained acceptance from both industry and research. The overall idea is based on well-established standards for loose coupling and platform-independent interface descriptions. Web services represent the most common realization of SOA that build on the main standards SOAP and WSDL. Over time, several standards and specifications have been introduced for different issues in Web services research.

MW4SOC '09, November 30, 2009, Urbana Champaign, Illinois, USA

Quality of Service (QoS) plays a crucial role in service-oriented systems, for instance during service selection. When integrating external services into business processes, it is important to consider the quality guarantees of the service provider. Therefore, Service Level Agreements (SLAs) [4] are used to define the expected QoS between service consumer and service provider. In general, QoS attributes can be classified as deterministic or non-deterministic. The former indicates that the QoS attribute is known before a service is invoked (e.g., price, security, etc.), while the latter includes all attributes that are unknown at service invocation time (e.g., response time, availability, etc.). For non-deterministic QoS attributes, monitoring approaches can be used to continuously measure current QoS values.

Conceptually, there are two main approaches for QoS monitoring: Server-side monitoring is usually accurate but requires access to the actual service implementation which is not always possible. In contrast, client-side monitoring is independent of the service implementation but the measured values might not always be up-to-date since client-side monitoring is usually done by sending probe requests (i.e., test requests that are similar to real requests). In this paper, we aim at combining the advantages of both approaches which has been realized in the Vienna Runtime Environment for Service-oriented Computing (VRESCo) [8]. Therefore, we have linked an existing client-side QoS monitoring approach [14] together with server-side monitoring based on Windows Performance Counters [18]. Furthermore, event processing is used to integrate both approaches and provide means to monitor SLAs. If SLA obligations are violated, notifications are sent to interested subscribers using E-Mail or Web service notifications.

The contribution of this paper is threefold: Firstly, we present two approaches for monitoring QoS attributes of Web services. Secondly, we show how these approaches have been integrated into VRESCo, and how simple SLA obligations can be defined and monitored using event processing. Thirdly, we evaluate the accuracy of the two monitoring approaches and discuss how combining them leads to a comprehensive QoS monitoring framework.

The remainder of this paper is as follows. Section 2 shows our QoS model, and describes the client- and server-side monitoring approach. Section 3 introduces the VRESCo runtime, shows how QoS monitoring has been integrated and how SLA obligations are monitored. Section 4 presents the accuracy of the QoS monitoring approaches and briefly discusses the usefulness of combining them. Finally, Section 5 gives related work and Section 6 concludes the paper.

2. QOS MONITORING

In this section, we first briefly introduce the QoS model we have used in our work. Then we present two conceptually different monitoring approaches for Web services which we have integrated into VRESCo.

2.1 QoS Model

There are several definitions of QoS in literature [6,12,19]. Figure 1 depicts our QoS model which consists of the four categories *Performance*, *Dependability*, *Security/Trust* and *Cost/Payment*. In the remainder of this paper we focus on the first two categories since they can be measured automatically, with a special emphasis on those attributes that are currently measured by our approach.

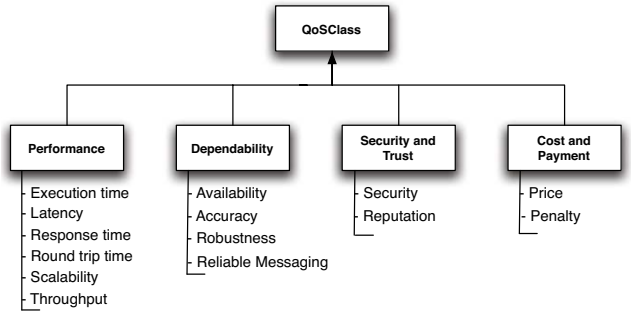


Figure 1: QoS Model

Performance-related attributes of services can be measured over several service invocations (see Figure 2). We consider the following attributes: *Execution time* q_{ex} represents the service invocation time at the provider. This consists of the actual *Processing time* q_{pt} and the *Wrapping time* q_w (i.e., time needed to wrap XML messages). *Latency* q_l defines the time needed for the client request to reach the service. *Response time* q_{rt} indicates the service invocation time at the service consumer (i.e., execution time plus latency), while *Round trip time* q_{rtt} measures the overall time needed for the request at the service consumer (i.e., response time plus wrapping time at the client). *Throughput* q_{tp} represents the number of service requests that can be processed within a given time period, while *Scalability* q_{sc} defines performance behavior of a service when the throughput increases. In general, these performance-related attributes are measured on the level of service operations.

