Implementing a Framework for QoS Measurement in SOA

A Uniform Approach Based on a QoS Meta Model

Andreas Hausotter, Arne Koschel
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science,
Hannover, Germany
email: Andreas.Hausotter@hs-hannover.de
email: Arne.Koschel@hs-hannover.de

Johannes Busch, Markus Petzsch, Malte Zuch
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science,
Hannover, Germany
email: Johannes.Busch@stud.hs-hannover.de
email: Markus.Petzsch@stud.hs-hannover.de
email: Malte.Zuch@hs-hannover.de

***Abstract*—The globalized markets now offer customers a variety of products and services as never before. This enormous selection reduces the loyalty of the customers to the companies. Today, products and services are highly interchangeable. This requires a strategic rethinking of the companies from a product-centric perspective to a customer-centric perspective. Therefore, businesses need to change their operational processes in a flexible and agile manner to maintain their competitive edge. A Service-oriented Architecture (SOA) may help to meet these need. Particularly in the insurance industry, this is an established key technology. Old monolithic software architectures have already been successfully transformed to 'traditional' SOAs. This could even be a good basis for future upgrades to micro-service architectures. But, before this upgrade is accomplishable, it is necessary to analyze and measure the already established SOA. As the application landscape of enterprises is inherently heterogeneous and highly distributed, it is a great challenge to provide services with a certain quality. A measurement system can help to capture the relevant QoS parameters of a software architecture. This is, particularly the case, when services are requested externally via the web. Therefore, quality of service (QoS) measurement and analysis is a crucial issue in Service-oriented Architectures. As the key contribution of this paper, we present a generic SOA Quality Model (SOA QM) based on the measurement standard ISO/IEC 15939, a SOA Information Model (SOA IM), and an architectural concept of a QoS System. The SOA IM is an XML-based specification for the measurement to be performed. The QoS System provides an execution platform for the SOA IM, based on a Complex Event Processing (CEP) approach and guarantees minimal impact on the SOA environment. The concepts are explained in detail using a standard process of the German insurance domain. Moreover, implementation details for our concept are given, including an overview of the general deployment of such a technology.**

Keywords—Service-oriented Architecture (SOA); Quality of Service (QoS); Measurement Process; Complex Event Processing (CEP).

I. Introduction

Distributed IT-systems are commonly used in today's companies to fulfill the needs of agility and scalability of their business processes to manage the highly variable demand of the market. Typical scenarios are real time logistics and delivery, just in time supply chain management and in general, handling services in real time to fit market demands.

The latter is commonly used within the finance and insurance industry during their internal computation of risk and money management and for their external customer services, like proposal calculations (including the current market conditions). Especially the external customer services must have a high quality in terms of time behavior. Google has shown that a latency of 100 ms up to 400 ms causes an impact of -0.2 % up to -0.6 % concerning the daily usage of web services by the customers [2]. The integration of those services to run business processes in a stable way, fulfilling the varying demand of the market, is commonly realized with Service-Oriented Architectures (SOA).

These architectures integrate services within distributed systems to run business processes with a high capability in terms of agility. Especially the distribution of the services over several systems allows scaling with the market demands.

Distributing and handling several services is a common concern of the insurance industry. But an increasing distribution and more complex business processes will only gain more agility with SOA, if the distribution of the services over several systems is realized in a reasonable way. For getting the required control of the distribution of those services, a measurement system is required. Measuring the general quality of service (QoS) in distributed systems is part of the motivation of this work and is explained in detail in the next subsection. The subsection thereafter will show our contribution to the general problem of measuring QoS in SOA within the application scenario of the insurance industry. This scenario is detailed in Section III.

A. Motivation

In many cases it is not possible to forecast, how much computing power and bandwidth the infrastructure needs to host the allocated services within the distributed computing system. Beside these design decisions of the infrastructure, there is a further problem in allocating the services to the right locations within the distributed system. This allocation will influence how much bandwidth and calculation power is available for the services and how many services will share identical resources during the same time. So, if several services will use the same part of the infrastructure, this could lead to increasing latencies over the whole system, resulting in an unfavorable time behavior for the users. Especially, if some services are requested with intense demands of the market, latencies could rise in an unpredictable manner.

This research is partly related to major German insurance companies, where latency considerations are particularly important. In this regard, research and prototype development can be tested by a quality of service (QoS) measurement system on a practice-relevant application case. In this case, the insurance companies' partners provide a typical service-oriented software architecture (SOA) for running business processes. In this context, insurance companies have to guarantee short response times of their services. Other quality of service parameters such as the measurement of reliability, resource consumption or error rate would also be possible. In the context of the practical application case, here the focus is on the measurement of the response time as it is particularly significant in the insurance industry.

Such a scenario is typical for the German insurance industry. At the end of the year, millions of users are able to switch their insurance contracts and will request therefore designated online services. The general demand is not foreseeable and the intense interaction between the insurance industry and the finance industry requires a high quality of those services. Especially, the historically low interest rates in today's market provokes, fast changing business models and the need for a fast adoption to new business processes.

Moreover, the ability to offer services of high quality to fulfill the external user demands and the internal interaction within the finance industry is very much required. To fulfill these demands, distributed systems with SOA will benefit from an across broader measurement of the quality of those services, particularly in terms of latency. Such a measurement system is the contribution of this work and is explained in the following subsection.

B. Contribution

The need for a new development of a flexible measurement system is influenced by the limitations of common solutions. As a result, the scenario of this work is based on a German insurance companies SOA, which already uses Dynatrace as a measurement solution [3].

Since the partner from the insurance industry is currently restructuring and modernizing his business processes and therefore, he needs a more flexible and generic approach to integrate an external measurement system for monitoring and analyzing the time behavior of his services. Additionally, a more detailed analyzer component was required to process the measured data.

So on the one hand, the approach has to be integrated in a generic way with minimal interaction points within the SOA of the partner from the insurance industry to guarantee a simple integration during the continuous development process. But on the other hand, the solution should offer a flexible and detailed analyzer component.

This article will present our currently ongoing applied research work. It extends our previous work from, mentioned in [1]. Since as a whole it is still 'work in progress', here we will mostly focus on measurement concepts and an adequate measurement model here.

The concrete presentation of the results and the evaluation would be beyond the scope of this work. Therefore, it will take place directly in the next publication, which will be in the direct context of this work. This following publication is currently in the process of submission to the conference 'The Tenth International Conference on Advanced Service Computing - SERVICE COMPUTATION 2018' and will appear there as soon as the acceptance is completed. In that case, the evaluation of the measuring system will be explicitly discussed there. Here in this work the preparatory presentation of the concept of the QoS measuring system takes place.

This work is the extension of our previous publication 'Agent based Framework for QoS Measurement applied in SOA' [1]. In this regard, we provide much more concrete technical details for our current prototype implementation. Based on the previous publication, this work here shows in detail how the required QoS parameter (the response time) is measured.

In particular, we will present more details on service call sequences and measurement within our framework (cf. Section VI-C). In addition, more implementation details on software, hardware and deployment of our prototype are presented in the whole Section VI. Detailed QoS measurement results will be presented in future work.

The required solution was defined by the following:

- generic approach to generate the measurement system,
- automatic integration of the measurement system into the existing SOA,
- lose couplings within the existing SOA,
- flexible agent based approach,
- technology independent approach using standards (XML),
- individual and customizable analyzer component.

As stated above, besides technical concepts, we will also present some details mainly from our utilized application scenario, which is based upon the ideas from the 'Check 24' process. Within this process, different offerings for the same kind of insurance are compared. These offerings typically origin from several insurance companies. For example, there are different offerings for car insurances. Based on certain input parameters, the end user eventually gets different insurance offers by this process. The proposal service used by 'Check 24' is a common service throughout the German insurance sector and is implemented by various insurance companies.

