Definition of Metric Dependencies for Monitoring the Impact of Quality of Services on Quality of Processes

Christian Mayerl, Kai Moritz Hüner, Jens-Uwe Gaspar, Christof Momm, Sebastian Abeck
Cooperation & Management, Institute of Telematics
Faculty of Computer Science, Universität Karlsruhe (TH)
76128 Karlsruhe, Germany
Email:{mayerl | huener | gaspar | momm | abeck}@cm-tm.uka.de

***Abstract*— Service providers have to monitor the quality of offered services and to ensure the compliance of service levels provider and requester agreed on. Thereby, a service provider should notify a service requester about violations of service level agreements (SLAs). Furthermore, the provider should point to impacts on affected processes in which services are invoked. For that purpose, a model is needed to define dependencies between quality of processes and quality of invoked services. In order to measure quality of services and to estimate impacts on the quality of processes, we focus on measurable metrics related to functional elements of processes, services as well as components implementing services. Based on functional dependencies between processes and services of a service-oriented architecture (SOA), we define metric dependencies for monitoring the impact of quality of invoked services on quality of affected processes. In this paper we discuss how to derive metric dependency definitions from functional dependencies by applying dependency patterns, and how to map metric and metric dependency definitions to an appropriate monitoring architecture.**

I. Introduction

A service provider has to monitor the quality of provided services and to ensure the compliance of service level agreements (SLAs). In this context, not only functional dependencies have to be considered, but also dependencies between quality of service (QoS) and quality of process (QoP). At runtime, a provider should notify service requesters about SLA violations and point to impacts on affected processes. For example, the impact of service performance on affected processes is of special interest [1]. In order to monitor QoS, we focus on metrics measurable at functional elements of a service or at components implementing services. To calculate or estimate impacts on QoP a sufficient understanding of dependencies between service metrics and process metrics is needed.

For that purpose, Fig. 1 illustrates our work. Models defining functional dependencies between processes and invoked services exist like in a service-oriented architecture (SOA). Models in Fig. 1.a and 1.b describe these dependencies on different abstraction levels. Furthermore, models which define dedicated metrics related to processes, services and components implementing these services exist, too (see Fig. 1.b to 1.c).

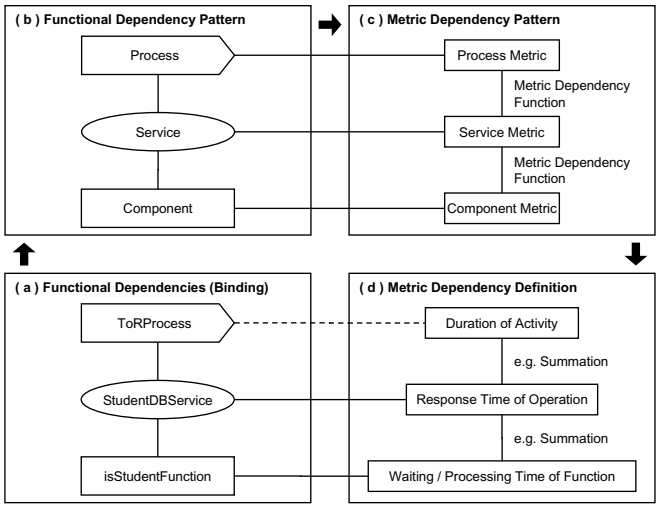

Fig. 1. Definition of Metric Dependencies

But a sufficient definition of metric dependencies is missing. For that reason, we discuss the following questions in this paper:

- Which elements are needed to define dependencies between metrics related to processes, metrics related to invoked services and metrics related to components implementing these services (see Fig. 1.c to 1.d)?
- How have these metric dependencies to be defined to allow an efficient mapping of metric dependency definitions to components of an appropriate monitoring architecture?

Therefore, this paper is organized as follows: Section II gives an overview of the background of our work and discusses related work. Section III introduces a model to define metric dependencies derived from functional dependencies. Section IV describes how defined metric dependencies can be mapped to components of a monitoring architecture. Section V tells about some experiences gained by implementing a demonstrator. The paper ends with a short conclusion and an outlook of further work in Section VI.

II. Background and Related Work

Business-driven metric dependencies [2] between processes and invoked services can be investigated at a business level, at which service providers have to prove their contributions to business, or at a system level, at which executable processes invoke services implemented by application components. Because composing and invoking services according to business processes is an important challenge in the context of service-oriented architectures (SOAs), we focus on metric dependencies at the system level. These kinds of dependencies can then be the bases for measurements of metric dependencies at a business level [3].

A. Background

As an example, we investigate an application environment of a university. These applications provide services to support a mobile studying in Europe according to the Bologna process. In this scenario, a lot of students for example request information about the current status of their study or about course offerings of future semesters to plan their next steps. Peaks of requests can be recognized especially at the beginning or at the end of semesters. For each request, business processes are initiated by the university to gather and provide necessary information for students. One example is the process of generating a transcript of records (ToR). This is accomplished by invoking several services based on applications which implement functions to manage student, course and examination data [4].

To ensure an acceptable performance [5] of generating ToRs, it is necessary to consider the metric dependencies between processes and invoked services and to monitor the impact of QoS on QoP. Basically, a business process is defined as a sequence of activities. These activities can be executed by human beings or can be executed automatically. In our case, we assume that the ToR generation process is fully automated and can be implemented using, for example, the business process execution language (BPEL) [6]. Invoked services are implemented by applications managing student, course and examination data. The invocation of application-based services is done via the web. Therefore, services have a uniform resource locator and a set of interfaces that can be utilized to access them. Related to service functionality, provider and requester agree on quality of service which is documented as a service level agreement (SLA) [7].

At runtime, a service provider needs a management environment to ensure the quality of provided services, to monitor the SLA compliance and to notify service requesters in case of failures. This environment should allow the measurement of metrics of services and to calculate the impact on quality of processes in which interfered services are invoked. In order to implement necessary management functions and to integrate them with existing management solutions we follow established standards like the web-based enterprise management (WBEM) initiative [8] and the common information model (CIM) standardized by the distributed management task force (DMTF). CIM defines management information for modeling computing and business entities, enterprise and service provider environments. It especially contains a sub model for defining metrics [9].

