Monitoring Performance-Related QoS Properties in Service-Oriented Systems: A Pattern-Based Architectural Decision Model

Ernst Oberortner[1], Stefan Sobernig[2], Uwe Zdun[3], Schahram Dustdar[1]

[1]*Distributed Systems Group, Information Systems Institute*
Vienna University of Technology, Austria
{e.oberortner,dustdar}@infosys.tuwien.ac.at

[2]*Institute for Information Systems and New Media*
Vienna University of Economics and Business (WU Vienna), Austria
stefan.sobernig@wu.ac.at

[3]*Software Architecture*
Faculty of Computer Science
Vienna University, Austria
uwe.zdun@univie.ac.at

In service-oriented systems, service-level agreements (SLA) are specified as contracts between service providers and service consumers. SLAs stipulate — among other things — the provided services' performance. A service provider must fulfill the agreements made in SLAs, e.g., to avoid paying penalties. Service consumers must verify whether the service quality as purchased is delivered. We present an architectural design decision model (ADDM) covering the design of a QoS monitoring infrastructure. The ADDM collects design decisions about measuring, storing, and evaluating performance-related quality-of-service (QoS) agreements. The model includes various requirements and puts forth design solutions by referring to established architectural, remoting, and design patterns. We evaluate the ADDM to build the architecture of a QoS monitoring infrastructure in a case study. The model guides through the decision-making process of designing a QoS monitoring infrastructure.

1 Introduction

Service-oriented systems are utilized to perform intra- and inter-organizational activities mostly in an automated fashion. Each business activity is aligned with a distributed IT service [51]. Service-oriented systems must be extended to ensure compliance with negotiated service level agreements (SLA) [10, 47, 33]. An SLA is a contract that contains — among other things — agreements on performance-related for invoking the provider's services over a network. If the service provider cannot meet the agreements, serious consequences follow, e.g., financial consequences caused by non-fulfillment penalties payable to the service consumer or a third party [39]. Likewise, the consumers want to observe and validate that the service provider does not violate the guaranteed SLAs. Against the background of these requirements, creating and maintaining a monitoring infrastructure for the performance clauses stipulated by SLAs is inevitable.

The architectural design process of an infrastructure for quality-of-service (QoS) monitoring is a challenging and comprehensive task. To give some examples, performance monitoring can be realized at different network communication layers, at the client side, at the server side, and in intermediary components; or using any combination of these. As a consequence, many architectural design decisions are to be taken about measuring performance-related QoS properties, evaluating the measurements performed against negotiated SLAs, as well as storing the measurement and the evaluation data.

In this paper, we present an architectural design decision model (ADDM) for guiding the decision-making process when designing a QoS monitoring infrastructure. The model proposes pattern-based architectural solutions to detect and to prevent SLA violations of performance-related QoS properties, such as round-trip time, network latency, or processing time [30, 32, 36, 34, 50]. This model identifies relevant design decisions arising throughout the decision-making process and established design solutions described in pattern form. These solutions are, in turn, based upon established software patterns in their solution space. In our ADDM, we refer to design patterns [12], remoting patterns [49], and architectural patterns [3]. The documented design decisions, requirements, and solutions were harvested through a literature and systems review [52, 18, 37, 24, 35, 19, 5, 15, 2, 36], as well as through implementing research prototypes and through conducting several industrial case studies.

The ADDM is an architectural design guide to adopt solutions according to the requirements imposed on your QoS monitoring infrastructure. A guided walk through the decision-making process traces the architectural design decisions taken and records why a particular solution was chosen, facilitating later maintenance and extension.

It is possible to apply the patterns to integrate various other compliance concerns into service-oriented systems, such as security, licensing, or internal policies. However, in this work, we concentrate on applying patterns to monitor performance-related QoS properties. Dealing with prevented and detected SLA violations, such as violation management, is out of the scope of this paper. However, the model's solutions influence the architecture of a violation management system and helps facilitate its architectural design. Furthermore, we do not consider reporting SLA violations to internal or to external stakeholders, e.g., the finance department, to the service engineers, or to the service consumers [46].

We organized the paper as follows: In Section 2 we comment on the importance of QoS monitoring, by giving an example of an online book store. The features of a QoS monitoring infrastructure are explained in Section 3. Then, in Section 4.1 we introduce common requirements on monitoring infrastructures. Then, we present the ADDM in Section 4. In Section 4.3, the relationships between the architectural design decisions covered by the model are discussed. We evaluate the ADDM in Section 5 by reporting on a case study. This case study illustrates a walk through the decision-making process leading to certain design solutions. We conclude the paper and hint at future work in Section 6.

2 Motivating Example

Consider an online book store built upon two processes (see Figure 1): a long-running business process for handling online orders and a short-running technical process displaying product details on a Web page. In these scenarios three parties exist: service consumers, service providers, and third party providers. A *service provider* offers services with a specific functionality to its customers. In our example, the online book store offers services to order books online. A *service consumer* accesses the offered services to request the services' functionality. To order books online, a buyer places an order by accessing the online store's Web site, ordering the books in a desired quantity. *Third-party providers* offer services to support the functionality of the service provider's services. For example, to process the consumer's order, the online store has to reorder the requested books from a wholesaler in case the items are out of stock. The two roles of service consumer and service provider alternate between the parties involved. For example, the online store acts as a service provider towards its online buyers while the store itself consumes the wholesalers' services.

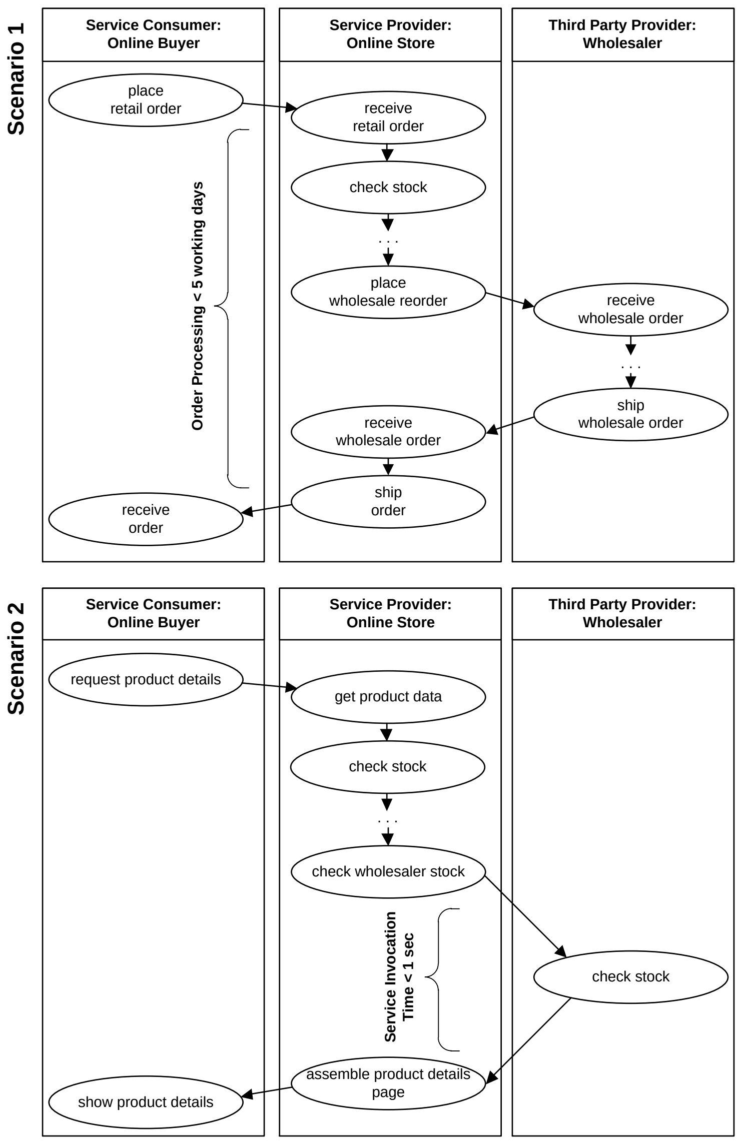


Figure 1: Two motivating scenarios

In Scenario 1, the online buyer orders a book at the online store. This scenario illustrates an SLA in a long-running process: Assume that there is a binding agreement between the online store (as the service provider) and its online buyers regarding the maximum duration of order processing. This order processing time amounts to five working days and represents the maximum time in which the online store commits itself to dispatch the orders. However, it does not include the delivery time taken by the postal services or other intermediate shipment used. In case of non-fulfillment, the customer receives the ordered product for free. Under such obligations, the online store must minimize the risk of non-fulfillment. In particular, the online store must take precautionary measures to prevent SLA violations due to a third party (i.e., the wholesaler) not delivering on time. This risk assessment requires some process monitoring capabilities. Equally, the client wants to verify the actual order processing time regularly during or once after the actual delivery, for instance, as part of a package tracking system provided by the online store.

In Scenario 2, an SLA for a short-running service invocation is illustrated. If the online store is out of stock, it must query the stock of the wholesaler. If such information should be displayed on the online store's Web site, it is important that service invocations into the wholesaler's system do not take too long, in order to avoid long delays in displaying the requested Web site to the customer; or to provide timely updates. Here, the SLA requires any service invocation to the wholesaler's system to take less than 1 second. If the online store stipulates such a QoS guarantee, they would want to monitor compliance with the guarantee themselves. If the wholesalers give such guarantees, they would show interest in monitoring the service invocations as well, to be able to document compliance and react swiftly in case of violations.

It is characteristic for QoS monitoring to face common and variable requirements. An exemplary commonality is measuring the maximum time span between two measuring points. A point of variation is, for example, storing measurement data either in a persistent storage for long-running transactions, or in memory for short-term observations.

Designing a monitoring infrastructure meeting the requirements is a particular challenge. This paper explains best practices for the various design options and supports making informed design decisions based on the requirements and the forces of each individual scenario.

3 Features of a QoS Monitoring Infrastructure

Infrastructures for QoS monitoring can be described by a set of features and certain feature configurations. A feature is a unit of functionality being of interest to the technical stakeholders of the QoS monitoring infrastructure. Figure 2 depicts our ADDM's covered features of a QoS monitoring infrastructure, using the Extended Eisenecker-Czarnecki Notation [9]. As illustrated, the three main features of our ADDM are: `Measuring`, `Evaluation`, and `Storage`.

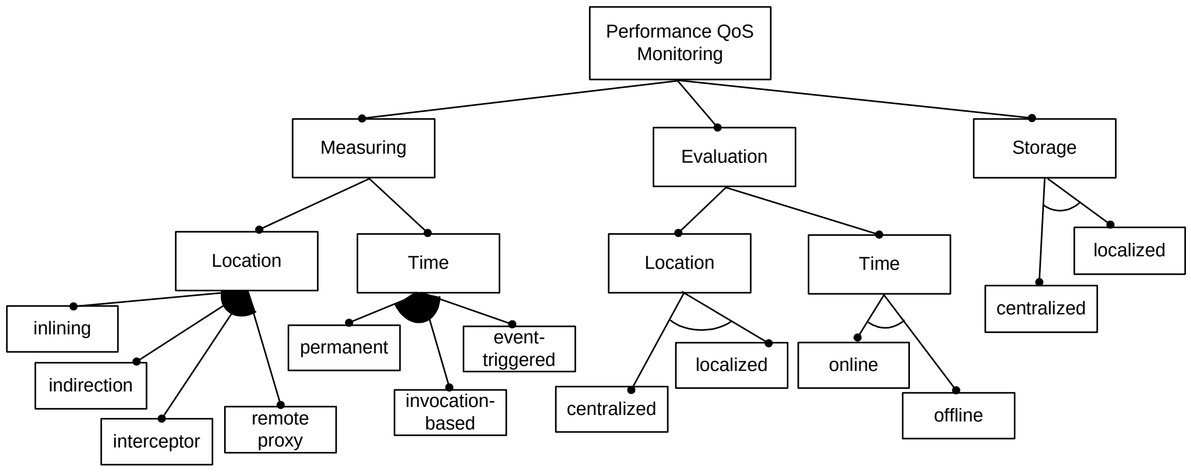


Figure 2: Features of a QoS monitoring infrastructure

Measuring

Multiple variants exist in a service-oriented system to measure performance-related QoS properties. Most importantly, different strategies of instrumenting techniques of the system components for runtime and execution monitoring at various spots are available. In our ADDM, we assume that system components can either be instrumented (1) at the service endpoints, such as the client and service applications, (2) at the level of the service middleware, such as the utilized middleware framework (e.g., Apache CXF) or a process engine (e.g., Apache ODE), or (3) at the execution platforms of both the service endpoints and the service middleware, such as language engines (e.g., a Java virtual machine).

As for instrumentation techniques, we identified the following: `inlining`, `indirection`, and/or `proxying`. While some of these techniques are entirely independent of the kind of instrumentation target, certain techniques are only applied to specific targets. By `inlining`, we mean to implement measuring points by introducing dedicated measuring code into the client or service applications directly. Alternatively, forms of `indirection` can be adopted, such as by utilizing variants of the WRAPPER pattern [12] at the service endpoints. For example, the client or service applications redirect the instruction calls to some sort of measuring wrapper that measures the performance-related QoS properties. At the level of the service middleware, variants of the INVOCATION INTERCEPTOR pattern [49] can be adopted, making it possible to predefine measuring points along the invocation path in order to measure performance-related QoS properties. Looking at the service interactions, measuring can also be achieved by `proxing`, i.e., deploying service-level proxying within the service consumers' or service providers' networks. In such a setting, a REMOTE PROXY service [49], responsible for measuring the QoS properties, trades service invocations on behalf of the actual service implementations.