This service can be called externally by applications such as 'Check 24' through a common interface given by a so called 'BiPro specification'. BiPro is widely used throughout the German insurance sector and the availability of these services has a significant impact on competitiveness. Internally the proposal service is, for example, used in the process 'Angebot erstellen' ('create proposal') of the general German 'Versicherungsanwendungsarchitektur (VAA)' (cf. [4]). The VAA describes a set of standardized insurance processes working within a generalized 'insurance application architecture'. Our project partner has implemented a similar process for its own agent's respective customer portal.

The remainder of this paper is structured as follows: In Section II we discuss some related work. Section III describes our application scenario in some detail. In Section IV and Section V our general Quality of Service (QoS) measurement model and overall concepts are described. Section VI looks at implementation details of our prototypic work. Finally, Section VII concludes this paper and gives some outlook to future work.

II. PRIOR AND RELATED WORK

In prior work, we already discussed several aspects of the combination of SOA, Business Process Management (BPM), Workflow Management Systems (WfMS), Business Rules Management (BRM), and Business Activity Monitoring (BAM) [5][6][7] as well as Distributed Event Monitoring and Distributed Event-Condition-Action (ECA) rule processing [8][9]. Building on this experience, we now address the area of QoS measurement for combined BRM, BPM, and SOA environments within the (German) insurance domain context.

Work related to our research falls into several categories. We will discuss these categories in sequence.

General work on (event) monitoring has a long history (cf. [10][11] or the ACM DEBS conference series for overviews). Monitoring techniques in such (distributed) event based systems are well understood, thus such work can well contribute general monitoring principles to the work presented here. This also includes commercial solutions, such as the Dynatrace [3] system or open source monitoring software, like for example, the NAGIOS [12] solution. In these systems, there is however, no focus on QoS measurement within SOAs. Also, they usually do not take application domain specific requirements into account (as we do with the insurance domain).

Active DBMS (ADBMS) offer some elements for use in our work (see [13][14] for overviews). Event monitoring techniques in ADBMSs are partially useful, but concentrate mostly on monitoring ADBMS internal events, and tend to neglect external and heterogeneous event sources. A major contribution of ADBMSs is their very well defined and proven semantics for definition and execution of Event-Condition-Action (ECA) rules. This leads to general classifications for parameters and options in ADBMS core functionality [14]. We may capture options that are relevant to event monitoring within parts of our general event model. QoS aspects are handled within ADBMS, for example, within the context of database transactions. However, since ADBMSs mostly do not concentrate on heterogeneity (and distribution), let alone SOAs, our work extends research into such directions.

The closest relationship to our research work is which directly combines the aspects QoS and SOA. Since 2002, several articles fall into this category. However, in almost all known articles, the SOA part focuses on WS-* technologies. This is in contrast to our work, which takes the operational environment of our insurance industry partners into account.

Examples of WS-* related QoS work include QoS-based dynamic service bind [15][16], related WS-* standards such as WS-Policy [17], and general research questions for QoS in SOA environments [18].

Design aspects and models for QoS and SOA are addressed in [15][19][20][21][22], SOA performance including QoS in [23], and monitoring for SOA is discussed in articles like [24][25][26][27].

While the above articles discuss several aspects of QoS work including performance monitoring in SOA style environments, none of them takes a standards based insurance domain into account. In contrast, however, we do so by grounding our work on typical insurance services and processes – the Check 24 service and the German 'Versicherungsanwendungsarchitektur (VAA)' (cf. [4]) – as well as in the German-speaking countries (cf. [30]) frequently utilized ISO/IEC 9126 standard for our adjusted SOA quality model.

III. APPLICATION SCENARIO

Customers are using online platforms to compare the conditions and proposals offered by different companies. The online platform check24.com allows customers to compare different insurance proposals. Therefore, the insurance companies need to respond to those requests to be aware of potential customers on such platforms. The underlying scenario for this work is a service for calculating individual proposals for such online platforms. This scenario is automatically requested by the online customer information platform and needs to respond in a timely manner. The business process, for calculating the proposal, follows four steps:

- check input parameters for plausibility,
- call all additional relevant services to get required data,
- calculate the proposal based on internal business rules,
- deliver the proposal to the requesting online platform.

The partner from the insurance industry has already developed a distributed system to create and run such business processes. This system uses the approach of SOA and integrates various SOA-style services located across several locations.

Measuring the time behavior is a feasible approach to maintain the overall system and scale it to changing market demands to fulfill the required quality of such services (QoS). The distributed system is designed with the concept illustrated in Fig. 1.

The system part alpha is the enterprise service bus (ESB) of the system, which is responsible to integrate the business processes with further applications and services. These business processes are parameterized by specific business rules, stored in a business rule database.

The communication with this business rule database is realized via web service calls. In general, alpha is the central communication component of the system.

The system part beta is the current process engine to run the business processes and is connected via JMS with alpha. These

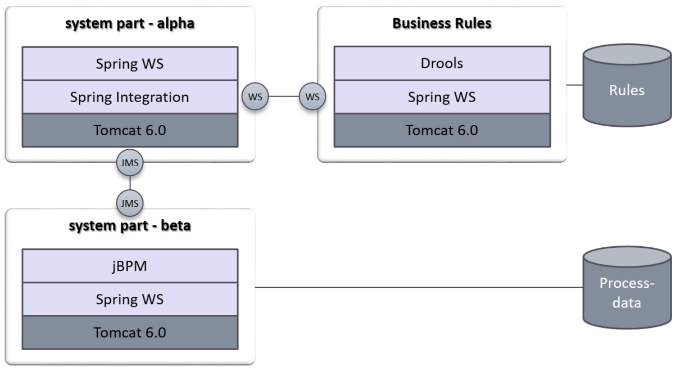

Figure 1: The application scenario

business processes are influenced by the stored business rules and the business process data, which are stored in a separated database.

This distributed system defines the scenario where several services are parameterized, called and integrated (via alpha) over several locations. The generic and XML-based measurement concept of this work will use this scenario to measure QoS-Parameters, especially the time behavior of services. The specific measurement model is described in the next section.

IV. MEASUREMENT MODEL

The assessment of the QoS in Service-oriented Architectures is based on a *SOA Quality Model (SOA QM)*, which combines characteristics and sub-characteristics in a multilevel hierarchy. For this purpose, we adjusted the ISO/IEC-Standard 9126 to meet the SOA-specific requirements. Fig. 2 illustrates the characteristics, sub-characteristics and relationships between these concepts. In our research work, we will focus on the *Time Behavior*, which contributes to *Efficiency*.

Although ISO/IEC 9126 was revised by the ISO/IEC-Standard 25010 (*Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*, cf. [28][29]) we use ISO/IEC 9126 as a starting point because of its high degree of awareness in German-speaking countries (cf. [30]). Moreover, the German version of the ISO/IEC 25000 series has been prepared by the German Institute for Standardization (DIN) but is not yet available (cf. [31]).

Instead of applying the quality metrics division of SQuaRE (i.e., ISO/IEC 2503x), our approach is based on the comprehensive ISO/IEC-Standard 15939 (cf. [28]). The basic model, as found in similar form in the contribution of Garcia et al. ([32]), has been aligned and extended by quality requirements, quality models, and some system components. In the following subsections, we describe the main concepts of our *SOA Measurement Information Model (SOA MM)* as shown in Fig. 4.

A. Information Need and Information Product

The determination of the QoS in a SOA is always demand-driven, since both the specification ('*What* and *how* should be measured?'), execution of the measurement itself and the subsequent interpretation of the results can cause a significant organizational and technical effort.

So, first of all, the *Information Need* with objectives, potential risks and expected problems has to be defined and documented properly. In terms of the application scenario presented in Section III, the objective is to assess the performance of the business process for calculating the offer in order to identify and resolve problems in time.