B. Related Work

Defining and monitoring QoS and impacts on QoP, the QoS UML profile [5] offers a basic terminology and structure to model quality of services and fault tolerance. For a structured model and a common understanding of QoS, it defines terms "category", "characteristic", "dimension", etc. As an example the QoS category "performance" is detailed in this work. Even though the UML profile consolidates previous work on QoS languages and notations, model elements for defining calculable dependencies between QoS characteristics are not sufficiently covered.

In order to ensure response time of transactions, [10] presents an approach for real-time load-balancing. Therefore, an analytical QoS model via different abstraction levels is defined: Services, transactions and nodes. With reference especially to transactions, two types of metrics or rather measurements are distinguished: Raw metrics (e.g. registration time, failed time, start and stop times) and aggregated metrics (e.g. down time, waiting time, failed count, service time, response time, violation rate, etc). For an efficient control mechanism, automated actions are differed from non-automated actions. This helps to ensure the quality of services. But for calculating the impact on processes and business, the qualitative dependencies between services and processes have to be monitored, too.

[11] presents a platform to define, compute and analyze business and IT metrics in relation to e-business processes based on Web services. Therefore, a basic model for functional dependencies between business processes, activities, services, end-points, operations, and messages is defined in UML. Additionally, the model contains "metrics" to be computed and "mappings" that define how operational data can be mapped into qualitative and quantitative measures. Furthermore, mappings are specified by instantiating mapping templates, and "meters" are the instruments to compute metrics. A metric database is structured according to data warehousing techniques enabling and simplifying multi-dimensional analysis of quality measures. Based on such a model of functional and metric dependencies between processes and services, management functions are needed to monitor QoS and also to calculate the impact on QoP.

[12] presents an approach for automated and distributed monitoring of SLAs. Defined metrics relate to processes (according to WSFL) and services (according to WSDL). Metric examples are response time, availability, security, cost, etc. Each service level objective (SLO) comprises a functional part (that refers to a system, endpoint, a process, or a set of processes) and a guarantee part applied on the functional part. SLA compliance can be monitored on service provider side and on service requester side. When the evaluation of an SLA depends on measurements from both sides, a measurement protocol is used for transferring measurements from requester

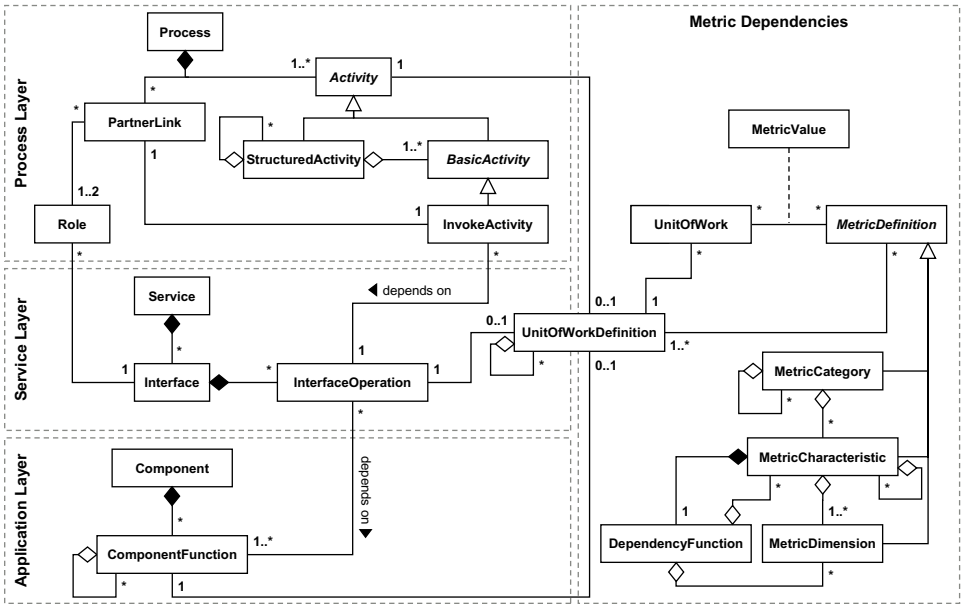

Fig. 2. Class Diagram Defining Metrics and Metric Dependencies

to provider. This protocol allows monitoring of dependencies between services and processes in which services are invoked. A method to define calculable QoS and QoP dependency instructions as part of quality models and to map them to the monitoring components is still missing.

[13]–[15] developed components and data structures for creating and monitoring SLAs in relation with web services. For the integration of the developed components with established management approaches, [13] defines a mapping between SLAs and CIM. Although, these components are needed for monitoring SLA compliance, a method for modeling and defining calculable metric dependencies is out of scope.

[16] specifies dependencies between a number of system resources and the business impact. The aim is to optimize SLOs from a business perspective. In order to calculate the business impact a mathematical model is used. Even though the followed theories and formalisms can help to model properties of system and application-based services, it is not the aim of [16] to monitor the modeled dependencies at runtime.

III. Modeling Metric Dependencies

In order to calculate or estimate the impact of QoS on QoP, a formal model is needed specifying metric dependencies between services and processes. Within a SOA, several functional dependencies exist (e.g. between a process activity and an invoked service operation [12], [17]) and can be used as a basis to identify additional dependencies. Fig. 2 shows a model of these dependencies based on concepts of well known standards like BPEL [6] and WSDL [18]. The model only describes those concepts of standards which are important to model functional dependencies between process activities, service operations and service implementing component functions within a SOA. During modeling metric dependencies of a specific architecture, the corresponding functional dependencies are assumed to be already described since their identification is considered as a task within the software development process or a binding process.

As depicted in Fig. 2, services expose interfaces with operations. These service operations can be invoked by process activities, which are linked in the context of a business process. In this work, only invoking activities with a request-response interaction pattern are focused. We assume that application-based services are implemented by functions of application components.

