A Flexible Approach for Business Processes Monitoring

Diana Comes, Steffen Bleul, Thomas Weise and Kurt Geihs

Distributed Systems Group, University of Kassel,
Wilhelmshöher Allee 73, 34121 Kassel, Germany
{comes,bleul,weise,geihs}@vs.uni-kassel.de
http://www.vs.uni-kassel.de/

Abstract. Business processes and their implementation as Web Service Compositions are not only dependent on Web Services and partners all over the Internet, but also on their failsafe execution. Service providers have to obligate their services to perform according to negotiated Quality of Service (QoS) parameters. For example, response time and throughput are important parameters to achieve fast and efficient services. Overloaded or failing services may compromise the reliability and execution of whole enterprise processes.
In this paper we introduce a flexible monitoring approach for the measurement of QoS in BPEL (Business Process Execution Language) processes. We propose a generic algorithm for QoS aggregation in BPEL processes. The novel generic aggregation algorithm applies customized aggregation functions for QoS dimensions. Furthermore, we present a BPEL monitoring system which supports ad-hoc sensor deployment and efficient runtime and offline data aggregation not only for whole process descriptions but also sections inside service processes.

Key words: Business Processes, Quality of Service, BPEL, Web Services, QoS Aggregation, Monitoring

1 Introduction

Web Services and BPEL processes are the de-facto standard for implementing business processes in SOAs. Enterprise software is encapsulated inside web services and offered to the partners over the Internet. Web services are composed into more complex BPEL workflows which thus implement business processes. The number of enterprises that adhere to this technology is rapidly increasing, since enterprises need to offer services across organizational boundaries to their partners. Since several services may provide the same functionality, the *Quality of Service* makes the difference between different offers for business processes.

In order to ensure fast executing business processes, the QoS of a service must fulfill the expectations of its client applications. If the business process does not meet the quality requirements, actions need to be performed in order to improve its behavior. Therefore the monitoring, measurement and evaluation

of non-functional properties of the processes is imperative. The Business Process Execution Language (BPEL) standard has emerged for the implementation of interaction between services over the Internet. BPEL enables the specification of Web Service orchestrations. However, BPEL does not contain any specifications regarding the QoS of a business process.

In this paper we address these non-functional requirements. We present a monitoring and assessment approach for the computing of QoS in business processes. Our assessment approach is flexible enough to fulfill the requirements of a continuously changing environment. So far, most research studies have not dealt with the following QoS issues in workflows: Automatic sensor deployment, replacing QoS parameters, customizable aggregation functions, and subsection aggregation. Thus, these related studies do not put enough focus on the needs of heterogeneous and dynamic changing environments.

The remainder of this paper is structured as follows. Section 2 presents the motivation for our flexible monitoring approach. Section 3 makes a short introduction to the WS-BPEL language and some of its main constructs. A formalization of our model can be found in section 4. In section 5 the generic algorithm for QoS computation of a business process is presented. The paper proceeds with section 6 where we give an overview of our monitoring framework. The evaluation of our framework can be found in section 7. We make a comparison to other works related to ours in section 8.

2   Motivation

A Service Oriented Architecture is a dynamic environment where services and respectively partners are continuously changing. We can describe the interaction between these services with BPEL, but BPEL does not include extensions allowing us to monitor or to ensure the performance. A SOA promotes the ability for flexibility and change, but this is not possible for the assessment of QoS related issues. At any time, new QoS dimensions like cost and bandwidth have to be introduced, measured and aggregated in order to allow a suitable evaluation for performance. We designed a flexible approach, where QoS parameters can be easily considered and aggregated with minimum effort on manual administration and no effort spent by the business process architect. In this paper we tackle the following issues:

Automated Deployment: A business process includes a set of activities in order to invoke Web Services and each activity performs at a certain QoS. In several research studies (e.g. [9], [10]), the BPEL process description is interweaved with comments and extra activities are inserted additionally to the BPEL process. These artifacts are used to define the required QoS parameters, its monitoring sources and their aggregation functions. Thus, every time the process description changes its behavior or partner services then the process architect has to adjust the whole QoS assessment artifacts. Our goal is not to alter the process description with artifacts for process monitoring.