B. Core measurement process

The *Information Product* is the result of the execution of the *Core Measurement Process* as depicted in Fig. 3 (cf. [33]). The *Information Need* provides the input for the sub-process *Plan the Measurement Process* (planning stage) and sub-process *Perform the Measurement Process* (execution stage) generates the output, i.e., the Information Product. The process goal is to satisfy the Information Need. All concepts presented below directly or indirectly contribute to the Information Product.

C. Concepts of the planning stage

SOA Services are investigated concerning their QoS, is the main focus of this research work. For this purpose, *Quality Attributes* are measured. In this context, the *Measurable Concept* outlines in an abstract way, *how* the attributes values are determined to satisfy the required Information Needs. In doing so, it references one or more sub-characteristics of the SOA QM.

For the application scenario described in Section III, the process performance is to be determined first and then evaluated. The corresponding Measurable Concept is the calculation of the processing time. To do this, instantiation of a process and termination of the process instance are to be determined. The process identification represents the Quality Attribute to be measured, and the sub-characteristic, referenced by the Measurable Concept, is the Time Behavior.

In order to implement the Measurable Concept and to perform measurements of attributes, first of all *Measures* are to be specified. A Measure assigns each Quality Attribute a value on a *Scale* of a particular *Type*. The ISO/IEC-Standard 15939 provides 3 different types of Measures, namely Base Measures, Derived Measures, and Indicators respectively.

A *Base Measure* specifies by its *Measurement Method* how the value of a Quality Attribute is to be determined. It is always atomic and therefore independent on other Measures.

A *Derived Measure* uses one or more Basic Measures or other Derived Measures, whilst the *Measurement Function* specifies the calculation method and thus the combination of the Measures used.

For the application scenario illustrated in Section III, the Basic Measures process instantiation t_{inst} and process instance termination t_{term} are specified. The identification of the processes instance $piID$ represents the Quality Attribute measured by t_{inst} and t_{term}. As the Measurement Method, we select the time of the start and end event respectively. The

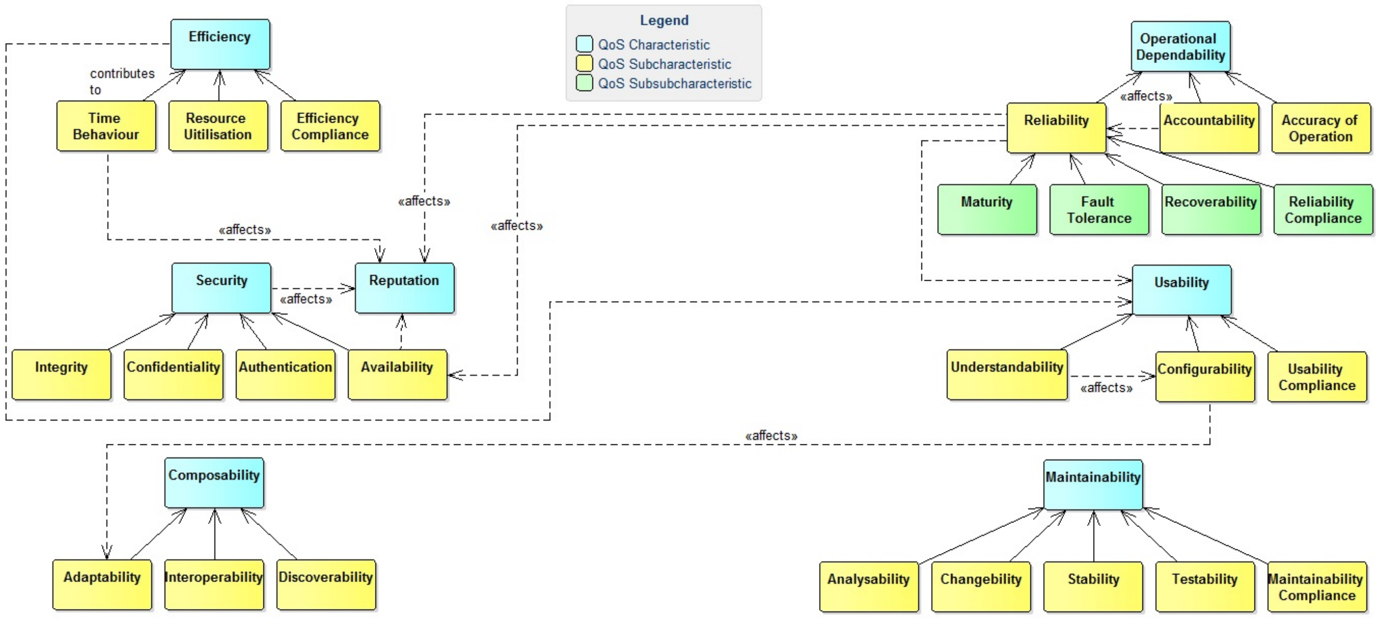

Figure 2: SOA Quality Model

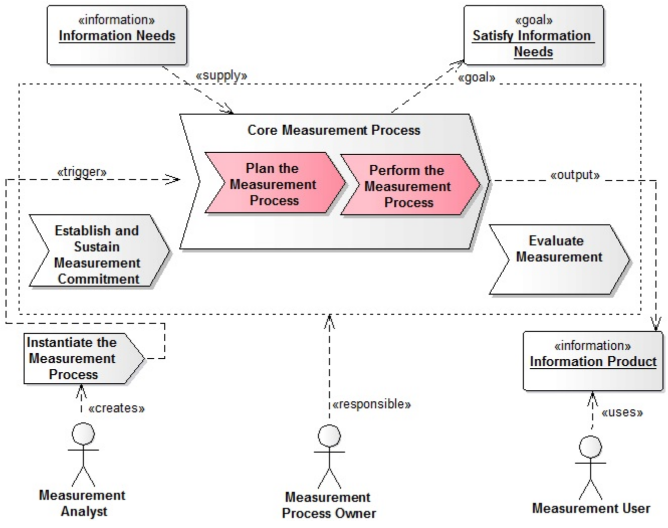
«information» Information Needs
«goal» Satisfy Information Needs
«supply»
«goal»
Core Measurement Process
Plan the Measurement Process
Perform the Measurement Process
«trigger»
«output»
Establish and Sustain Measurement Commitment
Evaluate Measurement
Instantiate the Measurement Process
«information» Information Product
«creates»
«responsible»
«uses»
Measurement Analyst
Measurement Process Owner
Measurement User

Figure 3: Process for determining the QoS (cf. [33])

Derived Measure processing time of the instance T_{Proc} will be calculated by the Measurement Function

$$T_{Proc}(piID) = \Delta t = t_{term}(piID) - t_{inst}(piID) \quad (1)$$

Another Derived Measure R calculates the percentage of the process instances within a given time period. Than there is a check, if the processing time T_{Proc} exceeds twice the standard deviation. R will be calculated by the Measurement Function

$$R = \frac{m}{n} \cdot 100 \quad (2)$$

where n is the number of all process instances in the given time period and m the number of process instances with $T_{Proc} \geq 2 \cdot \sigma$. The standard deviation σ is given by the formula

$$\sigma = \sqrt{\frac{1}{n} \cdot \sum_{1 \leq i \leq n} (T^i_{Proc} - \overline{T_{Proc}})^2} \quad (3)$$

Finally, an *Indicator* is a qualitative evaluation of Quality Attributes, which directly addresses the issue raised in the Information Needs. Indicators always use a nominal scale with qualifying values and thus show if necessary action is needed for further root cause analysis. An Indicator is derived from other Quality Measures, i.e., Base and Derived Measures, and Indicators. The combination of the Quality Measures used and the method of calculation is based on an *Analysis Model* in conjunction with *Decision Criteria* using thresholds and target values.