For modeling metric dependencies between services and processes, we focus on metrics which can be measured or calculated at runtime. Examples for such metrics are response time, availability and performance [10], [19], [20].

A. Specifying Metric Definitions

In addition to elements and interrelations to model functional dependencies, the model shown in Fig. 2 contains concepts to define metrics and their dependencies as well. Additional elements to define metrics and metric dependencies are shown on the right side of Fig. 2 and are based on concepts of the CIM metrics model [9] and the UML profile described in [5].

As discussed in [11], the model defines elements to specify a metric. Their semantics are explained below.

- A value of a metric dimension is received directly from the instrumentation of a respective functional entity.
- A metric characteristic groups several dimensions or other characteristics. A value of a characteristic is the result of a computation: Values of corresponding characteristics are computed recursively and then combined with corresponding dimension values. The instruction how to

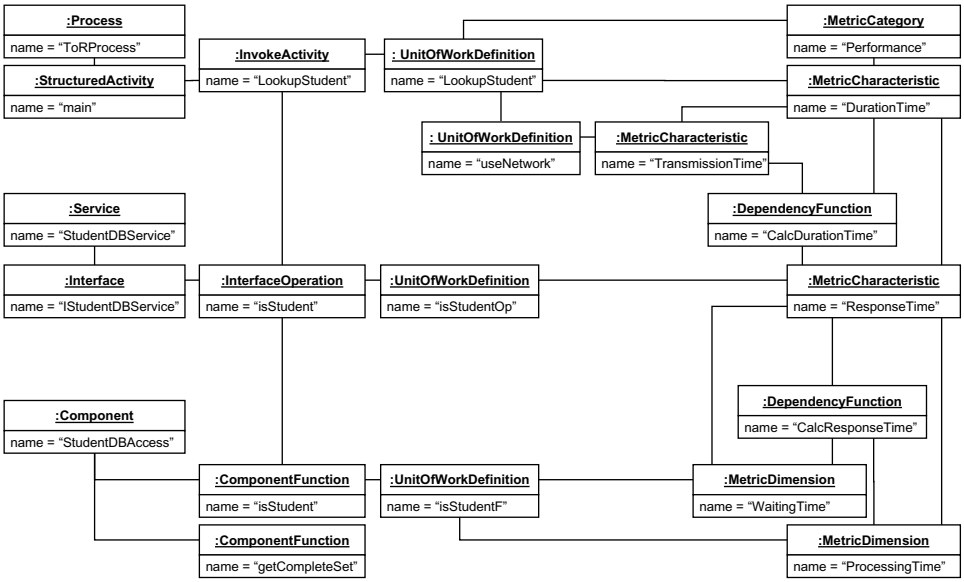

Fig. 3. Model for StudentDBService and ToRProcess

combine the different values is given by a dependency function associated with the characteristic.

- A metric category is used to group several characteristics or other categories. An example is given by the metric category "performance" which includes characteristics like "availability" or "response time". These characteristics are well known as performance metrics [10], [19], [20], whereas a universal calculation of a single performance value is not given.
- A dependency function is associated to a characteristic. By this concept, instructions are given how to calculate the characteristic value based on values of other related characteristics and dimensions. In Section III-B, a formal model is specified in order to enable a mathematical description of these calculation instructions.

At this point, functional dependencies as well as dependencies between different metrics can be described by the model shown in Fig. 2. In order to define metric dependencies, these two aspects have to be combined by relating metrics to functional entities. The binding is realized by defining units of work based on the equivalent concept of the CIM metrics model [9]. Semantics of this concept are described by an example in the following paragraph.

The objects of Fig. 3 instantiate classes defined in Fig. 2. The objects and links describe coherences of the introduced ToR process example. Since the class diagram describes conceptual model elements and interrelations on a meta level, the instances represent elements of the ToR process example but not instances at runtime. The process activity `LookupStudent` uses the service operation `isStudent`, which uses a component function `isStudent`. Correlated links between these functional entities represent functional dependencies.

In the described example, we want to monitor the impact of the response time of the service operation `isStudent` on the duration time of the process activity `LookupStudent` (see also [3]). For this reason, a response time metric related to the `isStudent` service operation is defined. For identifying bottlenecks in the implementation of the service, metrics for components are defined as well: A metric for waiting time measurement and processing time measurement respectively. The instances of `DependencyFunction` encapsulate information about how to calculate metric values based on values of correlated metrics as stated above. These dependency functions are formalized in the following section.

B. Formalizing Metric Dependencies

We focus on calculable metrics in this work. According to Section III-A and Fig. 2, these calculable metrics are called "metric characteristics" and are associated with a dependency function. This function includes information about how to calculate a metric value based on other metric values. In this section, we want to introduce a formalism in order to describe dependency functions and metric characteristics in a mathematical way. The purpose is to calculate metric values based on well-defined functions (see Fig. 1.c and 1.d).

Formally, in the context of quality measurement, we have a set E of functional entities (instances of `UnitOfWork` shown in Fig. 2) and a set Q of values which are measured for these functional entities. Examples for an element $q \in Q$ are "5 seconds" or "$99\,\%$". Later on, these elements of Q are used to define "metric values" according to Fig. 2. To distinguish between metric values and single elements of Q which are not yet related to a metric, an element $q \in Q$ is named "quality value".

By measuring a quality value $q \in Q$ for a specific functional entity $e \in E$, we receive a pair $(e, q) \in E \times Q$. In addition to E and Q, we define the set M of all metrics. A metric $m \in M$ can be formalized as a function which maps a quality

value $q \in Q$ to a functional entity $e \in E$. In order to get sets which can be used as domain and range of this function, we have to define subsets of E and Q.

For a specific metric $m \in M$, not all elements of Q are suitable (e.g. "99 %" as value for the response time metric $m_{\text{responseT}} \in M$ makes no sense, so the set $Q|_{m_{\text{responseT}}}$ would contain only time values of Q). In conclusion, we can say that a metric $m \in M$ defines a subset $Q|_m \subseteq Q$ of those elements of Q which are suitable for m. In this context, "suitable" means, for example, that an element $q \in Q|_m$ has the data type defined for m and an appropriate unit.