Dependability-related attributes address the ability of services to avoid frequent and severe failures. In contrast to performance-related attributes, they are measured on the service-level. *Availability* q_{av} represents the probability that a service is up and running, while *Accuracy* q_{ac} defines the ratio of successful service executions in relation to the total number of requests. More details about our QoS model (including the formulas to calculate the attributes) can be found in [13].

2.2 Client-side Monitoring

The first approach to address QoS monitoring is done client-side using the QUATSCH tool [14]. The overall idea is to send probe requests to the services that should be monitored. The service invocation is thereby divided into the time periods shown in Figure 2 that correspond to the QoS attributes introduced above.

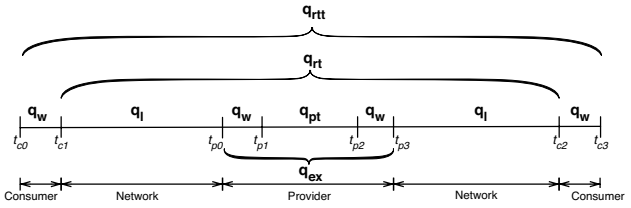


Figure 2: Service Invocation Times

The actual monitoring is done in three phases. In the preprocessing phase, the WSDL files of the services are parsed and stubs are generated. The performance measurement code is thereby weaved into the stubs using aspect-oriented programming (AOP). In the evaluation phase, the services are executed by probing arbitrary values as input parameters. If this is not successful, templates can be used to provide user-defined input. Finally, the result analysis phase stores the results of the evaluation phase in a database.

The interesting part of the client-side monitoring approach is that it is indeed able to accurately measure server-side attributes such as execution time. In QUATSCH, this is done using low-level TCP packet sniffing and analysis by leveraging the TCP handshake (i.e., SYN and ACK packets) to distinguish the different service invocation times [14].

2.3 Server-side Monitoring

The main drawback of the client-side approach is the fact that monitoring is done by regularly sending probe requests (e.g., every 5 minute). Single results should be handled with caution since they represent only snapshots (e.g., the service might be under heavy load when the probe request is sent). Decreasing the monitoring interval might mitigate this problem to some extent but this must be done carefully since short monitoring intervals (e.g., once every second) may finally affect the actual performance of the service.

Server-side monitoring addresses this problem by continuously measuring QoS attributes. Since no probe requests are needed anymore, the measured values represent "real" service invocations. However, as said above, this technique requires access to the actual Web service implementation which is not always possible in practice.

Windows Performance Counters (WPC), especially the counters regarding the Windows Communication Foundation (WCF) [10], are part of the .NET framework and provide such server-side QoS monitoring for Web services [18]. WPC supports a rich set of counters that can be measured at runtime. For our work, we focus on the following counters: *Call Duration* indicates the service invocation time which resembles *Execution Time* in the client-side approach. *Calls Per Second* defines how often a service has been invoked, while *Calls Failed Per Second* represents a similar counter for unsuccessful service invocations. Other performance counters (e.g., *Transactions Flowed Per Second*, *Security Validation and Authentication Failures*, etc.) could also be integrated seamlessly.

As before, monitoring is done in user-defined intervals. The different performance counter values are thereby aggregated and averaged within an interval, and finally re-set at the beginning of the next interval. For instance, if a service has been invoked 3 times, the average response time of these invocations is returned by the counter.

3. QOS/SLA INTEGRATION IN VRESCO

In this section, we show how the presented client- and server-side monitoring approaches have been integrated into VRESCo and how SLA monitoring can be achieved using the event processing engine.

3.1 VRESCo Overview

Before describing the QoS integration in detail, we briefly introduce the VRESCo service runtime environment [8]. The aim of this runtime is to address current SOC challenges and ease engineering of service-centric systems. More precisely, VRESCo addresses service metadata, QoS-aware service selection and service composition, dynamic binding and mediation of services, and complex event processing.