Aggregation Functions and QoS Parameters: In contrast to the measurement of QoS for single Web Services, the business processes additionally consist of different activities such as if-conditions, loops and parallel invocations of Web Services. This is why the measurement of QoS in business processes needs to be treated differently as for web services. The QoS value of a business process is computed out of the QoS values of the building blocks inside the process.

Usually, the QoS requirements and implicitly, the corresponding QoS measurements for business processes, vary over time. Thus the QoS monitoring and measurements need to be done as flexible as possible. For example, a service provider must ensure a certain response time for his business process. If, for some reason, the response time of the process is not the expected one, the service provider is in charge for analyzing the bad performance of the process. As the response time may be compromised by the bandwidth, the provider may also want to introduce the new QoS dimension bandwidth in the monitoring. In our approach we introduce a generic QoS assessment algorithm where we must only provide a set of aggregation functions in order to make it work with newly introduced QoS parameters.

Process and Section Measurement: Within a process description, we also consider sub-orchestrations, which we call sections of a business process. By section, we refer to a part of a BPEL process which begins within one activity and ends with another activity, while the second activity is triggered after the first one. Also, a structured activity like a `while` loop may represent a section, contained within the beginning and the end of the structured activity. However, a section may contain sequential and concurrent activities as well. Performing measurements and monitoring in a section is important as we further need to specify QoS requirements and manage the process within sections. This way the business process architect has a better view on what sections of the business process may cause problems in the behavior of the entire process.

3   An Overview of WS-BPEL

WS-BPEL [11] is an XML-based OASIS standard for describing and executing business processes which consist of web services. BPEL allows arranging web services in sequences, loops, and to execute them in parallel. A process specified with BPEL begins its execution with a start activity such as a `receive` or a `pick`. The BPEL engine creates a new instance of the process when a certain message arrives. BPEL activities are triggered either sequentially or concurrently. With `invoke`, a single web service is executed while the values of variables are set with `assign`. The activities nested inside a `flow` element are executed concurrently. In contrast, the activities defined by elements nested inside `sequence` are triggered in the same order that they appear.

Besides the control flow, additional information such as `partnerlinks` for defining corporate bodies that participate in the business process and `faultHandlers` listing the activities to be executed in case of failures can be specified with BPEL as well. Additional aspects like defining QoS parameters

and QoS aggregation cannot be specified with BPEL. To fill these gaps, we apply our model and algorithm.

4   Business Process Model incorporating QoS

We have designed a generic model for monitoring a business process by computing its QoS properties by aggregating the QoS of its components. In this section, we will provide the definitions necessary to specify this approach. In our model, we divide all BPEL elements relevant in the context of QoS computation into two classes:

1. the set of simple element types $S = \{$receive, reply, invoke, assign, throw, wait, ...$\}$ and
2. the set of complex element types C which are used for structuring the control flow like sequence, flow, if, while, and foreach, for example.

Thus, the BPEL activity types are members of the joint set $T = S \cup C$. The instances of a simple element contribute directly to the quality of service of the overall process. They do not contain any child elements from T. Complex elements, on the other hand, may contain arbitrary other complex or simple elements. They specify how these elements are to be executed and their QoS values can be computed by aggregating the QoS of their children.

For each element *elem* in a BPEL process specification which belongs to one of the types in T, a unique identifier *elem.id1* $\in$ *I1*, *I1* $\subset \mathbb{N}$ will be assigned in the initialization phase of our system. We furthermore assume that *each single execution* of *elem* has an identifier *id2* $\in$ *I2*, *I2* $\subset \mathbb{N}$ unique in the current process instance.

The quality dimensions q which can be measured in our system, are subsumed in the set Q. One example for such a set Q could be $\{$responsetime, availability, cost, bandwidth$\}$. For each quality dimension q, there exists a domain d_q which defines the set of possible values of this QoS feature. For $q =$ cost $\in Q$, d_{cost} would be $\{x | x \in \mathbb{R}^+\}$, for instance. We define the domain D as the union of all the domains d_q. The computation of the actual quality of service values in our model is based on two functions:

1. $f_{value} : Q \times I2 \mapsto D$ which determines the QoS value of a single invocation of a simple element and
2. $f_{agg} : Q \times C \times D^* \mapsto D$ aggregating all QoS values of the elements nested inside a complex element (where D^* is the set of spaces of vectors of arbitrary dimensionalities over the quality domains).