As for the timing of measurement, measurements can either be piggybacked onto actual service invocations (`invocation-based`) and/or probe service invocations. Probing is based on creating mock-up service invocations to perform the measurements without interfering with actual invocations. Probes are either emitted periodically (`permanent`) at regular intervals following a specified probing strategy, or can be triggered by system-wide events (`event-triggered`), such as by user demands.

EVALUATION

When designing a QoS monitoring infrastructure, it must be decided on *when* (`Time`) and on *where* (`Location`) to evaluate performance-related QoS measurements . As for the timing, SLA performance evaluation can either happen during the service (and therefore SLA) performance (`online`) or after an SLA's duration of validity (`offline`). While offline evaluation satisfies the requirements emerging from SLA accounting and reporting, online evaluation enables scenarios of preventing SLA violations as part of the SLA management.

Another important variation in monitoring systems results from organizing the evaluation feature in a `centralized` or in a `localized` manner. A `centralized` evaluation is performed by a central evaluation component, responsible for all clients and services in a service-oriented system. For instance, a business-process execution engine may be extended to perform the role of the evaluation component. `Localized` evaluation shifts the responsibility of performing evaluations to each of the service endpoints, i.e., both service clients and service providers.

STORAGE

In a QoS monitoring infrastructure the performance-related QoS measurements must be stored. For example, in case of deciding in favor of a permanent measuring solution and an evaluation solution at the end of the SLA's validity, the measurements must be stored until the end of the SLA's validity. In our ADDM, we differentiate between storing the measurement data in a `localized` and/or `centralized` manner. Storing the measurements locally means that the QoS measurements are stored locally at each client and service. For example, in case of deciding to utilize a proxying measuring solution, the QoS measurements can be stored locally at the proxy, resulting in a `centralized` storing solution too.

4 The Architectural Design Decision Model (ADDM)

In this section, we describe our ADDM to guide the various involved stakeholders through the decision making process. First, we explain the covered requirements within our ADDM. Then, we present several architectural decision decisions, which requirements influence each of the design decision, and propose pattern-based solutions. Each solution addresses the influencing requirements differently. We also discuss each design decision solution with respect to the influencing requirements and the design solutions of related design decisions. We conclude our ADDM by introducing relations between the design decisions..

4.1 The ADDM's Requirements on a QoS Monitoring Infrastructure

In this section, we explain the criteria driving the decision-making process and the various requirements imposed on a QoS monitoring infrastructure. Both, criteria and requirements, influence the architectural design decisions and the selection of appropriate solutions. In our model, we differentiate between decision criteria, system-specific, and implementation-specific requirements. Criteria are characteristics that must be known a priori, before making any design decisions, such as whether freestanding services are provided; or whether the provided services invoke services of third-party providers. System-specific requirements concentrate on the general requirements for the QoS monitoring infrastructure and its architecture. At this level, technical details regarding the implementation of the QoS monitoring infrastructure are omitted. For example, system-specific requirements are scalability or reusability. Implementation-specific requirements focus on realizing the components forming the QoS monitoring infrastructure. For example, implementing QoS monitoring could require accessing to the clients' or to the services' implementations.

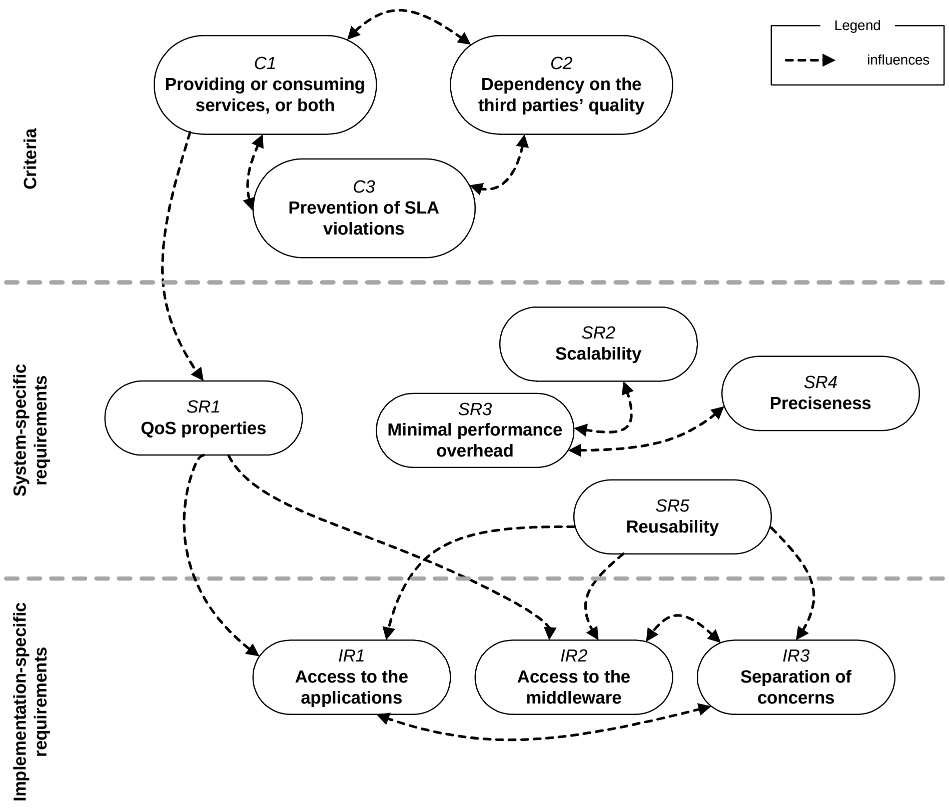

Legend
influences
Criteria
C1
Providing or consuming services, or both
C2
Dependency on the third parties' quality
C3
Prevention of SLA violations
System-specific requirements
SR1
QoS properties
SR2
Scalability
SR3
Minimal performance overhead
SR4
Preciseness
SR5
Reusability
Implementation-specific requirements
IR1
Access to the applications
IR2
Access to the middleware
IR3
Separation of concerns

Figure 3: Influences between the criteria and requirements

Figure 3 shows the requirements relating to each other. For example, a minimal performance overhead is required for realizing a scalable QoS monitoring infrastructure. Understanding the interrelatedness of requirements helps comprehend the relationships between the architectural design decisions.

Decision criteria

- C_1 – **Providing or consuming services, or both**
  This criterion considers whether the decision-taking party plans to provide or to consume services; or even both. While in a service-oriented system, a provider and consumer roles can be strictly separated, a service provider can also act as a consumer towards third-party services.

  The criterion is related to the decision whether it becomes necessary to either detect or prevent SLA violations (C_3). For a service provider, it is desirable to prevent SLA violations during the SLA's validity in order to avoid financial consequences and a diminished reputation. In contrast, a service consumer wants to detect potential SLA violations under a ruling SLA. Detecting and reporting violations is also a use case when the performance quality delivered by a service provider is directly dependent on third-party providers. With this, the criteria on providing or consuming services also is affected by the criterion C_2.

- C_2 – **Dependency on the third parties' quality**
  In service-oriented systems, service providers often invoke third-party services to accomplish their services' functionality. As a result and as illustrated in the motivating example (see Section 2), the services' quality so become dependent on the quality of some third parties' services. In order to avoid SLA violations, an appropriate QoS monitoring solution must be designed to take timed actions in case the performance of the third parties' services degrades.

  Third-party dependencies bear the risk of SLA inversions and generally limit a service providers capacity to commit to a high-performance SLA. Provided that there are no adequate monitoring facilities and the SLA details are not carefully crafted (e.g., by excluding features directly coupled to third-party performance from the SLA), a provider's SLA can only pass on whatever the respective third-party SLAs offer, degraded by the provider's non-fulfillment probability. Regarding the risk of SLA inversion, a monitoring infrastructure assists at detecting the non-fulfillment by third-party providers.

  If relying on third-party services, a service provider takes the role of a service consumer. With this, this criterion leads to C_1 and, as a consequence, to C_3.

- C_3 – **Prevention of SLA violations**
  Prior to entering the decision-making process, it must be decided if the QoS monitoring infrastructure should prevent or just detect SLA violations. This criterion results from criteria C_1 and C_2. For a service consumer it is maybe enough just to detect SLA violations. For example, in the motivating scenario, a detection of SLA violations for the service consumer is sufficient. But for the service provider only a detection of SLA violations is not satisfactory. For service providers it is desirable to prevent SLA violations in order to avoid financial consequences. This criterion results from criteria C_1. In contrast to detecting violations, preventing SLA violations has the benefit for the service-providing parties to effectively avoid any SLA violations. However, developing a preventive QoS monitoring solution is a complex design and development task. Note that, in our model, preventing SLA violations implies the capacity to detect them.

System-specific Requirements

System-specific requirements focus on the QoS monitoring infrastructure independent of its implementation. In our model, we consider the following system-specific requirements:

- SR_1 – **QoS properties**
  To measure the performance-related QoS properties, it is necessary to know which of the service performance indicators are negotiated in the underlying SLA. Typically, SLAs do not cover all measurable performance-related QoS properties. For example, for a service consumer the

service provider's message processing time (at the level of communication middleware) is not of interest. Rather, a service consumer is more concerned about the services' *round-trip time* or *time-to-response*.

Nevertheless, from the perspective of the service provider, it is necessary to measure the network-specific performance-related QoS properties as well in order to detect bottlenecks in long running service invocations.

After having decided whether to adopt a provider-side and/or consumer-side QoS monitoring solution (C_1), the selection of performance-related QoS properties to gather and to measure is next. Once the performance-related QoS properties and the measure instruments are decided upon, it becomes clear if and which kind of implementation-level access is required. Implementation-level access refers to either the provider- and/or client-side service implementations (IR_1) or even the provider- and client-side middleware implementations (IR_2); or even both.

- SR_2 – **Scalability**
  SLAs are contracts between one service provider and one service consumer. However, service providers can have several SLAs negotiated with multiple service consumers. Likewise, service consumers can hold various SLAs issued by different service providers. Hence, many SLAs must be monitored and tracked. Providing many SLA-aware services to many consumers implies that the consumers can invoke the service in parallel, requiring to monitor the performance-related QoS agreements of each service invocation. As a result, the QoS monitoring infrastructure should scale to a variable number of service clients and services provided. Scalability involves both up- and down-scaling. If the number of SLA-governed services and clients to monitor increases, the monitoring infrastructure must adapt and must guarantee availability. A decreasing number, however, should result in freeing system resources dedicated to monitoring tasks. The latter is particularly important when SLA monitoring is realized as a third-party service.

  The property of scalability, in particular up-scaling, is directly affected by the performance overhead incurred by monitoring invocations (SR_3). A highly scalable monitoring system implies a minimal performance overhead.

- SR_3 – **Minimal performance overhead**
  A further requirement is that the QoS monitoring solution does not introduce critical performance overhead into to distributed system. The monitoring of service execution introduces an INDIRECTION LAYER [3] into the distributed system because the implementations of the client and service applications must be instrumented to gather data related to executing remote invocations (e.g., by intercepting method invocations, message delivery, marshaling, etc. to gather execution timings) and related to network I/O (e.g., latency). It is not desirable that the execution performance of the overall system degrades due to monitoring the performance-related QoS properties. The requirement on minimal performance overhead is strong coupled with the scalability requirement (SR_2): the lower the performance overhead, the more scalable the QoS monitoring system.

  The overhead affects the scalability of a monitoring system (SR_2). The higher the overhead, the more unbiased are the performance-related QoS measurements (IR_4). This again results in imprecise evaluation results. High performance overhead can itself cause SLA violations. For example, in case a centralized evaluation component becomes overloaded it can influence the systems' performance, resulting that the processing of some clients' requests lasts longer than contracted.

- SR_4 – **Preciseness**
  The preciseness criterion relates to the rigor and validity of the performance-related QoS measurements and the evaluation results. Imprecise measurements cause imprecise evaluation results and, as a consequence, false positives and false negatives in detecting SLA violations. Monitor-

ing tasks such as measuring, evaluating, and storing should not distort the actual measurements. Most importantly, preciseness follows from minimizing the indirection overhead (SR_3).

- SR_5 – **Reusability**
  The QoS monitoring solution is required to be reusable in the heterogeneous software landscape which constitutes a service-oriented system. By reusability, we refer to the ability to deploy QoS monitoring for potentially diverse clients and services. This diversity results from the various implementation platforms and middleware frameworks used.

  As a consequence, there is a major tension between reusability and the need to instrument the service and the middleware implementations (IR_1, IR_2). A reusable monitoring system must respect a separation of concerns (IR_3), in particular by separating those monitoring concerns (storing, evaluation) from those which require platform- and implementation-specific adaptation (sensing).