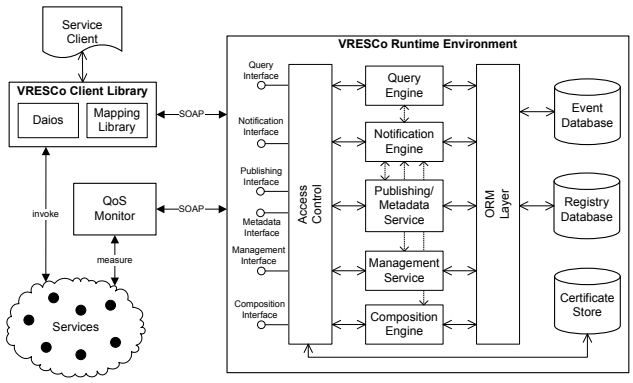


Figure 3: VRESCo Overview

The architecture is depicted in Figure 3. The runtime follows the "Software as a Service" concept and provides its core functionality as Web services that can be accessed using the client library or directly using SOAP. Services and associated metadata are published in the registry database using the publishing/metadata service, while the actual database access is done using an ORM layer. The VQL query engine provides type-safe querying of registry content, while the access control layer is responsible for allowing only authorized and authenticated users access to the VRESCo core services. The event notification engine [7] can be used to inform interested subscribers if events of interest occur (e.g., new service is published, etc.), while all events are additionally persisted in the event database. As discussed later, this event engine is leveraged for our QoS and SLA monitoring approach. Finally, the QoS monitor on the left-hand side represents the QUATSCH monitor [14] that follows the client-side monitoring approach. More information about the VRESCo core services and the service mediation approach are not within the scope of this paper and omitted for brevity. The interested reader is referred to [8].

3.2 QoS Integration

The overall architecture of our monitoring approach is shown in Figure 4. The client-side monitor QUATSCH was first integrated into VRESCo. Users can specify QoS monitoring schedules following the CRON time notation to define which service (or service operation) should be monitored in which time intervals. Since this is a client-side approach, the monitor runs on a dedicated QUATSCH host as shown in the middle of the figure. The actual monitoring is then done using AOP and TCP packet analysis as described in Section 2.2. Once the current QoS values have been measured, they are published into the VRESCo runtime. This is done via the QoS Manager that receives the values and, in turn, publishes them as corresponding QoS events into the event notification engine.

The monitoring is done regularly based on the user-defined monitoring schedules. The set of resulting QoS events represents the history of QoS information as collection of single QoS snapshots. To make these values easily accessible, they are aggregated by a QoS aggregation scheduler task on a regular basis, and finally attached to the corresponding service (or service operation).

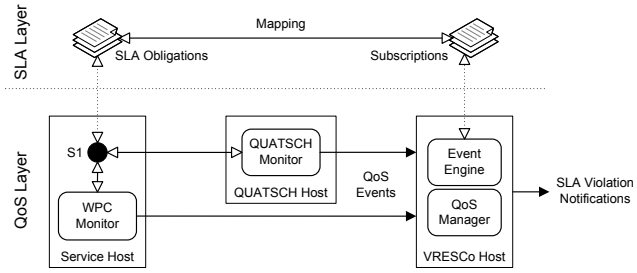


Figure 4: Monitoring Approach

As already discussed, the client-side approach has both strengths and weaknesses. Therefore, we decided to additionally integrate a server-side approach using WPC which is an integral part of the .NET framework. Consequently, the WPC-based approach is restricted to services implemented in .NET. The WPC monitor runs on the same host as the service (see Figure 4) and continuously monitors its QoS attributes. The measured values are published into the VRESCo runtime the same way as described before. For the WPC-based approach the monitoring schedules are defined in configuration files as shown in Listing 1. It defines which service/operation should be monitored, together with the monitoring and availability checking interval (in ms). The first describes how often the counters are retrieved while the latter is required since availability should be checked more often than other QoS attributes to get meaningful results.