Whereas $f_{value}(\text{cost}, 9)$ would return the single value from d_{cost} which resulted from the invocation of a simple element with *id2*$= 9 \in$ *I2*, we could define $f_{agg}(\text{cost}, \text{sequence}, X)$ as $\sum_{i=1}^{n} \boldsymbol{x}_i$, where $X = (x_1, x_2, \ldots, x_n)$ is a vector in d_{cost}^* and n would be the number of elements in this vector. For $X = (0.01, 0.03, 0.08)$, $f_{agg}(\text{cost}, \text{sequence}, X)$ evaluates to 0.12, for instance.

In Table 1, a set of such aggregation functions are listed for representative quality dimensions. The vector X contains the QoS values x_i that correspond to a process execution, which means that the corresponding activities were executed. We adapted the aggregation formulas from [1] to our approach. Since [1] consider a stochastic model and we perform aggregations on running or completed instances, we set the probabilities of executing an activity to 1 and obtained the functions from table 1:

QoS Dimension q	$fagg(q, c, X)$ $c = $ sequence	$fagg(q, c, X)$ $c = $ switch	$fagg(q, c, X)$ $c = $ flow	$fagg(q, c, X)$ $c = $ while
$q=$responsetime	$\sum_{i=1}^{n} x_i$	$\sum_{i=1}^{n} x_i$	$\max_{i \in 1..n} \{x_i\}$	$\sum_{i=1}^{n} x_i$
$q=$cost	$\sum_{i=1}^{n} x_i$	$\sum_{i=1}^{n} x_i$	$\sum_{i=1}^{n} x_i$	$\sum_{i=1}^{n} x_i$

Table 1. Aggregation Functions

5   The Quality of Service Aggregation Approach

In the following we present the generic algorithm for the QoS computation of the business process and its sections. We therefore assume that the values of the f_{value}-function for the simple elements within the process are known. The generic algorithm computes the QoS value of the entire business process and/or its sections. The QoS aggregation of the BPEL process is done in several steps and there are two ways for QoS aggregation possible: A) aggregation on stored monitored data and B) live aggregation. Our algorithm is applicable in both scenarios, QoS aggregation during runtime and also post-processing after the processes have finished their execution.

5.1   Startup phase: The BPEL Tree

During the startup phase of our system, the BPEL documents are translated into tree prototypes which contain only nodes for BPEL elements which are instances of either simple or complex elements. Since $T = S \cup C$ only contains the types of elements which are relevant for the execution of the BPEL process, this tree prototypes $Tree$ do not contain nodes for partnerLinks, for instance. Each node $qnode$ of the tree has the same structure and contains

1. the type $qnode.elem \in T$ of the element representing the node,
2. a unique identifier $qnode.id1 \in I1$ of the element in the tree,
3. the list with m quality dimensions $qnode.qDimensions \subseteq Q$ which are monitored or aggregated for this node,
4. a map $qnode.ChildrenValues \in D^{* \times n}$ which can hold the corresponding QoS values of the n children of the node; we need this map due to the propagation nature of our algorithm,

5. the map $qnode.Value$ holding a value $qnode.Value(q)$ for each of the quality dimensions $q \in qnode.qDimensions$ monitored for the element itself, and
6. a reference $qnode.parent$ to the parent node of $qnode$ (or `null` if $qnode$ is the root node of the tree).

Both $qnode.ChildrenValues$ and $qnode.Value$ are initially empty and remain empty in the prototype tree $Tree$. For each instance of the business process, our monitoring system creates a new copy $tree$ of $Tree$. In these copies, $qnode.ChildrenValues$ and $qnode.Value$ are filled in by the system. The result of quality measurement of a process instance is then a tree which contains the QoS values for each element of the business process in the field $qnode.Value$ of the corresponding node $qnode$. We furthermore define the function `getQNode`$(tree, id1)$ which returns the node $qnode \in tree$ with $qnode.id1 = id1$ (or `null` if such a node does not exist in $tree$).

We will call a node $qnode$ a simple node if it has $qnode.elem \in S$. Analogously, we call a node $qnode$ a complex node if it has $qnode.elem \in C$. Since simple elements cannot contain other elements, simple nodes are the leaves of the process trees.