Implementation-specific Requirements

Implementation-specific requirements focus on the implementation of the QoS monitoring infrastructure's components. We identified the following recurring implementation-specific requirements:

- IR_1 – **Access to the applications**
  Monitoring performance-related QoS properties of service invocations often requires access to the client's or service's implementation, in particular to apply certain measurement strategies. For example, to measure the *round-trip time* of a service invocation in the client, measuring points can be placed directly into the client's implementation. To measure the *processing time*, measuring points can be placed directly into the implementation of a service.

  This requirement conflicts with a separation of concerns (IR_3) and has the potential to decrease the reusability (SR_5) of a monitoring system.

- IR_2 – **Access to the middleware**
  For monitoring network-specific performance properties, such as the marshaling time, access to the middleware is required. The middleware must be adapted to measure the required network-specific performance-related QoS properties. Middleware frameworks offer different strategies for extending and intercepting the processing of invocations (e.g., INVOCATION INTERCEPTORS [49]).

  Before deciding on modifying the middleware framework, it must be clarified which kind of measurement probes are required (SR_1). Only then, it can be decided at which interception points certain processing steps are to be metered. Measuring the performance-related QoS properties by accessing the middleware improves over the separation of concerns (IR_3) because the measuring logic is decoupled from the services' or clients' implementations.

- IR_3 – **Separation of concerns**
  A monitoring solution must exhibit an overall state of separation of concerns. Multiple criteria contribute to this objective. First, a monitoring system must not modify the clients' or services' implementations in order to monitor the performance-related QoS agreements (IR_1). With this, there is a certain level of transparency because the monitoring solution is decoupled from the clients' and services' implementation. Monitoring solutions which realize their measurement sensors at the level of the middleware contribute to concern separation (IR_2). Separating concerns directly affects the system's reusability (SR_5).

4.2 The ADDM's Architectural Design Decisions

TODO: add some short section explanation here...

4.2.1 Architectural Design Decision: WHICH SLA PARTY WANTS TO MONITOR PERFORMANCE-RELATED QOS PROPERTIES?

One decision is to decide if a service provider or a service consumer wants to introduce a QoS monitoring solution. In our model, this design decision is influenced, as illustrated in Figure 4, by the requirement C_1. Our model proposes three decision alternatives.

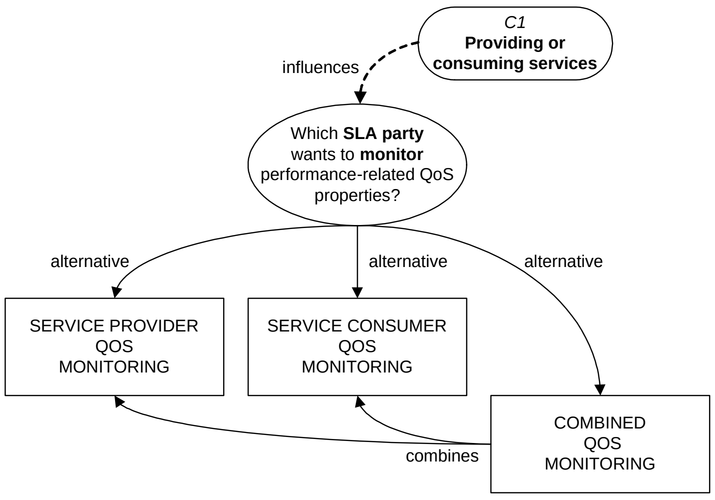

C1
Providing or consuming services
influences
Which SLA party wants to monitor performance-related QoS properties?
alternative
alternative
alternative
SERVICE PROVIDER QOS MONITORING
SERVICE CONSUMER QOS MONITORING
COMBINED QOS MONITORING
combines

Figure 4: WHICH SLA PARTY WANTS TO MONITOR PERFORMANCE-RELATED QOS PROPERTIES?

Solution: SERVICE PROVIDER QOS MONITORING

Integrate a QoS monitoring infrastructure into the service provider's network.

❖ ❖ ❖

In case of providing services, our model proposes a SERVICE PROVIDER QOS MONITORING solution. A SERVICE PROVIDER QOS MONITORING solution makes it possible to measure server-side performance-related QoS properties. Client-side performance-related QoS properties cannot be traced.

Known Uses:

- Windows Performance Counters (WPC) [28] of the the Windows Communication Foundation (WCF) [27] can serve for performing server-side QoS monitoring. WFC is part of the .NET framework [25].
- Rosenberg [35] has developed a SERVICE PROVIDER QOS MONITORING solution, called QUATSCH.

Solution: SERVICE CONSUMER QOS MONITORING

Integrate a QoS monitoring infrastructure into the service consumer's network.

❖ ❖ ❖

In case of consuming services, our model proposes a SERVICE CONSUMER QOS MONITORING solution. A SERVICE CONSUMER QOS MONITORING solution makes it possible to measure client-side performance-related QoS properties. Server-side performance-related QoS properties cannot be measured.

Known Uses:

- Mani and Nagarajan [22] explain the measuring of performance-related QoS properties within a client's implementation.

Solution: COMBINED QOS MONITORING

Integrate a common QoS monitoring infrastructure with the service consumer's and service provider's network. Measure the performance-related QoS properties in both networks and combine both measurements to evaluate the performance-related QoS agreements.

❖ ❖ ❖

In case, service provider and service consumer agree on a common QoS monitoring infrastructure, a COMBINED QOS MONITORING solution is proposed. As illustrated in Figure 4, a COMBINED QOS MONITORING solution incorporates the SERVICE PROVIDER QOS MONITORING and SERVICE CONSUMER QOS MONITORING solutions. This solution makes it possible to measure client- and server-side performance-related QoS properties.

A variant of COMBINED QOS MONITORING is realized by some process or workflow engines (e.g., Apache ODE, Riftsaw/JBOSS). A process engine implements service compositions according to process execution specifications. Composing services involves acting both as a service consumer and as a service provider (e.g., for asynchronous invocations with result callback). In addition, process engines contain native monitoring components, recording and providing lifecycle data of the process instances executed. Through dedicated monitoring APIs and depending on the granularity of lifecycle data stored (e.g., execution status), forms of COMBINED QOS MONITORING can be performed. Examples include aggregated process execution times. Note that measurements at the granularity level of individual process steps (i.e., service invocations) are usually not obtainable.

Known Uses:

- Michlmayr et al. [23] present a combined monitoring solution, requiring access to the clients' and services' implementation.
- Sahai et al. [37] introduce an SLA monitoring engine with two monitoring components. One in the service provider's network and one in the service consumer's network.
- Apache ODE [45] provides access to monitoring data through its *Process Instance Management Interface*. An event model describing process instance lifecycles and a corresponding event notification mechanism allow for creating aggregated monitoring solutions for the scope of process instances and processes.

Discussion

A SERVER-SIDE QOS MONITORING infrastructure makes it possible to measure server-side performance-related QoS properties, such as the processing time. In contrast, a CLIENT-SIDE QOS MONITORING solution allows for measuring the client-side performance-related QoS properties, such as the round-trip time. A COMBINED QOS MONITORING SOLUTION results in a more accurate evaluation of the QoS agreements.

4.2.2 Architectural Design Decision: WHERE SHOULD THE PERFORMANCE-RELATED QOS PROPERTIES BE MEASURED?

To gather performance data of service invocations, the clients' and the services' implementations must be instrument. This can be done at various layers of a distributed system, such as in the application layer, the network layer, or by extending the communication middleware.

In Figure 5 we illustrate how the requirements influence this design decision and which design solutions our model offers. The model's solutions are patterns that extend and utilize existing well-established design patterns, such as the WRAPPER pattern, the INTERCEPTOR pattern, the INVOCATION INTERCEPTOR pattern, or the PROXY pattern [12, 38, 49].

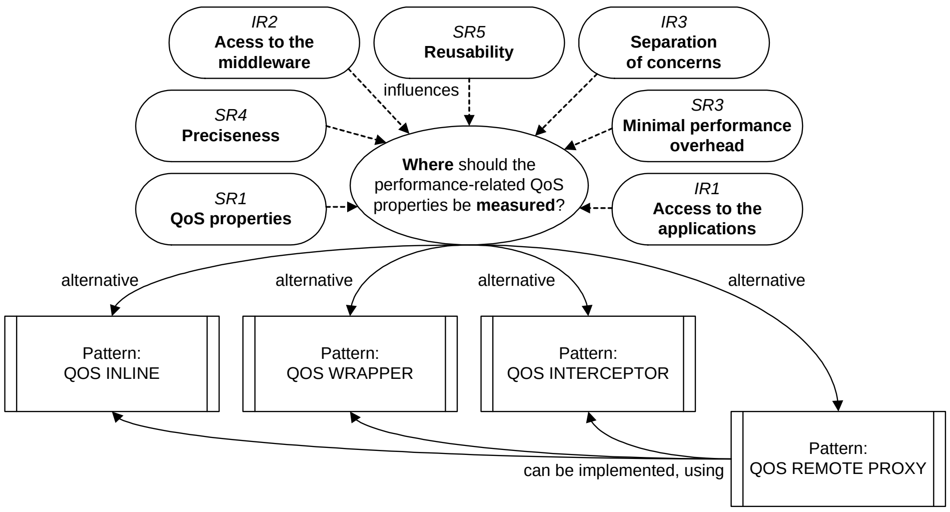

IR2 Acess to the middleware
SR5 Reusability
IR3 Separation of concerns
influences
SR4 Preciseness
SR3 Minimal performance overhead
Where should the performance-related QoS properties be measured?
SR1 QoS properties
IR1 Access to the applications
alternative
alternative
alternative
alternative
Pattern: QOS INLINE
Pattern: QOS WRAPPER
Pattern: QOS INTERCEPTOR
Pattern: QOS REMOTE PROXY
can be implemented, using

Figure 5: WHERE SHOULD THE PERFORMANCE-RELATED QOS PROPERTIES BE MEASURED?

Measuring the performance-related QoS properties should deliver precise measurements (SR_4) and have a minimal performance overhead (SR_3). The implementation of the measuring logic should provide separation of concerns (IR_3) in order to be reusable (SR_5). Dependent on which performance-related QoS properties should be measured (SR_1), access to the clients' or services' application (IR_1) or to the middleware is required (IR_2).

Our model provides four different solutions to measure the performance-related QoS properties. The solutions are patterns and we refer the reader to [30] for further information on the patterns.

Solution: QOS INLINE

Instrument the client's and the service's implementation with local measuring points and place them directly into their implementation.

Solution: QOS WRAPPER

Instrument the client's and service's implementations with local QOS WRAPPERS that are responsible for measuring the performance-related QoS properties. Let a client invoke a service using a client-side QOS WRAPPER. Extend a service with a server-side QOS WRAPPER that receives the client's requests.

Solution: QOS INTERCEPTOR

Hook QOS INTERCEPTORS into the middleware that intercept the message flow between the client and the service. Let the QOS INTERCEPTORS measure the performance-related QoS properties of service invocations.

Solution: QOS REMOTE PROXY

Implement and setup a QOS REMOTE PROXY in the service consumer's or service provider's network. In the service consumer's network, let each client invoke the services via the QOS REMOTE PROXY. In the service provider's network, make each service only accessible via a QOS REMOTE PROXY.

Discussion

The QOS INLINE pattern has a minimal performance overhead (SR_3), is scalable (SR_2) delivers precise results (SR_4), but, it requires access to the client's or service's applications (IR_1). Also, few performance-related QoS properties can be measured (SR_1), it is not reusable (SR_5), and hence, does not provide separation of concerns (IR_3).

The QOS WRAPPER pattern results in a minimal performance overhead (SR_3), is scalable (SR_2), reusable (SR_5), and does not affect the clients' or services' application (IR_3). The performance-related QoS measurements delivered by a QOS WRAPPER are slightly different to the measurements delivered by the QOS INLINE pattern.

Equivalent to the QOS WRAPPER pattern, the QOS INTERCEPTOR pattern has a minimal performance overhead (SR_3), is scalable (SR_2), reusable (SR_5), and provides separation of concerns (IR_3). A QOS INTERCEPTOR can be used to measure network-specific performance-related QoS properties, such as wrapping times or the network latency. Traditional middleware frameworks, such as .NET Remoting [26], Apache CXF [44], or Apache Axis APACHEAXIS, provide features to hook QOS INTERCEPTORS into service invocations dynamically (IR_2).

Because a QOS REMOTE PROXY is a centralized node in the network and one extra hop is required, the QoS measurements are different to the ones measured locally at each client or service (SR_4). The implementation of the QOS REMOTE PROXY can follow the QOS INLINE pattern, QOS WRAPPER pattern, or the QOS INTERCEPTOR pattern. However, the QOS REMOTE PROXY pattern is reusable (SR_5) and provides separation of concerns (IR_3) because it does not modify the clients' or services' application (IR_3). Using a QOS REMOTE PROXY for multiple clients or services can decrease the system's scalability (SR_2) and can increase the performance overhead (SR_3) because the QOS REMOTE PROXY can be a bottle-neck.

4.2.3 Architectural Design Decision: WHEN SHOULD THE PERFORMANCE-RELATED QOS PROPERTIES BE MEASURED?

A service's performance-related QoS properties are measured within the service invocation. This architectural design decision focuses on the time and frequency of the service invocations. In Figure 6 we illustrate the requirements' influences of this design decision.