Listing 1: WPC Monitoring Configuration

```
<vresco.qosmonitoring monitoringinterval="60000"
                      availabilitycheckinterval="5000">
 <webservices>
  <webservice wsdl="http://localhost:8013/s?wsdl">
   <operations>
    <add name="TestOperation"/>
   </operations>
  </webservice>
 </webservices>
</vresco.qosmonitoring>
```

In general, the two approaches are independent. However, some attributes can only be measured by one of the approaches (e.g., latency and response time have to be measured from the client-side). Table 1 shows the QoS attributes currently measured in VRESCo and depicts which approach has been taken for which attribute. *Throughput* and *Calls Per Second* seem to refer to the same QoS attribute. However, the first represents the maximum number of requests that can be processed, while the latter indicates the number of invocations that really occurred.

QoS Attributes	Monitored by
Execution Time	QUATSCH & WPC
Response Time	QUATSCH
Latency	QUATSCH
Availability	QUATSCH & WPC
Throughput	QUATSCH
Calls Per Second	WPC
Calls Failed Per Second	WPC

Table 1: QoS Attributes

It can be seen that two attributes are measured by both approaches. For both *Availability* and *Execution Time*, the WPC-based approach is usually more accurate since it does not need to send probe requests, but represents the values of real invocations. However, we decided to monitor using both approaches since the measured values might be different. In Section 4, we briefly discuss why this combination is useful and give some concrete examples.

3.3 SLA Monitoring

QoS monitoring approaches, as introduced in the last section, represent an essential foundation for SLAs, which define the expected QoS between service providers and consumers. In this section, we describe the SLA monitoring approach in VRESCo, and how clients can react to SLA violations. This approach is based on the VRESCo event engine and is depicted in the top part of Figure 4. To give a brief overview, simple SLA obligations can be attached to services. This is done using the publishing service that also allows to temporary start and stop SLA monitoring. These obligations are then transformed to subscriptions specified in the Esper Processing Language (EPL) [2] since the VRESCo event engine is based on the open source engine Esper. Those listeners can be directly attached to the engine, which does the actual matching between subscriptions and events. Finally, when such matches occur the subscribers are notified about the corresponding SLA violation.

Frameworks such as WSLA [4] have been proposed for defining complex SLAs, but they are rarely used in practice. Therefore, we decided to provide a mechanism for defining simple SLA obligations representing guarantees on the QoS attributes of services, which are shown in Table 2.

First of all, obligations can be either attached to service operations or revisions (in VRESCo, services can have multiple revisions). Every obligation is valid only within a given period of time after which it expires. The property name represents the QoS attribute to monitor (e.g., response time), while logical operator and property value are used to define threshold values (e.g., $<$ 500 ms). Aggregation functions (e.g., sum, max, avg, median, etc.) can further be defined on multiple QoS events. Obligations also define the notification mechanism and the address used for violation notifications (e.g., E-Mail or WS-Eventing notifications). Finally, sliding window operators can be used to define the time period to consider for the QoS events (e.g., the last 10 events or events within the last 5 minutes).

To give concrete examples, a simple SLA obligation could define that the availability of revision 23 should be greater than 0.99. This obligation is transformed to the following EPL expression (please note that logical operators must be inverted since subscriptions represent violation conditions):

Property	Description
Id	Identifier of the obligation
RevisionId	Identifier of the service revision
OperationId	Identifier of the service operation
Start Date	Start date of the obligation
End date	End date of the obligation
PropertyName	QoS attribute to monitor
Aggregation	Aggregation function on property
LogicalOperator	Logical operator used for comparison
PropertyValue	Threshold value used for comparison
ReactionType	Notification mechanism to use
ReactionAddress	Address of the subscriber
WindowType	Type of the sliding window operator
WindowValue	Value of the sliding window operator

Table 2: SLA Obligations

```
select * from QoSRevisionEvent where Revision.Id=23
and Property='Availability' and DoubleValue<=0,99
```

A more complex SLA obligation could define that operation 47 of service revision 11 should have an average response time of less than 500 ms within the last 24 hours. Besides the sliding window operator (`win:time`) this SLA obligation uses univariate statistics on event streams (`stat:uni` and `average`) which are provided by Esper:

```
select * from QoSOperationEvent(Revision.Id=11 and
Operation.Id=47 and Property='ResponseTime')\\
.win:time(24 hours).stat:uni('DoubleValue')
where average>=500
```