5.2   Monitoring a Running BPEL Process

While the business process is running, our monitoring system records a list $execution$ of records $execElem$, each holding

1. the unique identifier $execElem.id1 \in I1$ of the element which was invoked,
2. the unique identifier $execElem.id2 \in I2$ of the invocation itself,
3. and the measured quality of service values $execElem.Value$ which provide the results of the f_{value}-function
$$\left(execElem.Value(q) = f_{value}(q, execElem.id2) \, \forall q \in Q\right)$$

for a single invocation of an element in the BPEL tree. While the process is running, whenever an activity corresponding to a node in the process tree is finished, a new $execElem$ record is added to the execution list.

5.3   The QoS Aggregation Algorithm

The generic algorithm provides as a result the aggregated values for m QoS dimensions of an execution path of a BPEL tree or its sections. As input, the algorithm expects the execution list $execution$ and a copy $tree$ of the BPEL prototype tree. The generic algorithm listed below can be applied for any type of QoS dimension if suitable aggregation functions are provided.

The QoS values $qnode.Value$ of a complex node $qnode$ are computed from the values of its direct children in the tree. By applying the aggregation functions f_{agg} on the QoS values of the children nodes, we obtain the QoS values of $qnode$. The values of simple nodes are known from the $execElem$-records and given by the value of the f_{value}-function.

Algorithm 1: `aggregateQoS`($execution, tree$)

Input: $execution$: the execution list
Input: $tree$: the process tree to be filled with QoS values

1 begin
    `// analyze the complete execution list` $execution$
2   **for** $i \longleftarrow 1$ **up to** $execution.length$ **do**
3 $qnode \longleftarrow$ `getQNode`$(tree, execution[i].id1)$
4   **foreach** $q \in qnode.qDimensions$ **do**
5   **if** $qnode.elem \in C$ **then**
                `// the node` $qnode$ `is a complex node`
6 $qnode.Value(q) \longleftarrow$
                $f_{agg}(q, qnode.elem, qnode.ChildrenValues(q))$
7   **else**
                `/* the node` $qnode$ `is a simple node and` f_{value} `is`
                  `equivalent to` $execElem.Value$ `*/`
8 $qnode.Value(q) \longleftarrow f_{value}(q, execution[i].id2)$
            `// propagate this QoS value of this node to the parent`
            `node`
**9**   `addQToChildrenValuesOfParent`$(qnode.parent, q, qnode.Value(q))$

10 end

The Algorithm 1 starts with traversing the list *execution* of executed activities. Each record $execElem \in execution$ stands for a completed activity. Since the QoS values of an activity can be computed in the moment the activity is finished, each record allows us to derive a set of QoS values. In the case of an *execElem* which denotes completion of an activity belonging to a simple node, the QoS values are the data directly stored in *execElem* corresponding to the f_{value} function. If *execElem* belongs to complex node, its occurrence means that the QoS of this node can be aggregated from its child nodes since an activity can only terminate after all of its children have terminated. In both cases, the new QoS values are propagated to the parent node.

Because of this propagation nature, the steps 3 to 9 of algorithm 1 can also be executed online while the process is running. In other words, the quality of service of the process tree *tree* can be built on the fly. If this is done, components which supervise or enforce policies such as Service Level Agreements (SLA) or management components like our BPRules framework [6] for business process management, can be easily integrated.

5.4   Example

Figure 1 represents an example of a BPEL process execution. It is an example of QoS aggregation for *response time*. On the left side of the figure, the monitored values are represented. These are the *ids* of the executed activities, in the order of execution. Also we monitored the value of *response time* (which represent