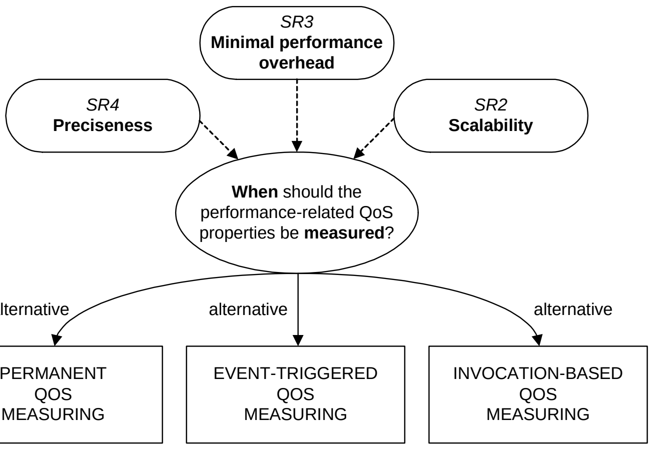

SR3
Minimal performance overhead
SR4
Preciseness
SR2
Scalability
When should the performance-related QoS properties be measured?
alternative
alternative
alternative
PERMANENT QOS MEASURING
EVENT-TRIGGERED QOS MEASURING
INVOCATION-BASED QOS MEASURING

Figure 6: WHEN SHOULD THE PERFORMANCE-RELATED QOS PROPERTIES BE MEASURED?

In our model we propose three different solutions for the architectural design decision about the measurement time.

Solution: PERMANENT QOS MEASURING

Send periodically, in pre-defined time intervals, probe requests to the service to measure the performance-related QoS properties permanently.

To get a permanent information about the performance-related QoS properties, the service consumer must adapt the client's measuring solution to send periodically probe requests to the services. During the probe requests, the performance-related QoS measurements can be measured. The service provider must invoke the services internally in periodic intervals to get a permanent information about the services' performance-related QoS properties. The service provider must develop a component that invokes the services and measure the performance-related QoS properties.

One possible solution is to develop a centralized component in the service consumers or service providers network that invokes the services periodically. For example, a QOS REMOTE PROXY can be extended to invoke the services periodically to measure the services' performance-related QoS properties permanently.

Invoking the services frequently, i.e., setting a short time interval, a precise information about the services' performance-related QoS properties can be gathered. A short time interval can produce performance overhead, resulting in a low scalability. Setting a long time interval, the preciseness can be diminished, but, the performance overhead is minimal and the scalability enhances.

Known Uses:

- The CISCO Response Time Reporter (RTR) Enhancement [7] measures the response times and availability of network resources using probe requests. The RTR can be utilized as a PERMANENT QOS MEASURING solution.
- Summers et al. [42] introduce the SLAM tool (SLA monitor), a framework to monitor SLA compliance using a probe sender, i.e., a PERMANENT QOS MONITORING solution.
- Rosenberg's QUATSCH toolkit [35] measures performance-related QoS properties during probe service requests.
- Traverse's SLA monitor, developed by Zyrion Inc. [54], allows to measure performance-related QoS properties in a specified time interval, i.e., the period of the SLA's validity.

Solution: EVENT-TRIGGERED QOS MEASURING

Take probes to measure the performance-related QoS properties in case selected events occur in the system.

If events occur rarely, the performance overhead is minimal and the scalability increases. But the preciseness of the performance-related QoS properties decreases. Occur events frequently, the measurements of the performance-related QoS properties becomes more precise. But the scalability decreases because of the performance overhead.

Known Uses:

- The CISCO Response Time Reporter (RTR) Enhancement [7] can be configured to measure the response times and availability when *a user-configured threshold is exceeded, a connection is lost and reestablished, or when a timeout occurs*.
- The QUATSCH toolkit [35] can be instantiated to send probe request in case of occurring events, such user requests.

Solution: INVOCATION-BASED QOS MEASURING

Measure the performance-related QoS properties every time a service invocation happens.

All performance-related QoS properties are measured within service invocations. In contrast, the previous solutions (PERMANENT QOS MEASURING and EVENT-TRIGGERED QOS MEASURING) measure the performance-related QoS properties in probe requests and are not triggered by invocations but by external events or timers. The INVOCATION-BASED QOS MEASURING focuses on measuring the performance-related QoS properties at the time a service invocation occurs.

In case service invocations happen often, the performance overhead can increase, resulting in a lower scalability. But the measurements of the performance-related QoS properties becomes more precise. Seldom service invocations result in a minimal performance overhead, dependent on the selected solutions of the other design decisions'. The scalability enhances, but, the preciseness can be diminished of seldom service invocations.

For selecting the INVOCATION-BASED QOS MEASURING solution, the service provider must not develop a component that sends probe requests to the services in order to measure the services' performance-related QoS properties.

Known Uses:

- Mani and Nagarajan [22] illustrate proxy-based measuring of performance-related QoS of web services per invocation.
- Afek et al. [1] implemented a framework for QoS-aware remote object invocations using Java RMI over an ATM network. QoS measuring works per invocation following the QOS WRAPPER pattern.
- The QoS CORBA Component Model (QOSCCM) [31] use the QOS INTERCEPTOR pattern to realize a non-intrusive INVOCATION-BASED QOS MEASURING solution for measuring performance-related QoS properties.

Discussion

For a service consumer, choosing a PERMANENT QOS MEASURING or EVENT-TRIGGERED QOS MEASURING solutions means to extend the clients to send probe requests to the service. Setting an appropriate time interval to send the probe requests is of particular interest, otherwise the service provider can identify the probe requests as potential denial of service (DoS) attacks. For a service provider an INVOCATION-BASED QOS MEASURING solution is a convenient solution because no additional components must be developed in order to measure the services' performance-related QoS properties. Furthermore, sending probe requests to the services can impact the system's performance overhead (SR_3), decrease the scalability (SR_2), and can influence the preciseness of the measurements (SR_4).

4.2.4 Architectural Design Decision: WHERE SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE EVALUATED?

This design decision concentrates on the location where the evaluation of the performance-related QoS measurements should take place. In Figure 7 we show how requirements influence this design decision.

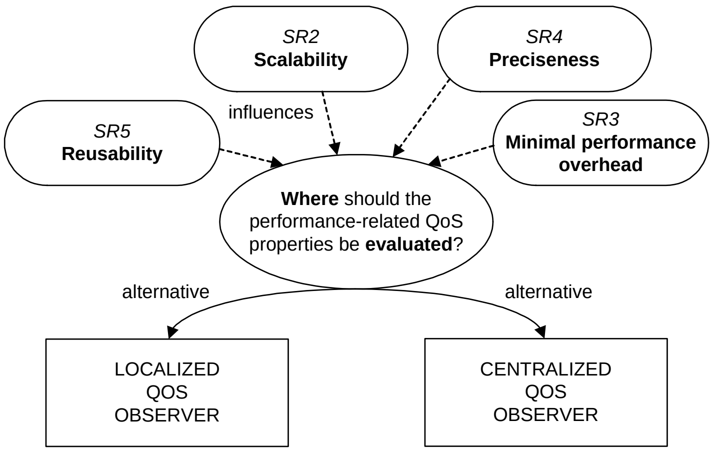


Figure 7: WHERE SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE EVALUATED?

The evaluation solution should be reusable, making it possible that multiple performance-related QoS measurements can be evaluated regarding the negotiated SLAs. It is required that the SLA evaluation solution has a minimal performance overhead and that the system's scalability does not decrease. The evaluation solution should provide precise evaluation results and should not influence the measurements of other performance-related QoS properties.

In our model, we propose two architectural design solutions for evaluating the performance-related QoS measurements. Both are strategies of the QOS OBSERVER pattern [49].

Solution: LOCALIZED QOS OBSERVER

Evaluate the performance-related QoS measurements locally at each client or service.

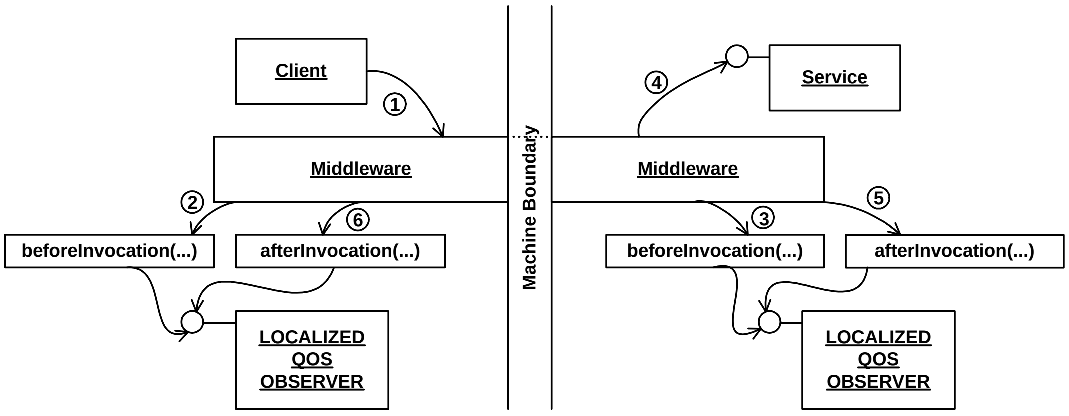


Figure 8: LOCALIZED QOS OBSERVER

In Figure 8 we sketch the LOCALIZED QOS OBSERVER solution that is based on the QOS OBSERVER pattern [49]. The LOCALIZED QOS OBSERVER resides within each client or service and is responsible for evaluating the performance-related measurements. The measuring solution passes the QoS measurements to the LOCALIZED QOS OBSERVER, making an immediate evaluation possible.

❖ ❖ ❖

A LOCALIZED QOS OBSERVER is a scalable solution with a minimal performance overhead. Dependent on the implementation, a LOCALIZED QOS OBSERVER is reusable. In case of implementing a WRAPPER, the LOCALIZED QOS OBSERVER is reusable. In contrast, implementing the LOCALIZED QOS OBSERVER within the services' or clients' implementation is not reusable.

A LOCALIZED QOS OBSERVER can influence other performance-related QoS measurements in case the measurements are evaluated or stored immediately. For example, a server-side LOCALIZED QOS OBSERVER that evaluates immediately a service's processing time can influence the measured round-trip time at the client-side.

Known Uses:

- CISCO's QoS Device Manager (QDM) [6] is installed locally at routers or switches for observe the network's QoS properties.
- Ecklund et al. [11] use a LOCALIZED QOS OBSERVER by introducing reusable QoS managers within a middleware.
- The Zyrion Traverse SLA Manager [54] monitors SLAs in a decentralized fashion, i.e., a LOCALIZED QOS OBSERVER.

Solution: CENTRALIZED QOS OBSERVER

Submit the performance-related QoS measurements to a CENTRALIZED QOS OBSERVER that evaluates the measurements regarding the negotiated SLAs.

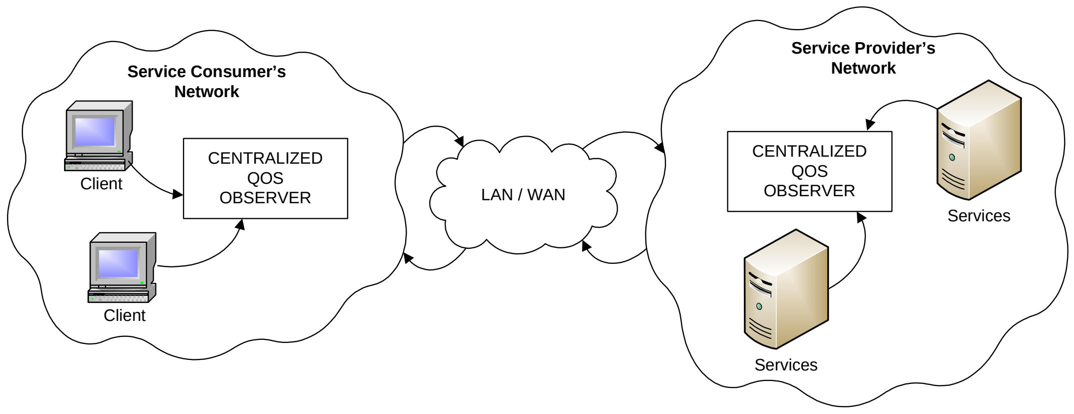


Figure 9: CENTRALIZED QOS OBSERVER

In Figure 9 we illustrate the architecture of a CENTRALIZED QOS OBSERVER. The clients send the performance-related QoS measurements to a CENTRALIZED QOS OBSERVER that is placed within the service consumer's network. At the server side, the services submit the performance-related QoS measurements to a server-side CENTRALIZED QOS OBSERVER to evaluate the measurements.

❖ ❖ ❖

A CENTRALIZED QOS OBSERVER is a reusable solution to evaluate the performance-related QoS measurements. The clients' and the services' measuring solution must be configured or implemented to submit the measurements to the CENTRALIZED QOS OBSERVER.

Sending the performance-related QoS measurements to the CENTRALIZED QOS OBSERVER over the network can impact the systems' performance. In a high scalable system, a CENTRALIZED QOS OBSERVER can be a bottle-neck of the QoS monitoring infrastructure. At the server-side, the performance-related QoS properties are measured and submitted to the CENTRALIZED QOS OBSERVER within the service provider's network. The sending of the measurements over the network can influence the client-side measurements, resulting in imprecise performance-related QoS measurements and evaluation results.