For the application scenario illustrated in Section III, the indicator adequacy of the processing time of all process instances in a given time period $SLoT_{Proc}(R)$ is based on the Derived Measure R according to Table I:

TABLE I. ADEQUACY OF THE PROCESSING TIME

R	$SLoT_{Proc}$
$0\% \leq R < 3\%$	high
$3\% \leq R < 5\%$	medium
$5\% \leq R < 100\%$	low

D. Concepts of the execution stage

After the concepts of the planning stage have been presented, now we explain the execution phase in brief (subprocess *Perform the Measurement Process*, depicted in Fig. 3). Section V will discuss their conceptual implementation in more detail.

The actual measuring procedure, i.e., the execution of the instructions for determining the value of a Quality Attribute, is called *Measurement*. Hereby, *Measurement Results* are created, collected in a container, namely *Data*, which is inserted into a *Data Store*.

The measurement system comprises different supporting software components, which are conceptually presented in Section V. The *QoS Measurement* performs the instructions specified in the Measurement Method or Measurement Function respectively, to generate the Measurement Results for further processing. The *QoS Analyser* performs the statistical analysis and evaluation of the collected data and creates the Information Product. The *QoS Reporting* makes the Information Product available to the *Measurement User* (cf. Fig. 3).

E. QoS Measurement Information Model

We designed a domain-specific language to specify the values of the concepts introduced above according to the Information Need. This specification document is referred to as *QoS Information Model (QoS IM)*. The aim of this approach is to automate the measurement process by the generation of artifacts required by the QoS system to execute a measurement.

The QoS IM consists of an abstract and a concrete section. In the abstract section, the concepts of the Planning Stage and partly the Execution Stage are specified. In the concrete section, the implementation specific definitions are explained. Since our QoS-System is based on a complex event processing (CEP) approach, the specification of events, agents and rules is subject of this section.

A sophisticated XML Schema was developed to realize the domain-specific language. We opted for XML as a universally accepted standard that is highly flexible, platform and vendor independent and supported by a wide variety of tools. In a follow-up project an XText-based tool will be developed that generates the (XML) QoS IM from a (XText) source code.

Its semantic model is shown in Fig. 4. The following rules for modeling apply:

- Concepts are mapped to XML elements (graphically represented by UML classes).
- Details of a concept are mapped to XML attributes of the owning element (graphically represented by UML instance variables).
- If possible, relationships between concepts are mapped to element hierarchies (graphically represented by UML associations).
- Otherwise, they are mapped to constraints (i.e., keyrefs) (graphically represented by UML dependencies).

V. Measurement Concept

In Section IV, a QoS IM based upon a SOA QM is described. To execute a specific QoS IM (and thus sub-process 'Perform the Measurement Process') an execution platform is needed. This platform and the underlying QoS architecture are given in this section. First reasons for choosing this specific architecture are discussed shortly. Furthermore, an overview is shown, detailing in the central agent concept and CEP.

A. Design decisions

As described above, the goal of the measurement concept is to provide the execution platform for a specific QoS IM. Therefore, basic design criteria for the measurement concept are derived from the QoS IM. Furthermore, quality criteria are given, which also have to be considered in the architecture design. These criteria are:

- measurement of Quality Attributes as described by QoS IM,
- flexibility of measurement and computation,
- low impact (modification, performance, etc.) onto SOA components.

The proposed Measurement Concept is based upon a general architecture given in [34]. The basic idea is to separate the measurement (e.g., sensors, agents, etc.) and 'analysis and statistics' functionality into different modules. This separation opens the opportunity to cater each module to their specific functional and quality requirements.

Overall, the given general architecture already fulfills the requirement to measure Quality Attributes and provide the needed evaluations to produce Measurement Results and Information Products.

The measurement module has to provide the QoS System with information about the observed service. To provide the needed flexibility a sensor has to be placed into it. To keep the impact onto the SOA at a low level, an agent based approach was chosen. Agents capsule the needed parsing and computation and thus can be easily integrated into arbitrary SOA modules. Furthermore, minimizing the performance impact (through threading, non-blocking, etc.) can be integrated into the agents.

The 'analysis and statistics' module does not have these strict requirements on performance impact. Flexibility of computation and measurement execution is the main quality requirement. Thus, a platform approach was chosen. Basically, artifacts generated through the QoS IM are placed into the QoS Platform and executed.

B. Overall system architecture

On a high level, the QoS System splits the QoS Measurement Agents and further processing (QoS Platform) into different components. This approach allows to easily split these components into different processes to comply with the quality requirements. While the Measurement Agents (encapsulating the agent concept) represents the client component, the server component is represented by the QoS Platform and contains the CEP engine and further analysis processing. Fig. 5 shows a high level overview of important components and their relationships.

The general purpose of the Measurement Agents is to emit specific events based on the defined Base Measures. As described in Section IV events are emitted, e.g., for process instance instantiation/termination. In general, concepts

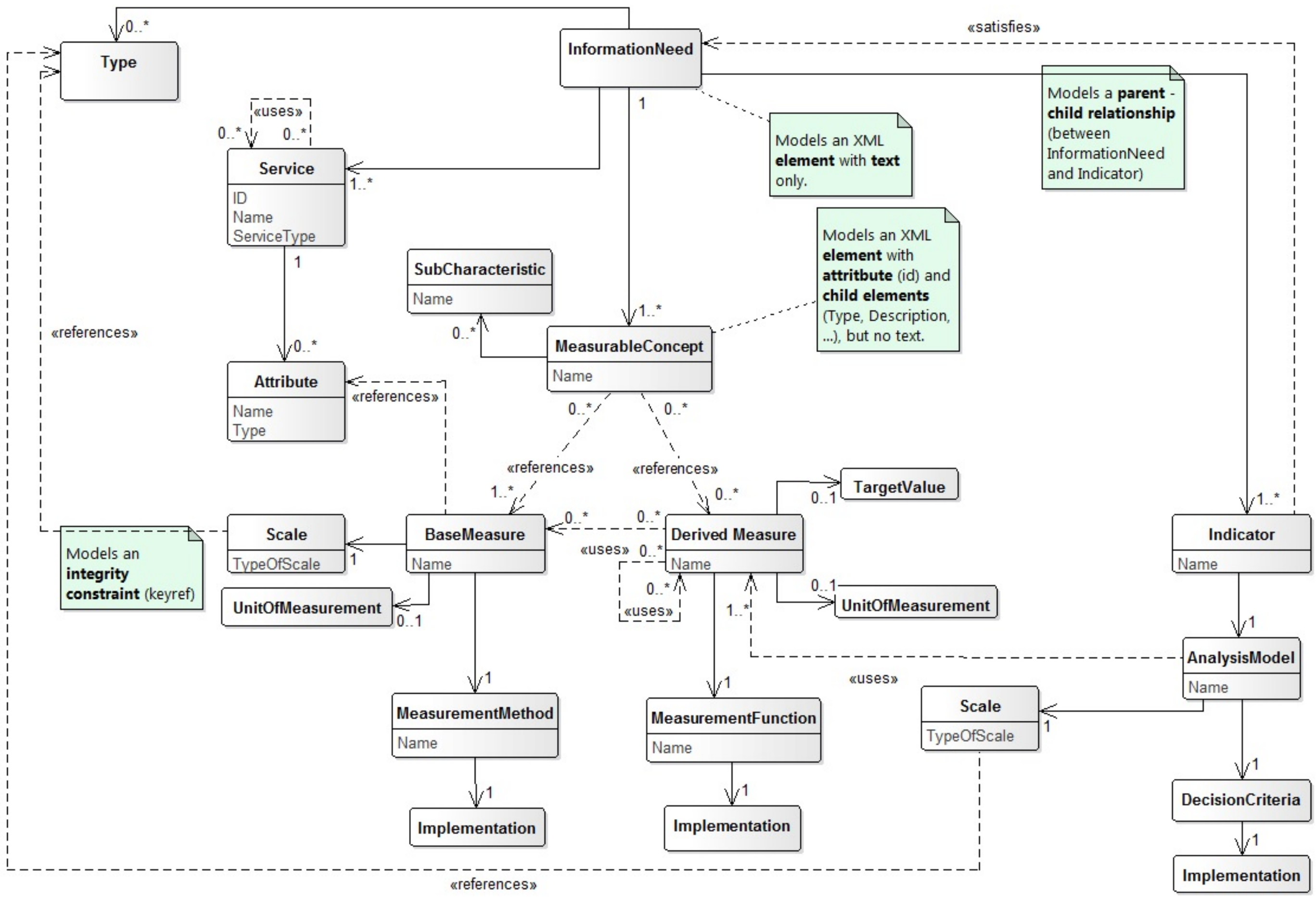

Figure 4: QoS Measurement Information Model (QoS MIM)