If a metric $m \in M$ defines a subset of Q, then m also defines a subset of $E \times Q$. This subset is defined below.

$$E \times Q|_m := \{(e, q) \in E \times Q : q \in Q|_m\}$$

In the same way, we can define a subset $E|_m \subseteq E$ in order to describe those functional entities a specific metric $m \in M$ is measured at. With $Q|_m$ and $E|_m$, we can define a metric $m \in M$ as a function $m : E|_m \rightarrow Q|_m$. At this point, we have described the `MetricValue` of Fig. 2 and the elements of $E|_m \times Q|_m$ can be called "metric values". As conclusion of these formal definitions, we can say that a metric $m \in M$ defines a subset $E|_m \times Q|_m \subseteq E \times Q$ which is called "the set of metric values for metric m".

In order to specify formal instructions for calculating the impact of QoS on QoP, we have to formalize dependencies as well. For the following definitions, we assume to have a process activity $p \in P$, a service operation $s \in S$ and a component function $c \in C$ with $P \subseteq E$, $S \subseteq E$ and $C \subseteq E$. Relations of these functional entities are also shown in Fig. 2. For a specific metric $m \in M$, we use an index to express the semantic of the metric (e.g. $m_{\text{responseT}}$ for a response time metric). Dependencies of used metrics also are shown in Fig. 4. According to the previous sections, we define two types of dependencies in the notation of logical predicates.

- Functional dependencies between entities: The predicate $d_E(p, s)$ means, that the functionality of p depends on the functionality provided by s.
- Dependencies between metrics: $d_M(m_{\text{durationT}}, m_{\text{responseT}})$ expresses the necessity of a response time value for the calculation of a duration time value.

For the definition of a calculable dependency function as shown below, both functional and metric dependencies have to be considered. Given a set of k metric dependencies $d_M(m_{q_0}, m_{q_1}), \ldots, d_M(m_{q_0}, m_{q_k})$ and a set of n functional dependencies $d_E(e_0, e_1), \ldots, d_E(e_0, e_n)$, a metric m_{q_0} for an entity e_0 can be calculated by using a dependency function $f_{\text{calc}} : Q^{k \cdot n} \rightarrow Q$. This function f_{calc} corresponds to the dependency function shown in Fig. 2.

$$\begin{aligned} m_{q_0}(e_0) = f_{\text{calc}}\big(&m_{q_1}(e_1), \ldots, m_{q_k}(e_1), \\ &m_{q_1}(e_2), \ldots, m_{q_k}(e_2), \ldots, \\ &m_{q_1}(e_n), \ldots, m_{q_k}(e_n)\big) \end{aligned}$$

Given a dependency function f_{calc} and a functional entity e_0, the value of a metric m_{q_0} can be calculated in the shown way based on recursively calculated other values of metrics $m_{q_1}, \ldots, m_{q_k}$. This dependency function can be represented as an abstract syntax tree (AST), where the internal nodes are labeled by the operators of the function and the leave nodes represent corresponding metric dimensions. If a value of a metric $m \in M$ is not calculable but provided directly by an instrumentation of a functional entity e (this case is described in Fig. 2 by a metric dimension), the notation $m(e)$ could also be used.

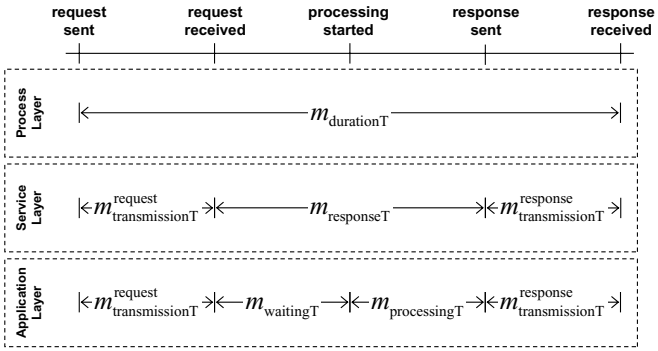

Fig. 4. Metric Dependency Pattern for Duration of Activities

As an example, Fig. 4 shows a metric $m_{\text{durationT}}$ to measure the duration of a process activity. As well, dependencies to other metrics (e.g. $d_M(m_{\text{durationT}}, m_{\text{responseT}})$) are shown and several metrics are assigned to abstract elements of a service-oriented architecture (SOA). Information about specified metric dependencies is based on [1], [19], [20]. In this work, a metric definition including metric dependencies and assignments to abstract architectural components is called a "metric dependency pattern". According to Fig. 1, such a pattern can be applied to a specific SOA by assigning a concrete architectural component (Fig. 1.a) to each metric of the pattern (Fig. 1.d).

In the following example, the metric $m_{\text{durationT}}$ is calculated for the activity `LookupStudent` of the ToR scenario described in Section II-A. Related dependencies also are shown in Fig. 3 and are formalized below.

$$\begin{aligned} &d_E(p_{\text{LookupStudent}}, s_{\text{isStudent}}) && d_E(s_{\text{isStudent}}, c_{\text{isStudent}}) \\ &d_M(m_{\text{durationT}}, m_{\text{transmissionT}}) && d_M(m_{\text{durationT}}, m_{\text{responseT}}) \\ &d_M(m_{\text{responseT}}, m_{\text{waitingT}}) && d_M(m_{\text{responseT}}, m_{\text{processiongT}}) \end{aligned}$$

The measurement of transmission time (time elapsing while a message is sent over a network) in this context is a special case. This metric can not be measured at a single functional entity but needs information of both the sending and the receiving entity [21]. In such a case, the receiving entity is used in the formalism assuming that the start time could be sent within the message. The transmission time of the response message can not be measured in this way. In this work, we

estimate this time based on the request transmission time.

$$m_{\text{durationT}}(p_{\text{LookupStudent}}) = f_{\text{timeAdd}}\left(m^{\text{request}}_{\text{transmissionT}}(s_{\text{isStudent}}), m_{\text{responseT}}(s_{\text{isStudent}}), m^{\text{response}}_{\text{transmissionT}}(p_{\text{LookupStudent}})\right)$$

$$m_{\text{responseT}}(s_{\text{isStudent}}) = f_{\text{timeAdd}}\left(m_{\text{waitingT}}(c_{\text{isStudent}}), m_{\text{processingT}}(c_{\text{isStudent}})\right)$$

$$f_{\text{timeAdd}}(t_1, \ldots, t_n) = \sum_{i=1}^{n} t_i$$

The dependency function f_{timeAdd} used in the example above has the following form: All metric values are assumed to have the same unit and the computation is just an addition of all values. In the example, we used a rather simple dependency function to illustrate how it is defined. In future work, the formalism can be used to define more complex dependencies.