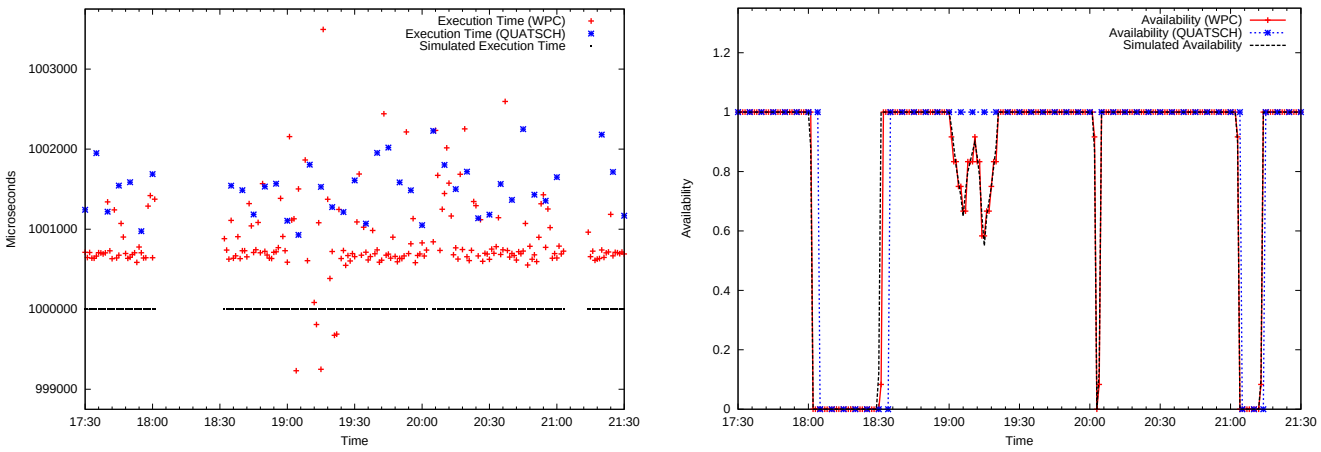
Once an SLA violation is detected, notifications are sent using E-Mail or Web service notifications to the specified address. The subscribers can then react accordingly, for instance by rebinding to functionally equal services [8]. In this regard, the SLA violation mechanism can also be used by service providers to monitor if services perform as intended. SLA violation notifications could then automatically trigger to start new instances of this service and publish them into VRESCo. Such scenarios and ways to define SLA penalty models are part of our future work.

4. EVALUATION

To evaluate our approach, we compare the accuracy of both monitoring approaches in terms of execution time and availability as two exemplary values that can be measured by both QUATSCH and WPC. Based on these findings, we discuss why a combination of both approaches is useful and highlight some of its advantages and disadvantages.

Our evaluation environment consists of a server hosting VRESCo and a set of C#/.NET dummy services that have a configurable execution time and a variable availability (3 downtimes of 30 min, 2 min and 10 min length, and simulated network problems with short interruptions between 19:00 and 19:20). Additionally, QUATSCH is hosted on a VMWare image running on a different server in the LAN.

Figure 5 depicts the results of our monitoring experiments where QUATSCH probes every 5 minutes whereas WPC measures every minute. We further use soapUI [16] to simulate clients. The different measurement intervals are based on the fact that QUATSCH sends real invocations to probe a service, while WPC has lower overhead because it queries performance counters provided by the operating system.



(a) Execution Time

(b) Availability

Figure 5: Accuracy of WPC and QUATSCH

Figure 5(a) depicts the measured execution time over 4 hours. The results show that both approaches are pretty accurate. The deviation from the simulated execution time (1 sec) is less than 2 ms for most measurements. The values for QUATSCH indicate that execution time can be indeed measured from the client-side. Additionally, it can be seen that WPC is more accurate because it represents the average execution time of real invocations. The gap between simulated execution time and average value measured by WPC is 0.88 ms which is partly caused by internal processing of the test services (e.g., threading, console output, etc.).