for agent implementation can be categorized by agent location and time of execution (cf. [35] and [36]). To measure a specific process instance, agents can be placed into corresponding SOA service calls. Thus, Measurement Agents are only logically placed into the QoS System component. One and currently used approach is to use the concept of interceptors, which offers a low modification impact and can deliver precise Measurement Results.

The QoS Platform consist of several components, most notably the QoS Measurement and QoS Analyzer. In general, the purposes of these modules are to collect, clean and compute the emitted events and provide further analysis of stored Measurement Results (specifically stored as complex events).

Before any event is given to the Measurement Method, it will be handled by the control module. Purpose of this module is event routing, general cleaning steps and an optional filter step. Cleaning (or formatting) events in the analyzer is needed because Measurement Agents are placed in the monitored system, thus shall minimize their performance impact. The Measurement Method is implemented as a CEP rule executed by the engine and emits complex events for further near real-time processing and long term analysis.

The QoS Analyzer module provides a basis for statistical analysis and evaluations. Every complex event is stored into a Data Store implemented as a relational database. The different analysis and evaluations defined by Derived Measures and Indicators are implemented through SQL and plain Java. Furthermore, the module provides an interface to the computed Information Product. As of now, this interface is not further defined. A comprehensive way to define a QoS specific interface is given in [37]. It is defined upon the ISO/IEC 25010 quality standard and specifically includes the ability to describe relationships between QoS attributes. This corresponds with our presented QoS IM approach.

C. Applying the described measurement concept

In Fig. 6, the given measurement concept (QoS Platform) is applied onto the application scenario, thus providing the missing link between the QoS IM and the insurance based application scenario. In this example, a simplified scenario is used consisting only of an external ’Check24’ mock-up service (representing a simple consumer), the central ESB and the proposal service (which represents the producer). The task of the measurement model and thus the concept is to measure the

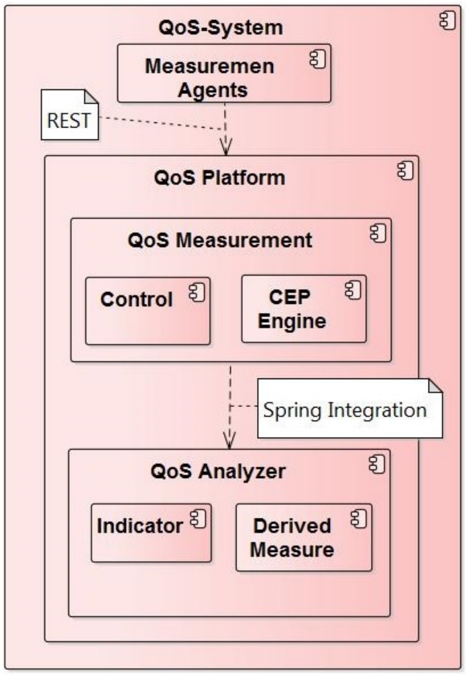

Figure 5: The QoS architecture

processing time of this service and to compute the Information Product for further evaluations.

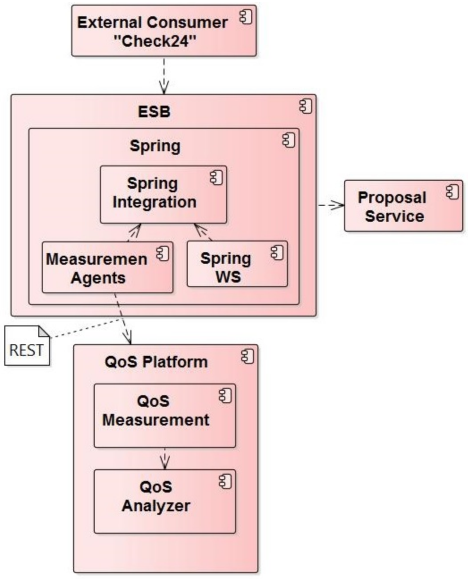

Figure 6: Applied QoS architecture

To measure this service the Measurement Agents, defined through base measures, are placed directly into the ESB. This offers a measurement independent of service location and different load balancing scenarios. To minimize the integration effort, functionality given by Spring Integration is extensively used (especially the interceptors for message queues). In this simplified example, base measures (and thus the agents) only determine service call start / end times and announces these

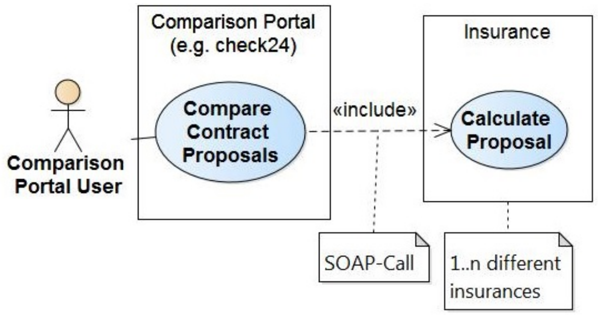
Comparison Portal (e.g. check24)
Insurance
Compare Contract Proposals
«include»
Calculate Proposal
Comparison Portal User
SOAP-Call
1..n different insurances

Figure 7: Use case diagram for Proposal Service

to the QoS Platform. Furthermore, the agents try to minimize their performance impact by using non-blocking techniques and performing only necessary parsing steps (e.g., service call IDs, etc.). These will be shown in detail in further publications.

As described above, the QoS Platform performs further cleaning and processing steps to compute the QoS IM Indicators ($SLoT_{Proc}(T_{Proc})$) and provides these to downstream systems (e.g., reporting, presentation, load balancing, etc.).

VI. IMPLEMENTATION DETAILS

Based on the above conceptual groundwork, this section contains a deeper insight into several implementation areas of the QoS System. First, a description of the measured services is given, providing further information about the application scenario. Second, the process of applying a specific QoS IM is shown. After that a detailed walk through of a QoS Event call is presented, describing the different stages and components that are passed through. Furthermore, the distribution of the components is shown, which culminates into a testable runtime environment.

A. Proposal Service

In Section III, the overall application scenario is described. Detailing on this a use case diagram is given in Fig. 7. When a user (such as a customer) requests a proposal for a new insurance contract, some information (birthday, coverage, etc.) has to be given to complete the request. Furthermore, certain parameters can be adjusted to gain better results.

After initial parameter checks (validity, completeness, etc.), requests are passed to different companies by SOAP web service calls containing all entered information. Responses (computed proposals) are only considered when they arrive in a timely manner to provide acceptable user experience. This imposes a hard boundary for the QoS Measurement because it has a considerable business impact in case of failures. If a customer decides to sign a new contract, further business processes (e.g., prepare an insurance offer) are started.