IV. Mapping Models to Monitoring Architecture

To monitor the impact of SLA violations, techniques and mechanisms are necessary to support the monitoring of dependencies between application-based services and affected processes. Monitoring the impact of QoS on QoP at runtime, we start from the following assumptions: In our case, metrics are only measured at the service provider, as it is not possible to get information about the service requester's system. At the provider's side service-oriented metrics either can be measured directly at a service interface (e.g. instrumented at the message handler of a SOAP engine), or can be calculated based on metrics measured at components which implement a service. Impacts of QoS on quality of processes which invoke services are calculated or estimated according to formalized instructions as described in Section III. Non measurable but necessary metrics lead to assumptions. For example, we assume that the transmission time of a request message is equal to the transmission time of the corresponding response message.

It has to be considered that only those metrics can be measured which are offered by management capabilities of either service interfaces or application components. Consequently, an adequate monitoring architecture contains functions to instrument and collect metrics as well as functions to aggregate and compare metrics with agreed service levels. To notify impacts on processes of the requester and report SLA compliance, additional functions are needed. An overview of the monitoring architecture and its components is given in the following sections.

A. Instrumenting and Collecting Metrics

[19] discusses metrics to be measured as qualitative properties related to web services. In this work, we focus on performance impacts of services on processes in a service-oriented architecture (SOA).

A `Web service Implementation` as shown in Fig. 5 exposes its functionality via a service interface described by WSDL [18]. Metrics related to services can either be measured at components implementing services or at the service interface itself. Where to measure a specific metric depends on the metric as well as the component instrumentation and the `Application Server`, the service is deployed on. As defined in Section III-A, a directly measured metric is called metric dimension.

Dimension values can either be sent to the `Metric Collector` or can be requested by the `Metric Collector`. Particular dimension values are provided at management ports of either the `Web service Implementation` or the `Application Server`. To measure dimensions at the service interface, the `Application Server` (more precisely the corresponding message handler) has to be instrumented. Thus, interaction and message transfer between service provider and requester can be observed. In any case, it should be possible to correlate a measured dimension value with a unit of work which can be an invoked component function, service operation or process activity. To do that, instrumentations have to provide context information to allow correlating metric values with units of work (see also [21]).

Another task of the `Metric Collector` is the transformation of dimension values into instances that are compliant with management information standards (e.g. CIM [9]). Converted values are directed to the `Storage`. The main task of that component is to archive measured dimension values and to associate these values with functional and metric entities. It is exposing an interface to read, write and notify event-based data that conforms to the information model specified for a particular scenario.

B. Aggregating Metrics and Calculating Impacts

Retrieved and stored dimension values are supposed to be processed in the `Agreement Monitor`. The main tasks of this monitor are:

- Aggregating service-oriented metrics
- Calculating impacts on processes
- Notifying and reporting impacts

Therefore, the `Agreement Monitor` consists of three subcomponents described below.

The `Metric Aggregator` is a passive component which is triggered either by requests from the `Metric Comparator` or by requests from the `Report Generator`. The task of the `Metric Aggregator` is to calculate metric values based on functional and metric dependencies. In the following, we say "characteristic" for a calculable metric according to Section III-A. In order to explain the functionality of the `Metric Aggregator`, the introduced ToR scenario is used. In the example, the `Metric Aggregator` is asked to provide a value of the process metric "duration time" for a specific unit of work related to the `LookupStudent` activity. Relevant dependencies are shown in Fig. 3 and Fig. 4.

In the example, the input for the `Metric Aggregator` is the ID for the duration time metric and the ID of the relevant unit of work. We assume that at runtime the `Metric Aggregator` knows about functional and metric dependencies. For example, this information can be stored in the