Figure 5(b) shows the simulated and measured availability of the test services. It can be observed that WPC detects downtimes faster than QUATSCH, which is due to the shorter monitoring interval. WPC further divides this interval into availability checking intervals (5 sec). Therefore, the availability of one monitoring interval can also be measured (e.g., at 18:31 and 20:02). In contrast to that, QUATSCH cannot do this fine-grained distinction (i.e., availability is either 0 or 1). As a result, QUATSCH does not recognize the short downtime at 20:02. The same is true for the timespan between 19:00 and 19:20 where WPC is quite accurate whereas QUATSCH does not detect this at all. It should be noted that the same behavior can be observed when the execution time in Figure 5(a) is varying.

Nonetheless, combining both approaches is still useful. Firstly, some QoS attributes can only be measured from the client-side (e.g., latency). Secondly, it is possible to distinguish between client- and server-side view on some QoS attributes (e.g., availability). For instance, if there is no network connection on the QUATSCH monitoring host, client-side availability decreases even if the service is running. However, this can be verified by the WPC approach. Thirdly, bogus server-side measurements can be detected by the client-side approach, by comparing measured QoS values over a longer period of time. Fourthly, another dimension to client-side monitoring could be added by integrating actually perceived QoS values on the client-side (in addition to the measured values of the QUATSCH probe requests). However, the combination of both approaches also has some drawbacks. For instance, clients must agree to install monitoring software which may not always be the case.

Finally, we have shown that the accuracy of the monitoring approaches makes them suitable for SLA monitoring as introduced in Section 3.3. As shown in our previous work [7], the throughput of the VRESCo event engine is high enough for the expected number of services (e.g., 2000 services with the same monitoring intervals as above). Furthermore, since SLA monitoring is based on events, it is easily possible to subscribe to SLA violations and react adaptively if needed.

5. RELATED WORK

Several different QoS models have been proposed in literature (e.g., [6, 12, 19]). However, most approaches do not discuss how QoS can be monitored. An overview of QoS monitoring approaches for Web services is presented by Thio et al. [17]. The authors discuss various techniques such as low-level sniffing or proxy-based solutions. The prototype system presented in their paper adopts an approach where the SOAP engine library is instrumented with logging statements to emit the necessary information for QoS measurement. A major drawback of this approach is the dependency on the modified SOAP library and the resulting maintenance and distribution of the modified library.

QoS monitoring has been an active research area for years which is not only focused on Web service technology. For instance, Garg et al. [3] present the WebMon system that aims at monitoring the performance of web transactions using a sensor-collector architecture. Similar to our work, their approach correlates client- and server-side response times which are measured by different components. In their work, the question is whether to instrument the web server or the web browser for doing the performance measurements.

There are many existing approaches for SLA monitoring and violation detection (e.g., [1,5,11,15] just to name a few). Skene et al. [15] introduce SLAng which is a general SLA language not only focused on Web services, but targeted to distributed systems and applications with reliable QoS characteristics. The language is modeled in UML while the syntax is defined using XML schema. The authors further define a model for all parties and services involved in such agreement. The actual constraints in the SLAs are then defined using the Object Constraint Language (OCL).

Raimondi et al. [11] describe an SLA monitoring system that translates timeliness constraints such as latency or availability of SLAs into timed automata, which are then used to verify execution traces of services. Their approach uses SLAng for defining SLAs and is realized as Axis handler.

Lodi et al. [5] describe a middleware that enables SLA-driven clustering of QoS-aware application servers. Instead of existing standards, they use XML for defining SLAs which was inspired by SLAng. The architecture consists of three components: The Configuration Service is responsible for managing the QoS-aware cluster, the Monitoring Service observes the application at runtime to detect violations of SLAs, and the Load Balancing Service intercepts client requests to balance them among different cluster nodes. If the cluster is mainly idle or close to breach the SLA (e.g., the response time converges to the upper bound), it is reconfigured (i.e., add/release nodes).