To evaluate the general approach and the QoS Platform, a Proposal Service is needed. Using an actual service of the insurance company wasn't possible due to the potential business impact. Also the implementation of a comprehensive business logic was considered to be too complex and thus abandoned. On the other hand, only special information (e.g.,

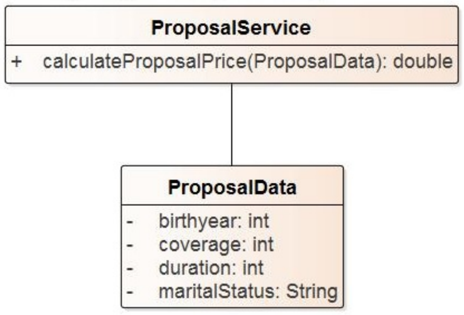

Figure 8: Interface of the (Mock) Proposal Service

correlation id) of the SOA (SOAP call) is needed to determine the Indicator. The Proposal request and result data isn't considered. Therefore, we decided to develop a Mock Service, which also provides the opportunity to customize the desired load behaviour.

The interface of the Mock Proposal Service is given in Fig. 8. It consists of a method returning the monthly cost of the contract to the customer. To calculate the costs, different information is required. For reasons of simplicity, the number of parameters was reduced to four attributes of a customer, in particular her/his birth year, insurance coverage, duration of the contract, and the marital status. Furthermore, these parameters are used in the Mock Service to control the load behaviour.

B. Relationship between IM and QoS platform

In Fig. 9, the general process for applying a specific IM by the QoS Platform is shown. It is divided into different phases. Not all phases have to be executed on each change of the IM.

The first step is to develop an IM for a specific Information Need. As described before, an IM consists of abstract and concrete parts. To satisfy the Information Need, an Indicator and thus different Derived and Basic Measures have to be defined. This represents the first phase. In addition to the abstract part, every Measure and Indicator contains a concrete part. The specification of these parts represents the second phase. In order to develop concrete parts, specific technologies and frameworks have to be chosen, which have to be provided by the QoS Platform or added to it if missing.

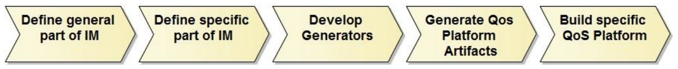
Define general part of IM
Define specific part of IM
Develop Generators
Generate Qos Platform Artifacts
Build specific QoS Platform

Figure 9: General Process of applying a specific IM

The goal of next phase is to develop the different Generators. This can be skipped if fitting Generators are already developed. The QoS Platform is designed around the concepts of extensibility and modularization. The same ideas were significant when designing the QoS Generator. In general, it provides a parser step to build an optimized in-memory model of an IM. Following this step the Generators are executed. They use the concrete and abstract parts to generate usable

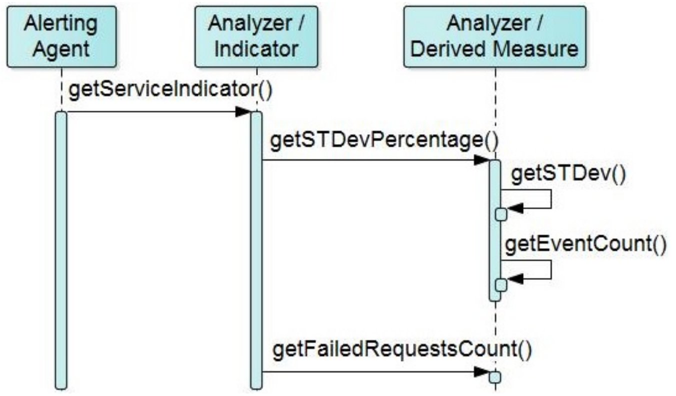

Figure 10: Determining an Indicator

artifacts (e.g., Java classes, SQL queries, etc.). A deeper insight into the QoS Generator itself and the concrete parts will be provided in future articles.

After the generation phase, the goal of the last phase is to build a specific QoS Platform by the integration of artifacts. This is done by developing the necessary bridges and transformations between an artifact and the interfaces (e.g., Event interface) of the platform. Furthermore, the integration of the agents into the SOA is part of this stage. After the last phase is completed, the QoS System can be deployed.

C. Computing an Indicator

The computation of an Indicator is split into two phases. Each phase is initiated by a particular system and refers to different sections of the QoS IM. Furthermore, a specific stage is executed by diverse modules of the QoS System.

The first phase to compute an Indicator is shown in Fig. 11. It details the creation of base events and the persistence of the resulting complex events for long term analysis. This step is executed by the Measurement Agents and QoS Measurement modules and focuses on short term computations.

The computation of an Indicator starts with a call of the corresponding domain service (process, business, or basic service) initiated by a service consumer. As shown in Fig. 11 currently the client is a JMeter based test client, which issues a SOAP call against the Proposal Service. This call is routed by the ESB to a corresponding service handler by passing through several (Spring based) message queues. A QoS Agent is attached to the ProposalRequestChannel as a message queue interceptor which is shown by the handleMessage(request) method call. All requests are parsed inside an Agent and relevant data (e.g., service call id, timestamp, etc.) are combined into raw event data package. A call of the QoS Measurement endpoint of the QoS Platform completes the task of an Agent.

Inside the Measurement module, a new QoS Event is created by formatting (and optionally filtering) the raw event data. Subsequently, this new event is inserted into the CEP Engine. Only if corresponding start and end QoS Events are inserted, the CEP rule is fired and a defined Measurement Method is executed. The method execution results in a complex event /

QoS Event which is send to the QoS Analyzer module. After persisting the event by the module, the first part is completed.

The second phase to compute an Indicator is given in Fig. 10. It includes the computation of further Derived Measures and concludes with the calculation of an Indicator. This step is executed by the QoS Analyzer module and focuses on long term analyses. Actually, this phase is executed only if it is requested by a downstream system (e.g., a system for alerting). For this, the downstream system sends a REST call to the Analyzer module of the Platform.

To get the current Indicator value, several Derived Measures have to be computed first. This is performed by the Derived Measure module. For this application scenario, the Indicator depends on several Measures based upon the standard deviation and number of 'failed' requests.

First, the percentage of requests with a duration above the doubled standard deviation is computed as shown by the getSTDevPercentage() method call. It is done by a combination of Java code and other Derived Measures. The Measures (e.g., getEventCount() method call) are computed by SQL queries against the Data Store. The percentage value is then computed by Java code. The getFailedRequestsCount() method call includes a SQL query. All long term analysis is performed by SQL queries and designed to be based on a specific time frame. The Indicator itself is determined via a Java based decision table which is created by the QoS Generator. The different computation approaches offer the required flexibility for various types of Indicators.

D. Deployment

Fig. 12 shows the current distribution of nodes and components within the QoS-aware SOA. The ESB and the Proposal Service are representing the core of the system. They are deployed in a combined .war file into a Tomcat web application server. Furthermore, it contains the Spring framework dependencies and a company-specific enterprise framework. The Measurement Agents are part of the Framework.jar. Also several configuration files are part of the .war, handling message routing, database access and further functions. The Service Schema .jar file bundles required schemas and classes to allow SOAP communication.

The Proposal Service (e.g., the Service handler and domain logic) itself is part of the ESB Service .jar file. All necessary configurations for an Agent to intercept a service request are part of this file. The corresponding database schema (e.g., for business objects, process and service call relevant data) is deployed into a DB2 database. A JMeter client was used during development and will be applied for further load testing. The Config.xml contains the SOAP request call and further JMeter configuration. The QoS Platform is deployed as a .war file into another Tomcat application server, separating the measured system from the analysis modules.

Table II summarizes our software versions in use. Especially the versions of the OpenSuse virtual machines are based on requirements of our partners from the insurance industry to achieve certain compatibility. The JMeter virtual machine is only relevant for functional testing. Thus, matching versions are irrelevant.