Known Uses:

- Sahai et al. [37] introduce an SLA violation engine, a CENTRALIZED QOS OBSERVER.
- Badidi et al. [4] present a CENTRALIZED QOS OBSERVER within the WS-QoSM architecture.
- The CISCO IOS IP SLAs [8] provide a CENTRALIZED QOS OBSERVER to evaluate the performance-related QoS measurements.
- Li et al. [20] use a CENTRALIZED QOS OBSERVER for evaluation.
- Michlmayer et al. [23] designed an event-driven CENTRALIZED QOS OBSERVER for detecting SLA violations.
- The EVEREST+ framework [21] includes a CENTRALIZED QOS OBSERVER to predict SLA violations.

4.2.5 Discussion

To compare the two presented solutions, a LOCALIZED QOS OBSERVER evaluates the performance-related QoS measurements at each client or service locally, whereas a CENTRALIZED QOS OBSERVER evaluates the measurements for all clients or services uniformly. You have to inform both observers about the performance-related QoS agreements. Because SLAs change or can get re-negotiated, dynamic interfaces are desirable. Having to evaluate multiple performance-related QoS measurements, a CENTRALIZED QOS OBSERVER can influence the system's performance (SR_3) and decrease the scalability (SR_2). A CENTRALIZED QOS OBSERVER is a reusable solution (SR_4) for all clients or services within the network. Implementing a LOCALIZED QOS OBSERVER solution following the ADAPTER or WRAPPER pattern [12] results in a reusable (SR_4) solution.

4.2.6 Architectural Design Decision: WHEN SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE EVALUATED?

The next architectural decision focuses on the evaluation of the performance-related QoS measurements. In our terminology, evaluating means to check the performance-related QoS measurements regarding the SLAs in order to detect or prevent SLA violations.

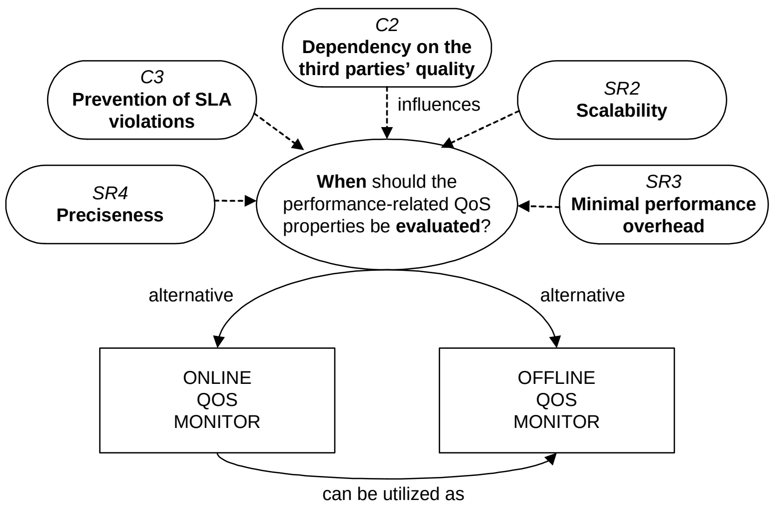

C2
Dependency on the third parties' quality
C3
Prevention of SLA violations
SR2
Scalability
influences
SR4
Preciseness
When should the performance-related QoS properties be evaluated?
SR3
Minimal performance overhead
alternative
alternative
ONLINE QOS MONITOR
OFFLINE QOS MONITOR
can be utilized as

Figure 10: WHEN SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE EVALUATED?

In Figure 10 we illustrate the design decisions requirements and the proposed solutions. The evaluation solution should deliver precise results in order to detect or prevent SLA violations. In case the services' performance depends on some third parties' services, the evaluation solution should alert performance drops well-timed. But, the evaluation solution should have minimal performance overhead in a high scalable system. An ONLINE QOS MONITOR can be used as an OFFLINE QOS MONITOR.

In the following, we present our model's solutions for this architectural design decision. The presented solutions are derivations of the QOS OBSERVER pattern [49].We explain the solutions' forces and consequences regarding the requirements.

Solution: ONLINE QOS MONITOR

Evaluate the performance-related QoS measurements with regard to the negotiated SLAs during the SLA's validity.

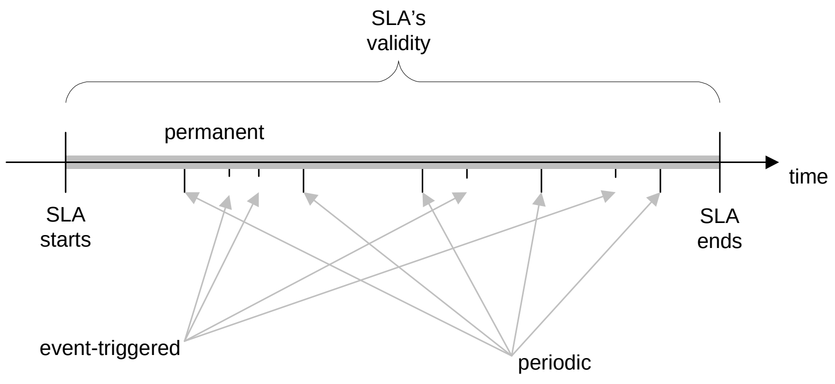


Figure 11: ONLINE QOS MONITOR

Typically SLAs are negotiated over a certain period of time. An ONLINE QOS MONITOR evaluates the performance-related QoS measurements within the SLAs' validity. In Figure 11 we sketch the the ONLINE QOS MONITOR solution for a better understanding.

❖ ❖ ❖

A **permanent** ONLINE QOS MONITOR evaluates the performance-related QoS properties after they have been measured immediately. Because of the permanent knowledge about the current compliance state, performance losses can be detected, making it possible to detect SLA violations before they occurred. But, the permanent evaluation can impact the systems performance.

An **event-triggered** ONLINE QOS MONITOR evaluates the performance-related QoS measurements in case certain events occur, such as user requests or system events. Dependent on the events' frequency, SLA violations can be avoided to take appropriate actions. In case the events' frequency is low, it is possible to violate SLAs. Between the occurrence of two events, the performance-related QoS measurements have to be stored somewhere. After the occurrence of an event, the stored measurements must be involved of the evaluation. Also dependent on the events' frequency, an event-triggered ONLINE QOS MONITOR can imply a performance overhead.

A **periodic** ONLINE QOS MONITOR evaluates the measured performance-related QoS properties in pre-defined time intervals. Dependent on the intervals' length, SLA violations can occur or can be prevented. A periodic ONLINE QOS MONITOR can impact the system's performance if the time interval is set to low. Within the time interval, the performance-related QoS measurements must be stored and after the elapsed time period included into the evaluation.

Known Uses:

- Ta and Mao [43] use an ONLINE QOS MONITOR solution to monitor SLAs.
- Zyrion's Traverse [54] is an ONLINE QOS MONITOR using proactive reporting to avoid SLA violations.
- Sahai et al. [37] describe an event-triggered ONLINE QOS MONITOR to evaluate the SLA when certain events happen.
- Crossflow [14, 13] combines an OFFLINE QOS MONITOR and an ONLINE QOS MONITOR. In the ONLINE QOS MONITOR, a QoS estimation component provides predictions of behavior of the currently running workflow instances. These estimates are based on performance models given as continuous time Markov models and produced by the OFFLINE QOS MONITOR. The ONLINE QOS MONITOR can compare the predictions and the real execution times of running workflows.

Solution: OFFLINE QOS MONITOR

Store the performance-related QoS measurements during the SLA's validity. Evaluate the stored measurements at the end of the SLA's validity.

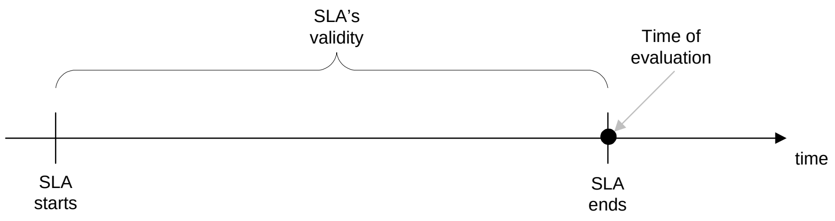


Figure 12: OFFLINE QOS MONITOR

In Figure 12 we illustrate on a time line when the OFFLINE QOS MONITOR evaluates the performance-related QoS measurements. As shown, the evaluation takes place after the SLA's validity.

❖ ❖ ❖

The OFFLINE QOS MONITOR has a minimal performance overhead because the performance-related QoS measurements just must be stored after they have been measured. After the SLA's validity the stored measurements are evaluated and possible SLA violations are detected. Because of a minimal performance overhead the scalability increases. An OFFLINE QOS MONITOR can deliver precise results in case the performance-related QoS properties were measured precisely.

As a consequence, the OFFLINE QOS MONITOR is not an adequate solution to prevent SLA violations. It is also difficult the detect any performance drops in case the services' quality depends on third-party services. Hence, an OFFLINE QOS MONITOR is not advisable for service providers. But, it is a convenient solution for a service consumer to detect SLA violations after the SLA's validity.

In case of deciding for an OFFLINE QOS MONITOR the architectural design decision WHERE SHOULD THE PERFORMANCE-RELATED QOS PROPERTIES? (see Section 4.2.7) must be answered because the performance-related QoS measurements must be stored during the SLA's validity.

Known Uses:

- The ProM framework, introduced by Dongen et al. [48], uses process mining techniques, making it possible to detect a processes' SLA violations offline.
- Jurca et al. [17] introduce an offline client-side QoS monitor to periodically report feedback to a trusted center in order to establish a reputation mechanism for client-side QoS monitoring.
- Crossflow [14, 13] combines OFFLINE QOS MONITOR and ONLINE QOS MONITOR. In the OFFLINE QOS MONITOR, past workflow executions are collected in a log. A continuous-time Markov chain is created using the log data. It is used to calculate the QoS values such as the time of workflow executions. These values are then used for calculating predictions and comparing them to the data of actually running workflows in the ONLINE QOS MONITOR.

Discussion

The ONLINE QOS MONITOR is a convenient solution to prevent SLA violations, and, hence, desirable for service providers. An ONLINE QOS MONITOR can also act as an OFFLINE QOS MONITOR. For example, at the end of an SLA's validity, an event can be triggered that the ONLINE QOS MONITOR evaluates the performance-related QoS measurements. An ONLINE QOS MONITOR can produce more overhead because it evaluates the performance-related QoS measurements permanently during the SLA's validity, whereas an OFFLINE QOS MONITOR evaluates the measurements once at the end of the SLA's validity. For a service consumer, an OFFLINE QOS MONITOR is often a convenient solution in order to detect SLA violations after the SLA's validity. In case of deciding in favor of an OFFLINE QOS MONITOR, architectural design decisions about storing the performance-related QoS measurements during the SLA's validity must be faced.

4.2.7 Architectural Design Decision: WHERE SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE STORED?

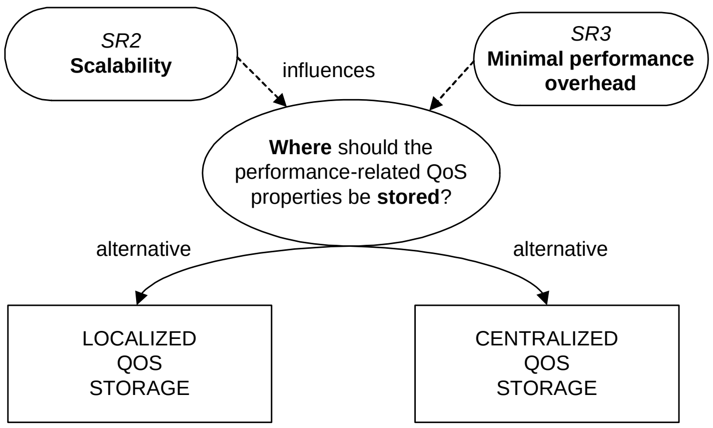


Figure 13: WHERE SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE STORED?

This architectural design decision focuses on where to store the performance-related QoS measurements and the evaluation results. Storing the performance-related QoS measurements is necessary in case the evaluation is performed at some later stage, such as at the end of an SLA's validity.

In Figure 13 we present how the requirements influence this architectural design decision. Storing the performance-related QoS measurements should have minimal performance overhead (SR_3) and should not influence the system's scalability (SR_2). In our model, we propose the two following solutions for storing.

Solution: LOCALIZED QOS STORAGE

Store the performance-related QoS measurements and/or evaluation results locally at each client or service.