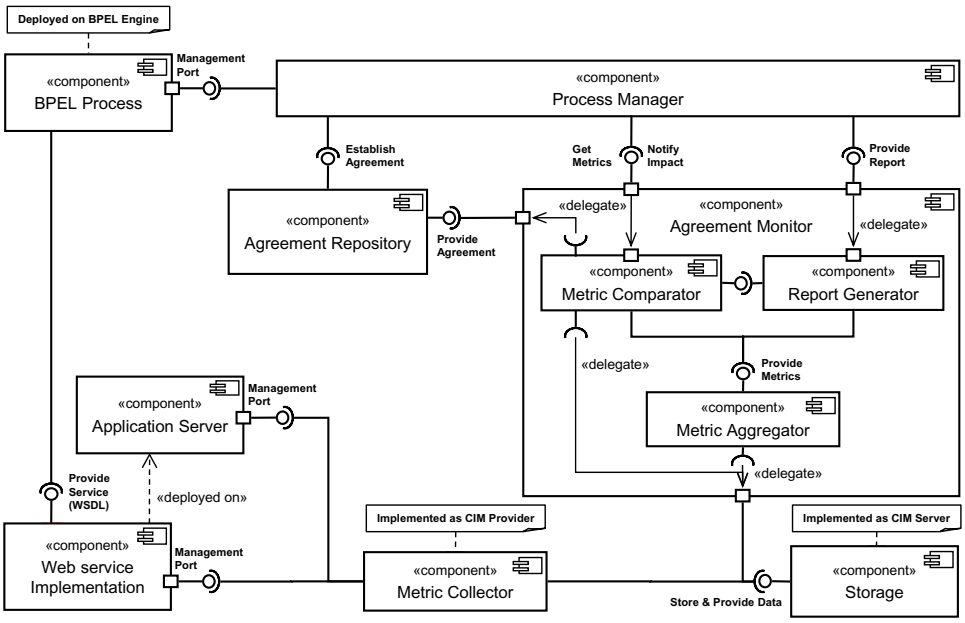

Fig. 5. Monitoring Architecture

`Storage` and is loaded during the initialization of the aggregator since it is invariant at runtime. According to Fig. 3 and Fig. 4, the process characteristic `DurationTime` is calculated based on the dimension `TransmissionTime` (here we do not distinguish between transmission times for a request message and a response message in order to simplify the example) and a service characteristic `ResponseTime` which is based on the component dimensions `ProcessingTime` and `WaitingTime`. In conclusion, the `Agreement Monitor` knows which dimensions are relevant to calculate the duration time of the `LookupStudent` activity by traversing the related metric dependencies. By traversing as well the functional dependencies related to the relevant unit of work (that means the reflexive aggregation of `UnitOfWorkDefinition` in Fig. 2), the `Metric Aggregator` can determine those units of work which are related to the relevant metric dimensions. Based on this knowledge, the `Storage` is asked for the relevant dimension values which are aggregated based on the information of the respective `DependencyFunction`.

Another requirement of the `Metric Aggregator` also is to keep track, if all input arguments are available to start the calculation of the targeted aggregated metric. Therefore, the aggregator may also write some temporary or intermediate calculated metrics back into the `Storage` or use some internal tracking. Another solution would be the utilization of data warehouse technology [11].

The `Metric Comparator` compares metric values defined as SLOs in SLAs with aggregated service-oriented metric values. Before the comparison can start, SLAs have to be negotiated and stored within the `Agreement Repository`. For that reason, the repository exposes interfaces as proposed in WS-Agreement [7] to establish and provide an agreement. Because the negotiation of SLAs is not in the focus of this work, not all interfaces described in [7] are shown in Fig. 5. During the validity period of an SLA the `Metric Comparator` can act in three different ways:

- Active role: The `Metric Comparator` can continuously check the compliance of a specific SLA constraint. For example, based on historical data the last 100 units of work related to a specific activity could be checked.
- Passive role triggered by the `Storage`: When a unit of work related to a specific activity under observation is started, an event-based message is created by the `Storage` which triggers the `Metric Comparator`. The event message includes the relevant unit of work ID and so the `Metric Comparator` can start calculation as described in the example above.
- Passive role triggered by a request to the `GetMetrics` interface: Via the `GetMetrics` interface of the `Agreement Monitor`, a unit of work ID and a metric ID can be sent as well. Based on historical data, the related value can be calculated and sent back to the `Process Manager`.

In order to estimate the impact of quality of invoked services on quality of affected processes, we assume context information included in messages containing at least an identifier that serves as reference to involved entities. Without this information, measurement data can not be assigned to the associated entities. Solutions to this problem are discussed in [21].

Next to the `Metric Comparator`, the `Report Generator` summarizes monitoring information for generating a regular report about SLA compliances.

V. Implementation Experiences

This section reflects experiences about the definition of metric dependencies as presented in Section III and the mapping of metric dependencies to an appropriate monitoring architecture as described in Section IV. As proof of concept, the monitoring architecture is implemented and the ToR generating process described in Section II A is used to evaluate the presented approach.

A. Demonstrator

The ToR generating process is implemented using BPEL [6] and is deployed on the Oracle BPEL Engine. The BPEL process invokes the service operation `isStudent` of the SOAP-based web service `StudentDBService` described by WSDL [18]. The components of the scenario are also depicted in Fig. 3. The `StudentDBService` exposing the `isStudent` operation is provided by the Apache Axis2 SOAP engine. The SOAP engine runs embedded in an Apache Tomcat 5.5 servlet container which is used as `Application Server` according to Fig. 5. The binding between the ToR process and the service could be done statically or dynamically. In our case, we assume that the binding is specified statically during the development of the ToR generating process.

For the measurement of metrics, there are several technologies to assist the development of a management interface. Examples are the Web Service Distributed Management (WSDM) or WS-Management. But neither WSDM nor WS-Management give sufficient support for instrumentation. Since application components implementing the `StudentDBService` are implemented in Java, the Java Management Extension (JMX) has been considered. JMX defines an architecture to instrument resources but it contains no management information model. Since we are focused on metrics related to response time, the Application Response Measurement API (ARM) standardized by The Open Group [22] fits in well. ARM contains features like identifying bottlenecks within software components, measuring application availability and performance as well as end-to-end transaction response time. To implement ARM, we used the OpenARM toolkit. Its instrumentation calls have been directly included into the Java methods implementing the `StudentDBService`. For the instrumentation of the service interface, we extended the Apache Axis SOAP engine with a SOAP handler [23] which counts service requests and tracks SOAP messages to determine response time according to ARM.

In order to evaluate our approach, we implemented several components of the monitoring architecture presented in Fig. 5. There are alternative approaches to implement management information and their relations according to metric dependencies. In [11] for example, a data warehouse is used. [24] presents an approach to integrate different management applications and information according to management processes.

In the current implementation, we decided to use technologies which are conforming to the standardized common information model (CIM) as part of the web-based enterprise management (WBEM) initiative [8]. Therefore, the CIM metrics model [9], [13] was related to functional dependencies of the ToR generating process and the `StudentDBService` as defined in Section III. Additionally, metric dependencies as well as dependency functions have been extended. Applying CIM to the monitoring architecture, the `Metric Collector` has been implemented as a CIM provider and the `Storage` as a CIM server. The `Metric Collector` transforms metric values into CIM-conform instances of CIM objects and forwards those to the CIM server in order to make them persistent and ready to be used by the subcomponents of the `Agreement Monitor`. The CIM server can be asked for `MetricValue` instances associated with `UnitOfWork` instances and corresponding `MetricDimension` instances. As described in Section IV-B, metric values can be retrieved from the CIM server, given a unit of work and a metric dimension.