TABLE II. USED SOFTWARE PRODUCTS

Virtual Machine	Software versions
JMeter Ubuntu VM	Apache JMeter: 3.2, Oracle Java: 1.8.0.131, Ubuntu: 14.04.5 LTS
ESB OpenSuse VM	Apache Tomcat: 6.0.44, Oracle Java: 1.6.0.45, OpenSuse: 13.2
DB2 OpenSuse VM	DB2: 10.5.0.5, OpenSuse: 13.2
QoS Platform Ubuntu VM	Apache Tomcat: 6.0.45, Oracle Java: 1.6.0.45, Ubuntu: 14.04.5 LTS

For development and initial testing, the QoS-aware SOA is deployed into virtual machines for several reasons. First, this deployment provides an easy handover procedure with the partner companies. Second, a 'clean' environment for development could be achieved, which helps especially with the database and ESB parts. Third, the distribution of these VMs on a more sophisticated hardware platform is easier. This is the next step and will be followed with more complex tests and first measurements in our ongoing work.

VII. CONCLUSION AND FUTURE WORK

The presented approach for monitoring a distributed SOA environment is a promising path to take: Our SOA QM is aiming to follow the ISO/IEC-Standard 15939 (cf. [33]), which enables a wide range of use cases. The Measurement Concept outlines an execution platform for the specific QoS IM, which should cause minimal impact on the SOA environment. The separation of Measurement Agents and QoS-Analyzer allows lightweight agents on the one hand and a very capable analyzer component on the other hand.

Our ongoing work of applying the QoS System to an application scenario is very much relevant to our partner in the insurance industry (the 'Check 24 process'). It will provide evidence of the practical usability of the created framework. In the present article, the framework and the corresponding platform are applied onto a basic, business relevant scenario (the Proposal Service). Furthermore, it is planned to apply these techniques to the more complex process 'Angebot erstellen' ('create individual proposal') of the VAA, thus implementing a more complex scenario.

It is expected that the monitoring system will help to discover potential bottlenecks in the current system design of our partner's distributed services and therefore creating high value in the process of solving these issues.

In future work, the actual measurement and analysis of the results will be done. It is also planned to apply these results onto cloud based environments. Moreover, a further sub-division or extraction from the current coarse granular SOA services into more fine-grained micro-services 'where it makes sense', (e.g., to allow for a better scalability of individual micro-services) will be investigated by us in future work.

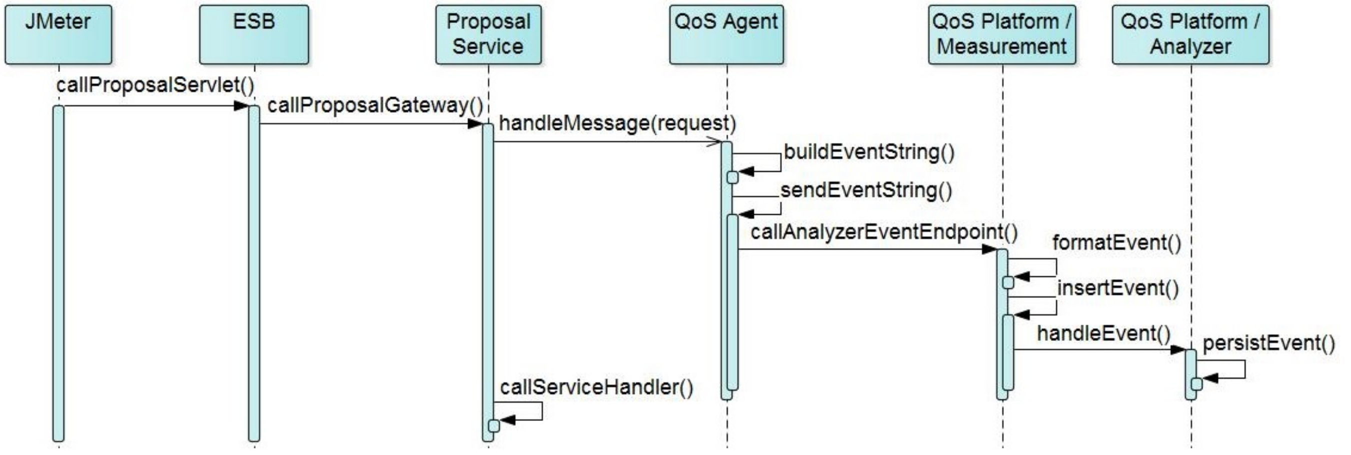

Figure 11: Integration and sequence of a QoS Event call

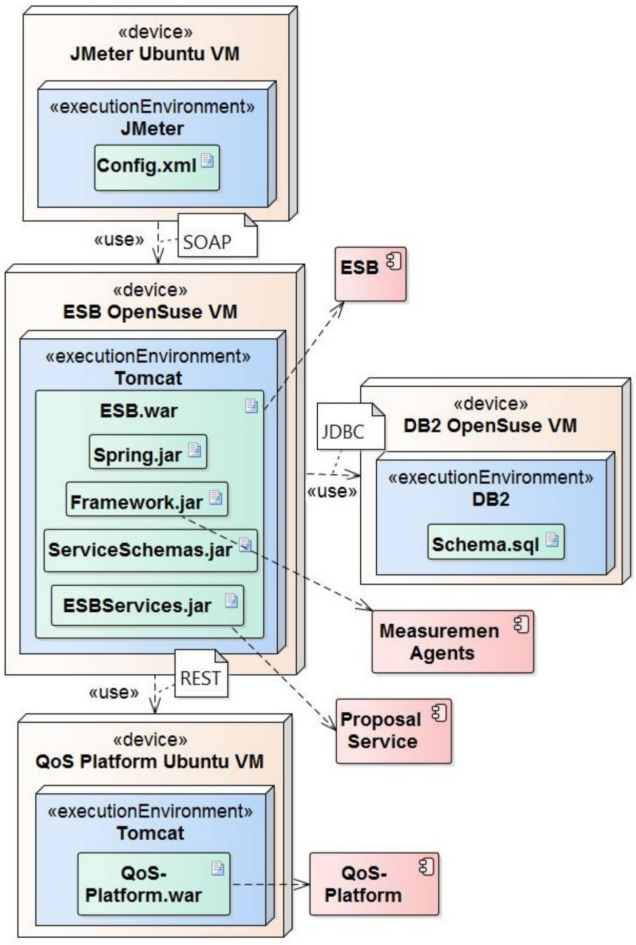

Figure 12: Deployment of the QoS system

References

[1] A. Hausotter, A. Koschel, J. Busch, M. Petzsch, and Malte Zuch, "Agent based Framework for QoS Measurement Applied in SOA," in: The Ninth International Conferences on Advanced Service Computing (Service Computation), IARIA, Athens, Greece, pp. 16-23, 2017.

[2] J. Brutlag, "Speed Matters for Google Web Search," Google Inc., Mountain View, 2009, [Online]. URL: http://services.google.com/fh/files/blogs/google_delayexp.pdf [accessed: 2016-12-26].

[3] Dynatrace LLC, "Dynatrace Application Monitoring," [Online]. URL: https://www.dynatrace.com/de/products/application-monitoring.html [accessed: 2016-12-26].

[4] GDV (Gesamtverband der Deutschen Versicherungswirtschaft e.V. – General Association o.t. German Insurance Industry), "Die Anwendungsarchitektur der Versicherungswirtschaft: Das Objektorientierte Fachliche Referenzmodell (The application architecture of the German insurance business – The functional object-oriented reference model", VAA Final Edt. Vers. 2.0, 2001, [Online]. URL: http://www.gdv-online.de/vaa/vaafe_html/dokument/ofrm.pdf [accessed: 2017-01-11].