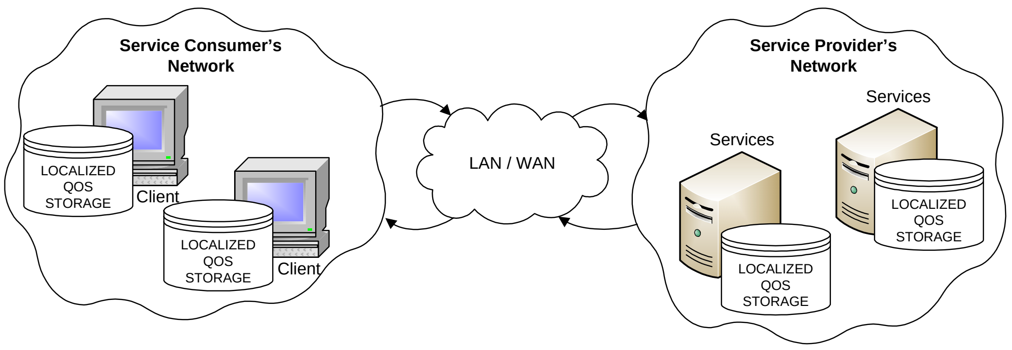


Figure 14: LOCALIZED QOS STORAGE

In Figure 14 we illustrate the architecture of a LOCALIZED QOS STORAGE. A LOCALIZED QOS STORAGE stores the performance-related QoS measurements locally at the client or the service, such as in a log file or in a local database.

❖ ❖ ❖

The solution does not impact the performance, because the performance-related QoS measurements must not be transmitted over the network to be stored. In addition, the scalability of a LOCALIZED QOS STORAGE increases.

In case the SLAs include multiple clients or services, the evaluation of the stored measurements can become more time consuming and complex. A CENTRALIZED QOS OBSERVER has to collect from each client or service the measurements for evaluation. In case of choosing a LOCALIZED QOS OBSERVER, the LOCALIZED QOS STORAGE is a fast and simple storing solution.

Known Uses:

- Zhu et al. [53] introduce the Localized Adaptive Data Collection and Aggregation Approach (LADCA), i.e., a LOCALIZED QOS STORAGE solution.
- The ServiceStore, introduced by Jin et al. [16], uses a LOCALIZED QOS STORAGE for QoS-aware service compositions.
- Shirazi et al. [40] designed a QoS-aware middleware. The middleware consists of a *Profiler* that stores performance-related QoS characteristics locally.
- The PISA server [29] to monitor workflow-based processes utilizes a LOCALIZED QOS STORAGE.

Solution: CENTRALIZED QOS STORAGE

Transmit the performance-related QoS measurements and/or evaluation results from each client or service to a centralized component that stores the data in a CENTRALIZED QOS STORAGE.

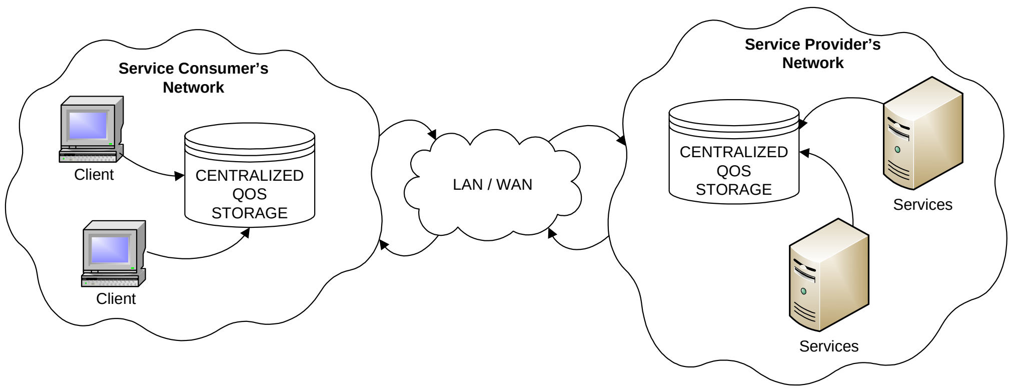


Figure 15: CENTRALIZED QOS STORAGE

In Figure 15 we illustrate the CENTRALIZED QOS STORAGE solution. A CENTRALIZED QOS STORAGE stores the performance-related QoS measurements in a centralized storage, such as in a database. In this case, the clients and services have to submit the performance-related QoS properties over the network in order to store them in the CENTRALIZED QOS STORAGE.

❖ ❖ ❖

In case the SLAs include multiple clients or services, a CENTRALIZED QOS OBSERVER can evaluate the stored measurements easily because they must not be collected from each client or service. If choosing a LOCALIZED QOS OBSERVER, then the client or service have to query and fetch the stored measurements over the network in order to evaluate them. A CENTRALIZED QOS OBSERVER is reusable for all clients or services within the network.

Sending the measurements over the network to the CENTRALIZED QOS STORAGE can result in a performance overhead. Also, the scalability can decrease.

Known Uses:

- Sahai et al. [37] build a high performance database that stores the QoS measurements.
- Li et al. [20] store the performance-related QoS measurements following the CENTRALIZED QOS STORAGE solution.
- Rosenberg et al. [36] use a CENTRALIZED QOS STORAGE to store evaluation results.

Discussion

Storing the performance-related QoS measurements following the LOCALIZED QOS STORAGE has the benefit that the measurements must not be transmitted over the network to a CENTRALIZED QOS STORAGE. In case of deciding in favor of a CENTRALIZED QOS OBSERVER to evaluate the measurements, the CENTRALIZED QOS OBSERVER can use a polling mechanism to collect the performance-related QoS measurements from each client or service. This can result in a longer evaluation process. Also, the clients and services must be equipped with an interface to gather to measurements from the LOCALIZED QOS STORAGE. It is also possible that the clients or services push the performance-related QoS measurements periodically to the CENTRALIZED QOS OBSERVER.

4.3 Relations between the Architectural Design Decisions

The ADDM provides pattern-based solutions for architectural design decisions, having forces and consequences against various requirements. Architectural design decisions are inter-dependent because several requirements influence multiple architectural design decisions. In this section, we discuss the relations between the architectural design decisions, stemming from common influencing requirements.

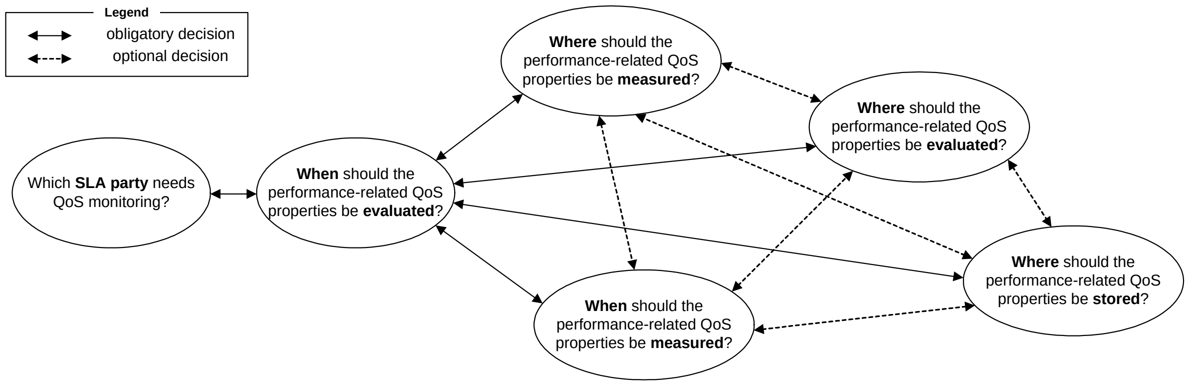

Legend
obligatory decision
optional decision
Where should the performance-related QoS properties be measured?
Where should the performance-related QoS properties be evaluated?
Which SLA party needs QoS monitoring?
When should the performance-related QoS properties be evaluated?
Where should the performance-related QoS properties be stored?
When should the performance-related QoS properties be measured?

Figure 16: Relations between the architectural design decisions

In Figure 16, we illustrate how the architectural design decisions depend on each other. Continuous lines between two design decisions mean that both design decisions must be taken, whereas dotted lines depict optional links between design decisions. After having clarified the requirements set, the model can be navigated for appropriate solutions.

Let us assume that the design decision WHICH SLA PARTY REQUIRES QOS MONITORING? is taken. Depending on the defined requirements (C_1), our model proposes a SERVICE PROVIDER QOS MONITORING, a SERVICE CONSUMER QOS MONITORING, or a COMBINED QOS MONITORING solution. The model documents which performance-related QoS properties can be measured, stored, and evaluated.

By following the model, one can keep track of defined requirements and the solutions adopted. For example, after defining for an architectural design decision that scalability (SR_3) is required, our model can propose for every subsequent design decisions solution candidates. Defining the requirements step-by-step for each design decision renders it possible to propose solutions for multiple subsequent design decisions. For example, if a scalable evaluation solution with a minimal performance overhead is required at any taken design decision, our model also proposes scalable solutions with a minimal performance overhead for the subsequent decision.

A decision-making process by eliciting the requirements stepwise, facilitates an interactive design process. Using the model allows for bookkeeping which design decisions which requirements were imposed. This improves the traceability throughout the decision-making process, recording the decision order. With this, the designers can replay the decision-making process, e.g., in order to adjust for changes in requirements.

5 Evaluation — A Case Study

In this section we present an industrial case study, dealing with multimedia web services that must comply to performance-related QoS agreements. We explain the case study, its services, and the services' performance-related QoS properties. Then we list the case study's requirements and exemplify the decision-making processes using the ADDM.

5.1 A Case Study Overview

The case study deals with advanced multimedia services offered by mobile virtual network operators (MVNO). Such services combine value-added application capabilities with the Internet and Next Generation Mobile Telecommunication Network capabilities. All these capabilities are integrated by the MVNO services to provide controls for calls and sessions, messaging features, location-aware features, multimedia content streaming, and parental monitoring. An MVNO serves as an intermediary between customers and the audio/video (a/v) streaming providers. It processes media search requests and streams a/v content according to customer-provided preferences, making it possible for the customers to watch, for example, live soccer matches with a selected audio commentary language.

For a better understanding of the case study, we give an example in Figure 17. The MVNO is the service provider and provides services with on-demand audio and video streaming content to its customer. First, a service customer must login into the system to order to access the services that offer the MVNO's audio and video streaming features. After a successful authentication and authorization, service customers can search desired video streams in a favored language by invoking the MVNO's search web service. Then, the MVNO invokes the web services of its a/v providers to fulfill the service customer's request. The MVNO receives the responses from the a/v providers, assembles them, and returns a list of possible streaming endpoints to the customer. The customer starts the multimedia streaming by selecting one endpoint.

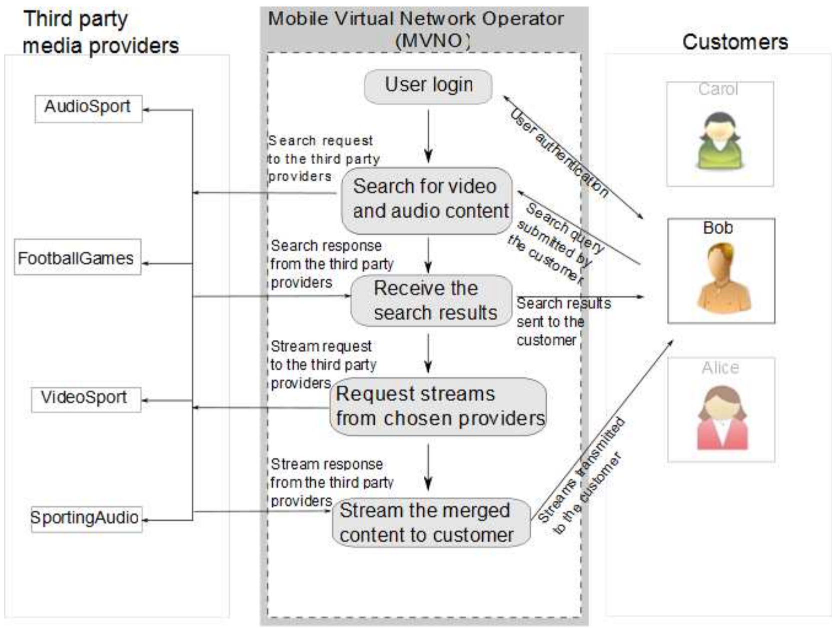

Third party media providers
Mobile Virtual Network Operator (MVNO)
Customers
AudioSport
FootballGames
VideoSport
SportingAudio
User login
Search request to the third party providers
Search for video and audio content
Search response from the third party providers
Receive the search results
Stream request to the third party providers
Request streams from chosen providers
Stream response from the third party providers
Stream the merged content to customer
User authentication
Search query submitted by the customer
Search results sent to the customer
Streams transmitted to the customer
Carol
Bob
Alice

Figure 17: An example scenario of the MVNO case study

5.2 The Case Study's Performance-related QoS Properties

The terms and conditions of the offered services are regulated by appropriate SLAs signed between the MVNO and the customer as well as between the audio and video providers and the MVNO enterprise. In the case study, the SLAs contain various agreements on the services' performance. It is crucial for the MVNO to check and avoid any potential violations with regard to the services offered to the service consumers as well as to detect any performance drops of the providers' services.

Performance-related QoS Properties	**Description**
Up-Time	A service's up-time is the probability that the service is operative and answering the service consumers' queries. In the case that an exception occurs during the services' processing, the service is considered as down. In the literature, a service's up-time is often referred as the service's availability.
Processing Time	The processing time is the elapsed time for processing the service consumers' queries. It does not take into account of processing the incoming requests and outgoing response in the underlying middleware.