After the binding between `ToR process` and `StudentDBService`, functional dependencies are well known and represented in the `Storage` as instances of `UnitOfWorkDefinition` and corresponding associations. For example, a functional dependency between the process activity `LookupStudent` and the service operation `isStudent` results in an association instance which describes a sub relation between corresponding units of work (the relevant associations of the CIM metrics model are `SubUoWDef` and `SubUoW` respectively). In addition to the representation of functional dependencies in the `Storage` component, metric dependencies are included as well. For that purpose, an extension of the CIM metrics model is used according to the model elements shown in Fig. 2.

To enable a process manager to decide whether an instance of a process is affected by an SLA violation or not, the impact of a service instance on an instance of a process activity has to be calculated. For that reason, we included information about interaction context into the SOAP header as proposed as well in [21]. This context information allows relating a metric value with a unit of work instance. Aggregating metrics, this context is used to relate metrics with the same context to one service instance and to calculate the impact on a corresponding process activity instance.

B. Evaluation

In Section IV, a monitoring architecture is presented which enables monitoring the impact of QoS on QoP. This architecture uses the metric dependency model described in Section III. As stated in Section V-A, we decided to use a CIM server to store measured metric values as well as dependencies. The concrete implementation of our CIM server is based on code of the WBEM Services project [25].

In Section IV-B, three possible roles of the `Metric Comparator` are described. For the second one, the `Metric Comparator` is triggered by the `Storage`. In this case, the `Metric Comparator` is supposed to monitor one specific service invocation: When the relevant service operation is called, the `Metric Collector` creates a correspond-

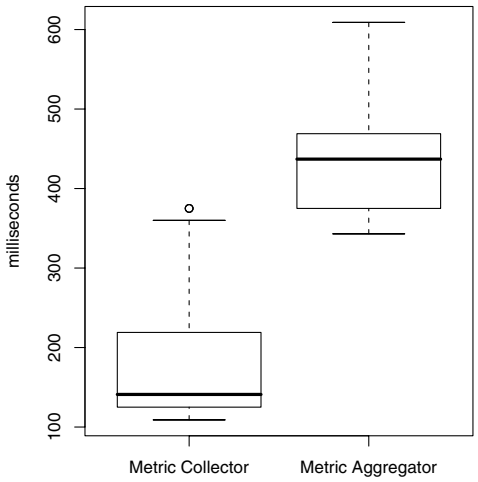

Fig. 6. Evaluation Results

ing `UnitOfWork` instance and the `Storage` informs the `Metric Comparator` which starts monitoring the SLA compliance for this instance. Of course, creation of the relevant `UnitOfWork` instances, instantiation of `MetricValue` associations as well as calculation of aggregated metrics has to be done within the duration of the service call. That means for this scenario, all communication between `Metric Comparator` and `Storage` as well as between `Metric Aggregator` and `Storage` is time critical. A lower bound for the response time of a service operation, which must not be undercut in order to enable monitoring of a specific service operation call, could be defined. For that reason, durations for the critical tasks have to be evaluated.

In Fig. 6, the evaluation result for two time critical tasks is shown. It has to be noticed, that the focus of this work is not on performance aspects of the presented management architecture. Therefore, the evaluation results in this section are not supposed to show good or bad performance of our implementation but give an impression about its behavior at runtime. The following two components have been used for the evaluation:

- The `Metric Collector` has been used to create a unit of work and to associate a metric value.
- The `Metric Aggregator` has been used to calculate the value of the metric characteristic "duration time" shown in Fig. 3 und Fig. 4.

Both tasks have been run 10.000 times on a virtualized system with a clock rate of 1 GHz and 512 MB RAM. In Fig. 6, the measured duration times are shown. We decided to use box plots in order to illustrate not only absolute values but also the variance of the measurement. A box plot depicts the lower quartile (box start), the median (bold line), and the upper quartile (box end). The horizontal lines ("whiskers") extend to at most 1.5 times the box width and must end at an observed value. If a value is beyond a whisker, it is depict as an additional point.

The medians shown in Fig. 6 are 141 milliseconds (Metric Collector) and 437 milliseconds (Metric Aggregator) respectively.

VI. Conclusion and Outlook

In this paper we have investigated a method how to derive metric dependencies from functional dependencies between services and processes in which services are invoked. The intention is to use calculable metric dependencies for monitoring the impact of QoS on QoP.

A. Conclusion

For that purpose, a model has been introduced defining functional dependencies between elements of a service-oriented architecture (SOA). Thereby, we have focused on dependencies between executable process activities, invoked service operations and application components implementing these services. Functional elements of interest have been extended by metrics which either can be measured or calculated. These metrics are the basis for the definition of metric dependencies. In order to estimate the impact of QoS on QoP, we have introduced a formalism to define calculation instructions. For the definition of a metric dependency function, we have applied a pattern of metric dependencies between activity duration and response time of a service operation. In our ToR process example, we assumed that only metrics at the invoked service and the components implementing this service could be measured. Metrics related to process activities have to be calculated. As example, only a simple summation instruction has been derived.

For monitoring impacts of QoS on QoP at runtime, the definition of metric dependencies is based on established management standards like CIM [9]. Thus, it is possible to map elements of the metric dependency model to components of a standardized monitoring architecture. To measure and collect raw metrics, services and components implementing services have to be instrumented. To calculate metrics which relate to processes and depend on metrics related to invoked services, measured values have to be aggregated. The implemented monitor aggregates metric values and compares them with SLOs defined in SLAs. In case of SLA violation, the monitor sends a notification about the impact to the service requester.

The monitor implementation has proofed that the model-based definition of metric depedencies could be mapped to standardized management solutions.

B. Outlook

At this point, we measure metrics at the service provider and calculate metrics at the service requester. Therefore, we derived a very simple metric dependency function to explain our approach. In further work, we are investigating additional factors affecting the qualitative dependencies between processes and invoked services. Consequently, metric dependency functions are becoming more complex. In order to keep the balance of effort for defining and calculating dependency functions and validity regarding the impact on processes, we started to measure process metrics, too. Thereby, we use

process metrics to compare measured values with estimated values and to improve derived dependency functions.