[5] C. Gäth et al., "Always Stay Agile! – Towards Service-oriented Integration of Business Process and Business Rules Management," in: The Sixth International Conferences on Advanced Service Computing (Service Computation), IARIA, Venice, Italy, 2014, pp. 40-43.

[6] A. Hausotter, C. Kleiner, A. Koschel, D. Zhang, and H. Gehrken, "Always Stay Flexible! WfMS-independent Business Process Controlling in SOA," in: IEEE EDOCW 2011: Workshops Proc. of the 15th IEEE Intl. Enterprise Distributed Object Computing Conference, IEEE: Helsinki, Finnland, 2011, pp. 184-193.

[7] T. Bergemann, A. Hausotter, and A. Koschel, "Keeping Workflow-Enabled Enterprises Flexible: WfMS Abstraction and Advanced Task Management," in: 4th Int. Conference on Grid and Pervasive Computing Conference (GPC), 2009, pp. 19-26.

[8] A. Koschel and R. Kramer, "Configurable Event Triggered Services for CORBA-based Systems," Proc. 2nd Intl. Enterprise Distributed Object Computing Workshop (EDOC'98), San Diego, U.S.A, 1998, pp. 1-13.

[9] M. Schaaf, I. Astrova, A. Koschel, and S. Gatziu, "The OM4SPACE Activity Service - A semantically well-defined cloud-based event notification middleware," in: IARIA Intl. Journal On Advances in Software, 7(3,4), 2014, pp. 697-709.

[10] B. Schroeder, "On-Line Monitoring: A Tutorial," IEEE Computer, 28(6), pp. 72-80, 1995.

[11] S. Schwiderski, "Monitoring the Behavior of Distributed Systems," PhD thesis, Selwyn College, University of Cambridge, University of Cambridge, Computer Lab, Cambridge, United Kingdom, 1996.

[12] Nagios.ORG, "Nagios Core Editions," [Online]. URL: https://www.nagios.org/ [accessed: 2016-12-26].

[13] N. W. Paton (ed.), "Active Rules for Databases," Springer, New York, 1999.

[14] ACT-NET Consortium, "The Active DBMS Manifesto," ACM SIGMOD Record, 25(3), 1996.

[15] M. Garcia-Valls, P. Basanta-Val, M. Marcos, and E. Estévez, "A bi-dimensional QoS model for SOA and real-time middleware," in: Intl. Journal of Computer Systems Science and Engineering, CLR Publishing, 2013, pp. 315-326.

[16] V. Krishnamurthy and C. Babu, "Pattern Based Adaptation for Service Oriented Applications," in: ACM SIGSOFT Softw. Eng. Notes 37, 2012(1), 2012, pp. 1-6.

[17] T. Frotscher, G. Starke (ed.), and S. Tilkov (ed.), ”Der Webservices-Architekturstack,” in: SOA-Expertenwissen, Heidelberg, dpunkt.verlag, 2007, pp. 489-506.

[18] F. Curbera, R. Khalaf, and N. Mukhi, ”Quality of Service in SOA Environments. An Overview and Research Agenda,” in: it - Information Technology 50, 2008(2), 2008, pp. 99-107.

[19] G. Wang, A. Chen, C. Wang, C. Fung, and S. Uczekaj, ”Integrated Quality of Service (QoS) Management in Service-Oriented Enterprise Architectures,” in: Proceedings of the 8th IEEE Intl. Enterprise Distributed Object Computing Conference (EDOC’04), Washington DC, USA, IEEE, 2004, pp. 21-32.

[20] S.W. Choi, J.S. Her, and S.D. Kim, ”QoS Metrics for Evaluating Services from the Perspective of Service Providers,” in: Proc. of the IEEE International Conference on e-Business Engineering, Washington DC, USA : IEEE Computer Society (ICEBE’07), 2007, pp. 622-625.

[21] M. Varela, L. Skorin-Kapov, F. Guyard, and M. Fiedler, ”Meta-Modeling QoE”, PIK-Praxis der Informationsverarbeitung und Kommunikation, 2014, Vol. 37(4), pp. 265-274.

[22] Z. Balfagih and M.F. Hassan, ”Quality Model for Web Services from Multi-stakeholders’ Perspective,” in: Proceedings of the 2009 International Conference on Information Management and Engineering, Washington DC, USA : IEEE Computer Society (ICIME’09), 2009, pp. 287-291.

[23] R.W. Maule and W.C. Lewis, ”Performance and QoS in Service-Based Systems”, Proc. of the 2011 IEEE World Congress on Services, IEEE Computer Society, 2011, pp. 556-563.

[24] F. Rosenberg, C. Platzer, and S. Dustdar, ”Bootstrapping Performance and Dependability Attributes of Web Services,” in: Proc. International Conference on Web Services (ICWS’06), 2006, pp. 205-212.

[25] M. Schmid, J. Schaefer, and R. Kroeger, ”Ein MDSD-Ansatz zum QoS-Monitoring von Diensten in Serviceorientierten Architekturen,” in: PIK Praxis der Informationsverarbeitung und Kommunikation, 31 (2008) 4, 2008, pp. 232-238.

[26] B. Wetzstein et al., ”Monitoring and Analyzing Influential Factors of Business Process Performance,” in: Proc. IEEE Intl. Enterprise Distributed Object Computing Conf. (EDOC’09), 2009, pp. 141-150.

[27] S.M.S. da Cruz, R.M. Costa, M. Manhaes, and J. Zavaleta, ”Monitoring SOA-based Applications with Business Provenance”, Proc. of the 28th Annual ACM Symposium on Applied Computing (ACM SAC), ACM, 2013, pp. 1927-1932.

[28] ISO - International Organization for Standardization (ed.), ”ISO/IEC 25010:2011 - Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models”, 2011.

[29] M. Azuma, ”SQuaRE: the next generation of the ISO/IEC 9126 and 14598 international standards series on software product quality, ” in: Proc. of European Software Control and Metrics (ESCOM), 2001, pp. 337-346.

[30] H. Balzert, ”Lehrbuch der Softwaretechnik: Softwaremanagement”, Springer Spektrum, Heidelberg, 2008.

[31] DIN NIA, ”ISO/IEC 25000 System und Software-Engineering - Qualitätskriterien und Bewertung von System- und Softwareprodukten (SQuaRE - Leitfaden für SQuaRE,” [Online]. URL: http://www.din.de/de/mitwirken/normenausschuesse/nia/normen/wdc-beuth:din21:204260933 [accessed: 2017-01-01].

[32] F. Garcia et al., ”Towards a consistent terminology for software measurement,” in: Information and Software Technology, vol. 48, 2006, No. 8, pp. 631-644.

[33] ISO - International Organization for Standardization (ed.), ”ISO/IEC 15939:2007 - Systems and software engineering - Measurement process,” 2007.

[34] A. Wahl, A. Al-Moayed, and B. Hollunder, ”An Architecture to Measure QoS Compliance in SOA Infrastructures,” Service Computation, 2010, pp. 27-33.

[35] E. Oberortner, S. Sobernig, U. Zdun, and S. Dustdar, ”Monitoring Performance-Related QoS Properties in Service-Oriented Systems: A Pattern-Based Architectural Decision Model,” EuroPLoP, 2011, pp. 1-37.

[36] E. Oberortner, U. Zdun, and S. Dustdar, ”Patterns for Measuring Performance-related QoS Properties in Service-oriented Systems,” Pattern Languages of Programs Conference, 2010, pp. 1-21.

[37] L. Lavazza, S. Morasca and D. Tosi, ”Enriching Specifications to Represent Quality in Web Services in a Comprehensive Way,” IEEE Symposium on Service-Oriented System Engineering, 2015, pp. 203-208.