fvalue) for the simple elements (e.g.: `receive`, `reply`, `assign`, `invoke`). These represent the input data to the QoS aggregation algorithm.
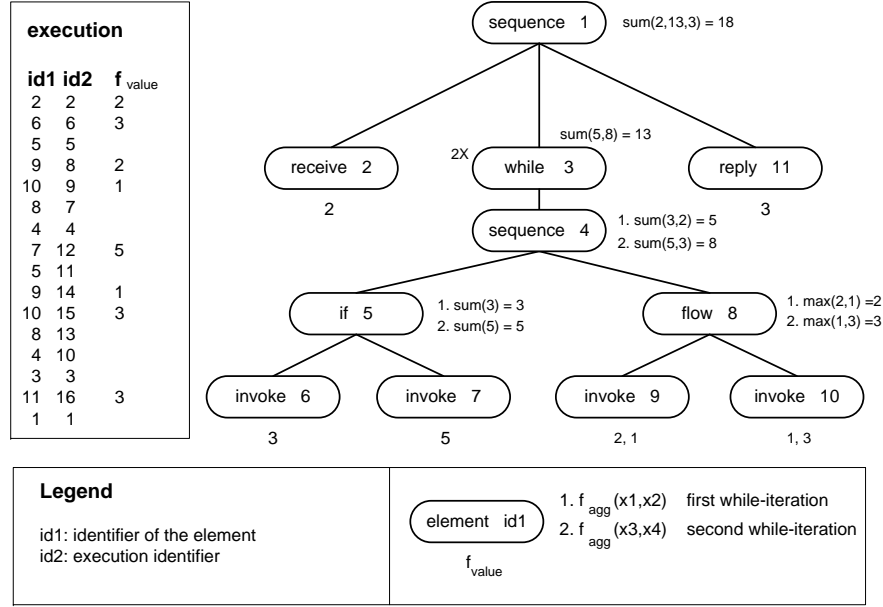
Fig. 1. Response Time aggregation

The `while` element (*id1=3*) performs two iterations. The activities inside the `while` have two values except for the `if` element *(id1=5)*. In the first iteration the first branch of the `if`-element is executed. The computation starts with the `receive` element, *id1=2, $f_{value}(responsetime,2)=2$*, which is the first element that is completed. Then this value is added to the *ChildrenValues* map of the parent node (`sequence`, *id1 = 1*). The process is continued with every completed element in the *execution* list. Finally, the last completed element is `sequence` with *id1 = 1*, which is also the root of the tree. By applying the aggregation function on the values of the children (the *ChildrenValues* map) of the root node ($f_{agg}(responsetime, sequence, X) = 2+13+3=18$), we determine the aggregation value of the *response time* for the entire tree.

6   Automated Deployment and Monitoring Framework

The main tasks of our framework are the automated deployment, monitoring and assessment of BPEL processes. For the monitoring purpose, previous to the deployment, sensors need to be associated to the BPEL process. The monitoring task is supported by the utilization of sensors which is a feature offered by
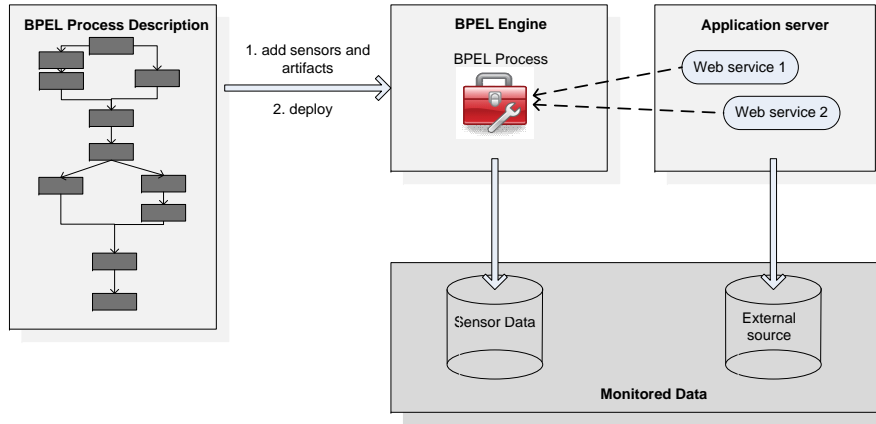
Fig. 2. The framework

the Oracle BPEL Process Manager engine, where our business processes are deployed. A sensor is associated to a BPEL activity and is fired during the execution of the activity and on the occurrence of certain events.

Before process deployment, we dynamically associate sensors to each activity in the process. The sensors are declared apart from the BPEL process description inside separate XML files. We automatically generate these sensors files from the BPEL description. They are then interpreted by the Oracle BPEL engine for firing the sensors. The sensors provide valuable information, such as the timestamp when the associated activity was activated, completed or faulted. Figure 2 illustrates the deployment of a BPEL process and its associated sensors to the BPEL engine. The BPEL process and its web services may be monitored by different parties and the monitored data is stored into different sources.