In this work, we focused on a process metric related to only one activity as part of a process. In order to calculate the impact of invoked services on whole processes the scope has to be extended. Based on metric values measured at invoked services, the impact of an activity on the whole process has to be estimated. This is done by a process manager, who is able to monitor and control the implementation of a process. Therefore, not only the process logic but also the engine executing the process has to be taken into account. Depending on a process design, a process manager has to analyze how far a specific QoS affects not only a specific activity but also the quality of the whole process. An appropriate process design can compensate temporary QoS or SLA violations.

We presented a method for the definition of metric dependencies according to functional dependencies in the context of a SOA. These metrics have to be defined business-driven [2], in order to monitor qualitative impacts on business processes. To facilitate the measurement of business-driven metrics, an appropriate instrumentation of relevant functional elements has to be ensured. Consequently, metrics have to be modeled and implemented in connection with functional elements during the development of a SOA. Therefore, models as shown in this paper can be used. After the modeling of functional dependencies, and based on measurable metrics, metric dependencies should be derived automatically. Keeping this vision in mind, further work is going on.

ACKNOWLEDGMENT

The authors wish to thank the members of the research group Cooperation & Management (C&M) for helpful discussions and valuable comments on previous versions of the paper. C&M is a group of researchers at the Institute of Telematics at the University of Karlsruhe (TH). The group is directed by Prof. Dr. Sebastian Abeck.

REFERENCES

[1] J. Sauvé, A. Moura, M. Sampaio, J. Jornada, and E. Redziuk, "An introductory overview and survey of business-driven it management," in *1st IEEE/IFIP International Workshop on Business-driven IT Management*, apr 2006.

[2] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*. Wiley, 1994, vol. 1, pp. 528–532.

[3] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics: Science, Services and Agents on the World Wide Web Journal*, vol. 3, no. 1, 2004.

[4] W. Juling and A. Maurer, "Karlsruhe integrated information management (kim)," *Journal of Information Processing and Communication*, vol. 3, 2005, in German.

[5] *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*, Object Management Group (OMG), May 2006.

[6] *Web Services Business Process Execution Language Version 2.0*, Organization for the Advancement of Structured Information Standards (OASIS), Nov. 2006. [Online]. Available: http://www.oasis-open.org/committees/download.php/21575/wsbpel-specification_public_review_draft_2_diff.pdf

[7] *Web Services Agreement Specification (WS-Agreement). Version 2005/09*, Open Grid Forum (OGF), Nov. 2006. [Online]. Available: http://forge.gridforum.org/sf/docman/do/downloadDocument/projects.graap-wg/docman.root.current_drafts/doc13652

[8] *Web-Based Enterprise Management (WBEM)*, Distributed Management Task Force (DMTF). [Online]. Available: http://www.dmtf.org/standards/wbem

[9] *Common Information Model (CIM) Metrics Model*, Distributed Management Task Force (DMTF), Jun. 2003. [Online]. Available: http://www.dmtf.org/standards/documents/CIM/DSP0141.pdf

[10] A. Sahai, J. Ouyang, V. Machiraju, and K. Wurster, "Specifying and guaranteeing quality of service for web services through real time measurement and adaptive control," E-Services Software Research Department, HP Laboratories, Palo-Alto, E-Service Management Project, 2001.

[11] M. Sayal, A. Sahai, V. Machiraju, and F. Casati, "Semantic analysis of e-business operations," *Journal of Network and Systems Management*, vol. 11, no. 1, Mar. 2003.

[12] A. Sahai, V. Machiraju, M. Sayal, L. J. Jin, and F. Casati, "Automated sla monitoring for web services," in *13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Oct. 2002.

[13] M. Debusmann and A. Keller, "Sla-driven management of distributed systems using the common information model," in *8th IFIP/IEEE International Symposium of Integrated Network Management*, Mar. 2003.

[14] H. Ludwig, A. Keller, A. Dan, and R. P. King, *Web Service Level Agreement (WSLA) Language Specification*, IBM Corporation, 2003.

[15] H. Ludwig, A. Dan, and R. Kearney, "Cremona: An architecture and library for creation and monitoring of ws-agreements," in *2nd International Conference on Service Oriented Computing*, Nov. 2004.

[16] J. Sauvé, F. Marques, A. Moura, M. Sampaio, J. Jornada, and E. Radziuk, "Optimal design of e-commerce site infrastructure from a business perspective," in *39th Annual Hawaii International Conference on System Sciences*, Jan. 2006.

[17] A. Arsanjani, "Service-oriented modeling and architecture," IBM developerWorks, Tech. Rep., Nov. 2004. [Online]. Available: http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1

[18] *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, World Wide Web Consortium (W3C), Mar. 2004. [Online]. Available: http://www.w3.org/TR/wsdl20

[19] D. A. Menascé and V. A. F. Almeida, *Capacity Planing for Web Services*. Prentice Hall PTR, 2002.

[20] S. Kalepu, S. Krishnaswamy, and S. W. Loke, "Verity: a qos metric for selecting web services and providers," in *4th International Conference on Web Information Systems Engineering Workshops*, Dec. 2003.

[21] A. Sahai, V. Machiraju, J. Ouyang, and K. Wurster, "Message tracking in soap-based web services," in *8th IEEE/IFIP Network Operations and Management Symposium*, Apr. 2002.

[22] *Application Response Measurement (ARM)*, The Open Group, Dec. 2004, issue 4.0 Version 2 - Java Binding. [Online]. Available: http://www.opengroup.org/tech/management/arm

[23] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services. Concepts, Architectures and Applications*. Springer, 2004.

[24] C. Mayerl, F. Tröscher, and S. Abeck, "Process-oriented integration of applications for a service-oriented it management," in *1st IEEE/IFIP International Workshop on Business-Driven IT Management*, Apr. 2006.

[25] "Wbem services. java web based enterprise management." [Online]. Available: http://wbemservices.sourceforge.net