Table 1: The case study's QoS compliance concerns

In Table 1, we list the case study's performance-related QoS properties. For MVNOs, not only proper quality level of connections, but also the quality of all the other services provided by MVNO must be assured, such as the streaming service. It is a non-trivial task as the MVNO services' quality usually depends on the third-party services' quality. The key properties in the case study are the up-time and the processing time. Up-time represents the degree of availability of every service to the consumers, i.e., running and answering the consumers' requests.Processing time constraints stipulate that services must process the consumer requests in a negotiated period of time.

5.3 The Case Study's Requirements

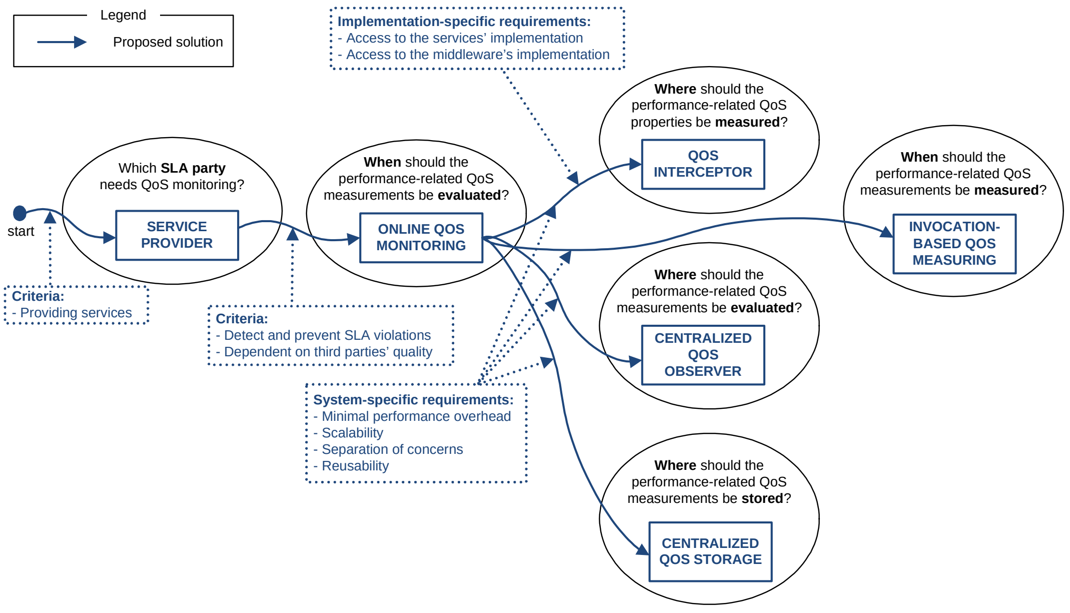
The MVNO provides services to service consumers (C_1) to stream multimedia content. The service consumers and the MVNO negotiate SLAs regarding the multimedia services' performance-related QoS properties. The MVNO as a service provider wants to introduce a QoS monitoring infrastructure, where the services' performance quality depends on the performance quality of the third parties' services (C_2). As a service provider, the MVNO should prevent SLA violations (C_3) in order to avoid financial consequences and a diminished reputation.

Currently the MVNO offers three services to its customers and it is highly possible that the MVNO will increase the number of offered services. It is foreseeable that more and more new consumers will access the services in the near future, making it inevitable to design a scalable (SR_2) architecture with a minimal performance overhead (SR_3). Furthermore, reusability (SR_5) is advisable.

Because the MVNO provides services to the service consumers, access to the services' implementation was provided (IR_1). The selection of a web service framework was a decision left to us. We haven chosen in favor of a framework that enables separation of concerns (IR_3) to monitor the performance-related QoS properties without modifying the services' implementation (IR_2).

5.4 The Case Study's Solutions

In the case study, we have used the ADDM in order to design a QoS monitoring infrastructure that fulfills the afore-mentioned requirements. We illustrate the proposed solutions in Figure 18.



Legend
Proposed solution
Implementation-specific requirements:
- Access to the services' implementation
- Access to the middleware's implementation
start
Which SLA party needs QoS monitoring?
SERVICE PROVIDER
Criteria:
- Providing services
When should the performance-related QoS measurements be evaluated?
ONLINE QOS MONITORING
Criteria:
- Detect and prevent SLA violations
- Dependent on third parties' quality
System-specific requirements:
- Minimal performance overhead
- Scalability
- Separation of concerns
- Reusability
Where should the performance-related QoS properties be measured?
QOS INTERCEPTOR
When should the performance-related QoS measurements be measured?
INVOCATION-BASED QOS MEASURING
Where should the performance-related QoS measurements be evaluated?
CENTRALIZED QOS OBSERVER
Where should the performance-related QoS measurements be stored?
CENTRALIZED QOS STORAGE

Figure 18: Proposed solutions of our ADDM

Using our ADDM, we were able to propose a SERVICE PROVIDER QOS MONITORING solution because the MVNO provides services to its consumers. To prevent SLA violations, the performance-related QoS properties must be evaluated during the SLAs' validity. Our model proposed to develop an ONLINE QOS MONITORING solution to detect any performance drops of the third parties' services, making it possible to take counter-measures against SLA violations.

To provide the necessary instrumentation, the QOS INTERCEPTOR pattern and an INVOCATION-BASED QOS MEASURING solution can be applied to meet the requirements IR_2 and IR_3. We adopted the Apache CXF web service framework [44] because it provides convenient solutions to intercept the messages to measure the services' performance-related QoS properties without modifying the services' implementation (IR_1).

Using our ADDM, we have proposed to use a CENTRALIZED QOS OBSERVER solution to evaluate the performance-related QoS measurements. The QOS INTERCEPTORS must be instrumented to submit the measurements to the CENTRALIZED QOS OBSERVER over the network. As a trade-off, the performance overhead can increase in case the number of services and clients increase. To store the QoS measurements, our model helped decide in favor of a CENTRALIZED QOS OBSERVER that is deployed on the same node as the CENTRALIZED QOS STORAGE, resulting in a convenient ONLINE QOS MONITORING solution.

6 Conclusion and Future Work

The main contribution of this paper is an architectural design decision model (ADDM) for designing the architecture of a QoS monitoring infrastructure for service-based systems. The model covers architectural design decisions about the measuring performance-related QoS properties, their evaluation against negotiated SLAs, and the storing of the measurements and evaluation results. Our model proposes architectural solutions by organizing existing architectural, remoting, and design patterns. We describe the forces and the consequences of each proposed solution of an architectural design decision against criteria, system-specific, and implementation-specific requirements.

To evaluate the ADDM, we employed it in an industrial case study. The case study deals with advanced QoS-aware multimedia services, making it possible to watch movies or live-streams in a favored language. We present the case study's proposed solutions for its QoS monitoring infrastructure in order to prevent SLA violations, incurring a minimal performance overhead and being reusable and scalable. In the case study, we used the ADDM in order to design the architecture of the case study's QoS monitoring infrastructure from scratch. As future work, we want to examine the dynamic behavior of the ADDM, i.e., to analyze an existing QoS monitoring infrastructure against some requirements by using the ADDM. In such cases, we can research questions about how the architecture must be changed regarding some requirements?

In this work, we limited ourselves to designing an internal QoS monitoring infrastructure that resides within the service provider's or service consumer's network. As potential future work we think about external QoS monitoring, governed by a third party outside the service consumer's and service provider's network. In this case, the service provider and the service consumer have to transmit the SLAs and the performance-related QoS measurements to the third party. An external QoS monitoring infrastructure raises questions about trust and security. For example, questions arise regarding the vulnerability to forge the QoS measurements or the evaluation results.

Feature-complete and production-grade QoS monitoring systems offer a reporting feature for presenting QoS evaluation results to the stakeholders, for instance, to the finance department for billing, to the engineers for diagnosing and planning, or directly to the service consumers [46]. As future work we plan to cover issues pertaining to QoS reporting. One possible solution to report the QoS monitoring results is a web-based dashboard, as described in [41]. We will review selected technical aspects, especially the choice of representation (e.g., processable report formats, UI-based reporting, notification schemes) and coverage (i.e., the range of reported details and the level of granularity).

Acknowledgment

We thank our shepherd Eduardo Fernandez for his constructive and supporting help during the shepherding process. Ed helped improve the quality of the patterns and the paper itself.

This work was supported by the European Union FP7 projects COMPAS, grant no. 215175, and INDENICA, grant no. 257483.

References

[1] Y. Afek, M. Merritt, and G. Stupp. Remote Object Oriented Programming with Quality of Service or Java's RMI over ATM, 1996.

[2] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A survey of QoS architectures. *Multimedia Systems*, 6:138–151, May 1998.

[3] P. Avgeriou and U. Zdun. Architectural Patterns Revisited – A Pattern Language. In *Proceedings of 10th European Conference on Pattern Languages of Programs (EuroPlop 2005)*, pages 1 – 39, Irsee, Germany, July 2005.

[4] E. Badidi, L. Esmahi, M. A. Serhani, and M. Elkoutbi. WS-QoSM: A Broker-based Architecture for Web Services QoS Management. In *Innovations in Information Technology, 2006*, pages 1–5, 2006.

[5] P. Bianco, G. A. Lewis, and P. Merson. Service Level Agreements in Service-Oriented Architecture Environments. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2001.

[6] CISCO. QoS Device Manager (QDM). `http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/qdm/index.htm`.

[7] CISCO. Response Time Reporter (RTR) Enhancements. `http://www.cisco.com/en/US/docs/ios/12_0t/12_0t3/feature/guide/RTRenh.html`.

[8] Cisco. Cisco IOS IP Service Level Agreements (SLAs), 2011. `http://www.cisco.com/go/ipsla` *(last accessed: February 2011).*

[9] K. Czarnecki and U. W. Eisenecker. *Generative Programming — Methods, Tools, and Applications*. Addison-Wesley Longman Publishing Co., Inc., 6th edition, 2000.

[10] F. Daniel, F. Casati, V. D'Andrea, E. Mulo, U. Zdun, S. Dustdar, S. Strauch, D. Schumm, F. Leymann, S. Sebahi, F. d. Marchi, and M.-S. Hacid. Business Compliance Governance in Service-Oriented Architectures. In *Proceedings of the 2009 International Conference on Advanced Information Networking and Applications*, pages 113–120, Washington, DC, USA, 2009. IEEE Computer Society.

[11] D. J. Ecklund, V. Goebel, T. Plagemann, E. F. Ecklund, Jr., C. Griwodz, J. O. Aagedal, K. Lund, and A.-J. Berre. QoS Management Middleware: A Separable, Reusable Solution. In *Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems*, IDMS '01, pages 124–137, London, UK, 2001. Springer-Verlag.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[13] P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. Crossflow: Cross-organizational workflow management in dynamic virtual enterprises, 2000.

[14] P. Grefen and Y. Hoffner. Crossflow: Cross-organizational workflow support for virtual organizations. In *Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises, RIDE-VE 1999, Sydney, Australia*, pages 90–91. IEEE, 1999.

[15] R. Hauck and H. Reiser. Monitoring Quality of Service across Organizational Boundaries. In *Proceedings of the Third International IFIP/GI Working Conference on Trends in Distributed Systems: Towards a Universal Service Market*, pages 124–137, London, UK, 2000. Springer-Verlag.

[16] J. Jin, Y. Zhang, Y. Cao, X. Pu, and J. Li. ServiceStore: A Peer-to-Peer Framework for QoS-Aware Service Composition. In *Proceedings of the 2010 IFIP international conference on Network and parallel computing*, NPC'10, pages 190–199, Berlin, Heidelberg, 2010. Springer-Verlag.

[17] R. Jurca, B. Faltings, and W. Binder. Reliable qos monitoring based on client feedback. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 1003–1012, New York, NY, USA, 2007. ACM.

[18] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11:57–81, 2003. 10.1023/A:1022445108617.

[19] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. Monitoring, Prediction and Prevention of SLA Violations in Composite Services. In *ICWS*, pages 369–376. IEEE Computer Society, 2010.

[20] Z. Li, Y. Jin, and J. Han. A Runtime Monitoring and Validation Framework for Web Service Interactions. In *Proceedings of the Australian Software Engineering Conference*, pages 70–79, Washington, DC, USA, 2006. IEEE Computer Society.

[21] D. Lorenzoli and G. Spanoudakis. EVEREST+: Run-time SLA violations prediction. In *Proceedings of the 5th International Workshop on Middleware for Service Oriented Computing*, MW4SOC '10, pages 13–18, New York, NY, USA, 2010. ACM.

[22] A. Mani and A. Nagarajan. Understanding quality of service for Web services – Improving the performance of your Web services, 2002. `http://www.ibm.com/developerworks/library/ws-quality.html`, *last accessed: February 2011.*

[23] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. Comprehensive QoS monitoring of Web services and event-based SLA violation detection. In *MWSOC '09: Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing*, pages 1–6, New York, NY, USA, 2009. ACM.

[24] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo. *IEEE Transactions on Services Computing*, 3:193–205, 2010.

[25] Microsoft. .NET. `http://www.microsoft.com/net/`.

[26] Microsoft. .NET Remoting. `http://msdn.microsoft.com/en-us/library/kwdt6w2k(v=vs.71).aspx`.

[27] Microsoft. Windows Communication Framework. `http://msdn.microsoft.com/en-us/netframework/aa663324.aspx`.