Chau et al. [1] present a similar approach for modeling and event-based monitoring of SLAs which is part of the eQoSystem project. The SLA model refines the WSLA specification: SLAs consist of multiple SLOs and use various metrics that indicate different measurement aspects of a process. Furthermore, action handlers can be defined to react when SLOs are violated. Similar to our work, the SLA monitoring approach is based on events. These events are assumed to be emitted by the business process and contain a snapshot of the current process state. In contrast to that, our QoS events focus on the service- and operation-level. Furthermore, we additionally address how QoS attributes of Web services can be monitored from client- and server-side.

6. CONCLUSION

Monitoring QoS attributes of Web services is an essential aspect to enforce SLAs established among business partners. In this paper, we have shown that a combination of client- and server-side QoS monitoring can be beneficial regarding the overall monitoring capabilities since both approaches have strengths and weaknesses. These monitoring capabilities combined with a powerful Web service runtime enable an event-based detection of SLA violations for Web services, while subscribers can react appropriately to such violations. For future work, we plan to automatically react to SLA violations, such as deploying new service instances on-the-fly or dynamically increase certain virtual machine capabilities (that are often used to host Web services). Furthermore, we envision to measure and publish the actual response times at the client-side, in addition to the QUATSCH measurements that rely on probe requests.

Acknowledgements

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). Additionally, we would like to thank Alexander Schindler for realizing the WPC-based monitoring approach.

7. REFERENCES

[1] T. Chau, V. Muthusamy, H.-A. Jacobsen, E. Litani, A. Chan, and P. Coulthard. Automating SLA Modeling. In *Proc. of the 2008 Conference of the Center for Advanced Studies on Collaborative Research (CASCON'08)*, 2008.

[2] Esper, 2009. `http://esper.codehaus.org/`.

[3] P. K. Garg, K. Eshghi, T. Gschwind, B. R. Haverkort, and K. Wolter. Enabling Network Caching of Dynamic Web Objects. In *Proc. of the 12th Int. Conference on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS'02)*, 2002.

[4] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[5] G. Lodi, F. Panzieri, D. Rossi, and E. Turrini. SLA-Driven Clustering of QoS-Aware Application Servers. *IEEE Transactions on Software Engineering*, 33(3):186–197, 2007.

[6] D. A. Menascé. QoS Issues in Web Services. *IEEE Internet Computing*, 6(6):72–75, 2002.

[7] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. Advanced Event Processing and Notifications in Service Runtime Environments. In *Proc. of the 2nd Int. Conference on Distributed Event-Based Systems (DEBS'08)*. ACM, 2008.

[8] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. End-to-End Support for QoS-Aware Service Selection, Invocation and Mediation in VRESCo. Technical report, Vienna University of Technology, 2009. `http://www.infosys.tuwien.ac.at/Staff/michlmayr/papers/TUV-1841-2009-03.pdf`.

[9] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11):38–45, 2007.

[10] C. Peiris, D. Mulder, A. Bahree, A. Chopra, S. Cicoria, and N. Pathak. *Pro WCF: Practical Microsoft SOA Implementation*. Apress, 2007.

[11] F. Raimondi, J. Skene, and W. Emmerich. Efficient online monitoring of web-service SLAs. In *Proc. of the 16th ACM SIGSOFT Int. Symposium on Foundations of Software Engineering (SIGSOFT'08/FSE-16)*, 2008.

[12] S. Ran. A Model for Web Services Discovery with QoS. *SIGecom Exchanges*, 4(1):1–10, 2003.

[13] F. Rosenberg. *QoS-Aware Composition of Adaptive Service-Oriented Systems*. PhD thesis, Vienna University of Technology, June 2009.

[14] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping Performance and Dependability Attributes of Web Services. In *Proc. of the IEEE Int. Conference on Web Services (ICWS'06)*, Sept. 2006.

[15] J. Skene, D. D. Lamanna, and W. Emmerich. Precise Service Level Agreements. In *Proc. of the 26th Int. Conference on Software Engineering (ICSE'04)*, 2004.

[16] soapUI, 2009. `http://www.soapui.org/`.

[17] N. Thio and S. Karunasekera. Automatic measurement of a QoS metric for Web service recommendation. In *Proc. of the Australian Software Engineering Conference (ASWEC'05)*, 2005.

[18] WCF Performance Counters, 2009. `http://msdn.microsoft.com/en-us/library/ms735098.aspx`.

[19] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, May 2004.