[28] Microsoft. Windows Performance Counters. `http://msdn.microsoft.com/en-us/library/ms735098.aspx`.

[29] M. Z. Muehlen and M. Rosemann. Workflow-based Process Monitoring and Controlling – Technical and Organizational Issues. In *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS-33*, pages 1–10. IEEE Computer Society Press, 2000.

[30] E. Oberortner, U. Zdun, and S. Dustdar. Patterns for Measuring Performance-Related QoS Properties in Distributed Systems. In *Pattern Languages of Programming Conference (PLoP)*, 2010.

[31] Object Management Group (OMG. Quality Of Service For CCM (QOSCCM), 2008.

[32] L. O'Brien, P. Merson, and L. Bass. Quality Attributes for Service-Oriented Architectures. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, page 3, Washington, DC, USA, 2007. IEEE Computer Society.

[33] M. Papazoglou. Compliance Requirements for Business-process driven SOAs. In A. Mazzeo, R. Bellini, and G. Motta, editors, *E-Government Ict Professionalism and Competences Service Science*, volume 280 of *IFIP International Federation for Information Processing*, pages 183–194. Springer Boston, 2008.

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

[35] F. Rosenberg. *QoS-Aware Composition of Adaptive Service-Oriented Systems*. PhD thesis, Vienna University of Technology, 2010.

[36] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping Performance and Dependability Attributes of Web Services. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, pages 205–212, Washington, DC, USA, 2006. IEEE Computer Society.

[37] A. Sahai, V. Machiraju, M. Sayal, L. J. Jin, and F. Casati. Automated SLA Monitoring for Web Services. In *IEEE/IFIP DSOM*, pages 28–41. Springer-Verlag, 2002.

[38] D. C. Schmidt, H. Rohnert, M. Stal, and D. Schultz. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[39] F. Schulz. Towards Measuring the Degree of Fulfillment of Service Level Agreements. In *Information and Computing (ICIC), 2010 Third International Conference on*, volume 3, pages 273–276, June 2010.

[40] B. Shirazi, M. Kumar, and B. Y. Sung. QoS Middleware Support for Pervasive Computing Applications. In *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9 - Volume 9*, HICSS '04, Washington, DC, USA, 2004. IEEE Computer Society.

[41] P. Silveira, C. Rodríguez, F. Casati, F. Daniel, V. D'Andrea, C. Worledge, and Z. Taheri. On the Design of Compliance Governance Dashboards for Effective Compliance and Audit Management. In *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'09)*, 2009.

[42] J. Sommers, P. Barford, N. Duffield, and A. Ron. A Framework for Multi-objective SLA Compliance Monitoring. In *In Proceedings of IEEE INFOCOM (minisymposium*, 2007.

[43] X. Ta and G. Mao. Online End-to-End Quality of Service Monitoring for Service Level Agreement Management. *Int. J. Commun. Syst.*, 21:383–404, April 2008.

[44] The Apache Software Foundation. Apache CXF. `http://cxf.apache.org/`.

[45] The Apache Software Foundation. Apache ODE. `http://ode.apache.org/`.

[46] The Open Group. SLA Management Handbook – Volume 4: Enterprise Perspective, 2004.

[47] H. Tran, T. Holmes, E. Oberortner, E. Mulo, A. B. Cavalcante, J. Serafinski, M. Tluczek, A. Birukou, F. Daniel, P. Silveira, U. Zdun, and S. Dustdar. An End-to-End Framework for Business Compliance in Process-Driven SOAs. In *Proceedings of SYNASC*, 2010.

[48] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. In *Lecture Notes in Computer Science: Applications and Theory of Petri Nets 2005: 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005. / Gianfranco Ciardo, Philippe Darondeau (Eds.)*, volume 3536, pages 444–454. Springer Verlag, jun 2005.

[49] M. Voelter, M. Kircher, and U. Zdun. *Remoting Patterns – Foundations of Enterprise, Internet, and Realtime Distributed Object Middleware.* Wiley & Sons, October 2004.

[50] W. D. Yu, R. B. Radhakrishna, S. Pingali, and V. Kolluri. Modeling the Measurements of QoS Requirements in Web Service Systems. *Simulation*, 83(1):75–91, 2007.

[51] U. Zdun and S. Dustdar. Model-Driven and Pattern-Based Integration of Process-Driven SOA Models. In F. Leymann, W. Reisig, S. R. Thatte, and W. M. P. van der Aalst, editors, *The Role of Business Processes in Service Oriented Architectures*, volume 06291 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[52] L. Zeng, C. Lingenfelder, H. Lei, and H. Chang. Event-Driven Quality of Service Prediction. In *Proceedings of the 6th International Conference on Service-Oriented Computing*, ICSOC '08, pages 147–161, Berlin, Heidelberg, 2008. Springer-Verlag.

[53] J. Zhu, S. Papavassiliou, and J. Yang. Adaptive Localized QoS-Constrained Data Aggregation and Processing in Distributed Sensor Networks. *IEEE Trans. Parallel Distrib. Syst.*, 17:923–933, September 2006.

[54] Zyrion, Inc. Traverse Service Level Management – Monitoring SLAs for Services and Infrastructure. `http://www.zyrion.com/company/whitepapers/Traverse_SLM.pdf`.

A Architectural Design Decisions, Requirements, and Solutions

A.1 WHICH SLA PARTY NEEDS QOS MONITORING?

Requirements	C_1 – Providing or consuming services, or both
Solution	SERVICE PROVIDER QOS MONITORING
Description:	**Integrate a QoS monitoring infrastructure into the service provider's network.**
Forces:	measuring of server-side performance-related QoS properties is possible
Consequences:	client-side performance-related QoS properties cannot be measured
Solution	SERVICE CONSUMER QOS MONITORING
Description:	**Integrate a QoS monitoring infrastructure into the service consumer's network.**
Forces:	measuring of client-side performance-related QoS properties is possible
Consequences:	server-side performance-related QoS properties cannot be measured
Solution	COMBINED QOS MONITORING
Description	**Integrate a common QoS monitoring infrastructure in the service consumers and service provider's network. Measure the performance-related QoS properties in both networks and combine both measurements to evaluate the performance-related SLAs.**
Forces:	measuring of client- and server-side performance-related QoS properties is possible
Consequences:	service provider and service consumer have to agree on a common QoS monitoring infrastructure

Table 2: WHICH SLA PARTY NEEDS QOS MONITORING?

A.2 WHERE SHOULD THE PERFORMANCE-RELATED QOS PROPERTIES BE MEASURED?

Requirements	SR_1 – QoS properties SR_3 – Minimal performance overhead SR_4 – Preciseness SR_5 – Reusability IR_1 – Access to the applications IR_2 – Access to the middleware IR_3 – Separation of concerns
Solution	Pattern: QOS INLINE
Description:	**Instrument the client's and the service's implementation with local measuring points and place them directly into their implementation.**
Forces:	- a minimal performance overhead can be detected - precise QoS measurements of round-trip and processing times - no access to the middleware is required - does not influence other QoS measurements
Consequences:	- network-specific performance-related QoS properties cannot be measured - access to the application is required - no separation of concerns is provided - not reusable
Solution	Pattern: QOS WRAPPER
Description:	**Instrument the client's and service's implementations with local QOS WRAPPERS that are responsible for measuring the performance-related QoS properties. Let a client invoke a service using a client-side QOS WRAPPER. Extend a service with a server-side QOS WRAPPER that receives the client's requests.**
Forces:	- no access to the clients' or services' application is required - no access to the middleware is required - a minimal performance overhead - separation of concern - reusable - precise QoS measurements
Consequences:	- cannot measure network-specific QoS properties
Solution	Pattern: QOS INTERCEPTOR
Description:	**Hook QOS INTERCEPTORS into the middleware that intercept the message flow between the client and the service. Let the QOS INTERCEPTORS measure the performance-related QoS properties of service invocations.**
Forces:	- no access to the application required - minimal performance overhead - separation of concerns - reusable
Consequences:	- access to middleware is required - can influences other QoS measurements, leading to imprecise QoS measurements
Solution	Pattern: QOS REMOTE PROXY
Description:	**Implement and setup a QOS REMOTE PROXY in the service consumer's and/or service provider's network. In the service consumer's network, let each client invoke the services via the QOS REMOTE PROXY. In the service provider's network, make each service only accessible via a QOS REMOTE PROXY.**
Forces:	- no access to the application required - separation of concerns - reusable - no access to the clients' and/or services' middleware necessary
Consequences:	- the performance overhead can increase - can influence other measurements, hence - can lead to imprecise QoS measurements

Table 3: HOW TO MEASURE PERFORMANCE-RELATED QOS PROPERTIES?

A.3 WHEN SHOULD THE PERFORMANCE-RELATED QOS PROPERTIES BE MEASURED?

Requirements	SR_2 – Scalability SR_3 – Minimal performance overhead SR_4 – Preciseness
Solution	PERMANENT QOS MEASURING
Description:	**Send periodically, in pre-defined time intervals, probe requests to the service to measure the performance-related QoS properties permanently.**
Forces:	- a high scalability in case of a long time interval - a performance overhead increases in case of a short time interval - precise measuring results if setting a short time interval
Consequences:	- a low scalability in case of a short time interval - a minimal performance overhead if setting a long time interval - imprecise measuring results if setting a long time interval
Solution	Pattern: EVENT-TRIGGERED QOS MEASURING
Description:	**Send probe requests to the service to measure the performance-related QoS properties in case certain events occur in the system.**
Forces:	- the scalability increases in case of rare event occurrence - a minimal performance overhead if events occur rarely - precise measuring results if events occur frequently
Consequences:	- low scalability if events occur frequently - a performance overhead in case of high event frequency - imprecise measuring results in occur rarely
Solution	INVOCATION-BASED QOS MEASURING
Description:	**Measure the performance-related QoS properties every time a service invocation happens.**
Forces:	- highly scalable if services are invoked rarely - a minimal performance overhead in case of rare service invocations - precise measurements if services are invoked often
Consequences:	- the scalability decreases in case of frequent service invocations - the higher the service invocation frequency, the higher the performance overhead - imprecise measuring results if services are invoked rarely

Table 4: WHEN SHOULD THE PERFORMANCE-RELATED QOS PROPERTIES BE MEASURED?

A.4 WHERE SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE EVALUATED?

Requirements	SR_2 – Scalability SR_3 – Minimal performance overhead SR_4 – Preciseness SR_5 – Reusability
Solution	LOCALIZED QOS OBSERVER
Description:	**Evaluate the performance-related QoS measurements locally at each client or service.**
Forces:	- the scalability increases because all the measurements are evaluated locally - a minimal performance overhead is possible - reusable in case the evaluation is not implemented with the clients' or services' implementation
Consequences:	- the evaluation can impact the measuring results, leading to imprecise evaluation results
Solution	CENTRALIZED QOS OBSERVER
Description:	**Submit the performance-related QoS measurements to a CENTRALIZED QOS OBSERVER that evaluates the measurements regarding the negotiated SLAs.**
Forces:	- reusable for all clients and services - precise evaluation results
Consequences:	- the scalability can decrease, because - a performance overhead is possible because all clients and services must transmit the measurements to the CENTRALIZED QOS OBSERVER

Table 5: WHERE SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE EVALUATED?

A.5 WHEN SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE EVALUATED?

Requirements	C_2 – Dependency on the third parties' quality C_3 – Prevention of SLA violations SR_2 – Scalability SR_3 – Minimal performance overhead SR_4 – Preciseness
Solution	ONLINE QOS OBSERVER
Description:	**Evaluate the performance-related QoS measurements with regard to the negotiated SLAs during the SLAs' validity.**
Forces:	- SLA violations can be prevented - performance drops of the third-party services' quality can be recognized on time
Consequences:	- the scalability decreases - a performance overhead is possible - evaluating the performance-related QoS measurements immediately can impact the measuring of other QoS properties
Solution	OFFLINE QOS OBSERVER
Description:	**Store the performance-related QoS measurements during the SLA's validity. Evaluate the stored measurements at the end of the SLA's validity.**
Forces:	- SLA violations can be detected - a minimal performance overhead is provided - the scalability increases - the preciseness of the evaluation results depends on the storing solution
Consequences:	- SLA violations cannot be prevented - detecting performance drops of the third-party services' quality is difficult

Table 6: WHEN SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE EVALUATED?

A.6 WHERE SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE STORED?

Requirements	SR_2 – Scalability SR_3 – Minimal performance overhead
Solution	LOCALIZED QOS STORAGE
Description:	**Store the performance-related QoS measurements and/or evaluation results locally at each client or service.**
Forces:	- minimal performance overhead - the scalability decreases
Consequences:	- influences the evaluation solution regarding performance overhead and scalability
Solution	CENTRALIZED QOS STORAGE
Description:	**Transmit the performance-related QoS measurements and/or evaluation results from each client or service to a centralized component that stores the data in a CENTRALIZED QOS STORAGE.**
Forces:	- easier CENTRALIZED QOS EVALUATION because the measurements do not have to be collected together
Consequences:	- the number of services and clients influence the scalability - a minimal performance overhead can be detected

Table 7: WHERE SHOULD THE PERFORMANCE-RELATED QOS MEASUREMENTS BE STORED?