Even if our generic algorithm takes advantage of the sensors offered by the Oracle BPEL Process Manager, it could also run with another BPEL engine as well. If the BPEL engine used does not support sensors similar to those the Oracle Engine provides, these software components should be additionally implemented. The premise that our algorithm runs on another BPEL engine is that its input data is delivered properly. The direct impact of sensors is on the creation of the *execution* list, which contains the identifiers of the activities in the order that they were triggered.

The main component of our assessment framework is the *QoS Aggregator* which performs the aggregation of the business process and its sections. Figure 3 illustrates the QoS aggregator component as well as its input and output data.

The flexibility of our monitoring framework is sustained by the use of plug-ins, which are software components. The advantage of using plug-ins is that they may be easily added. As the aggregation and value functions (f_{agg} and f_{value}) are subject to modifications, we will use for each aggregation and value functions additional plug-ins. Plug-ins are reusable components. The f_{agg} function plug-
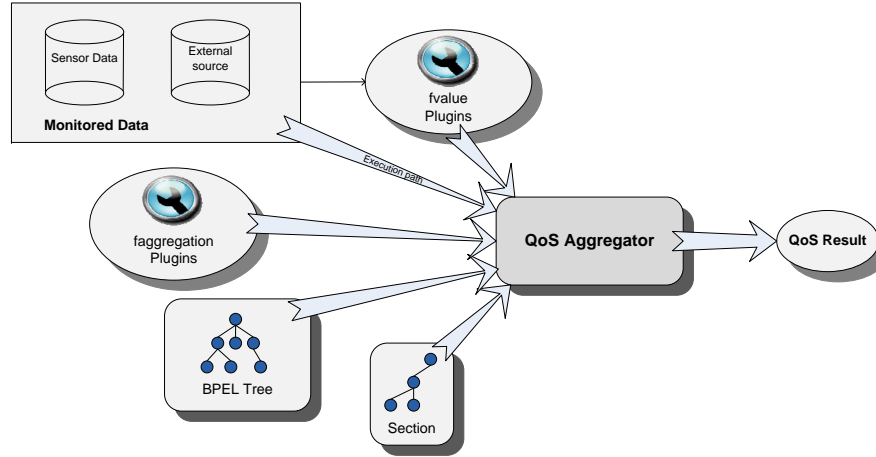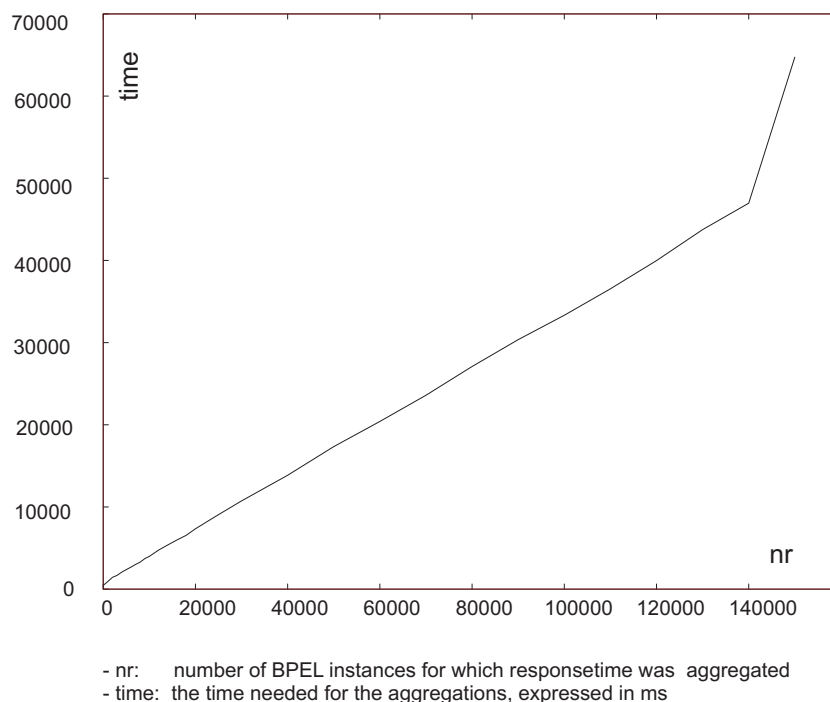
Fig. 3. The QoS Aggregator Component

ins can be reused over several instances since they only depend on the type of BPEL activities. The advantage of the f_{value} function plug-ins is that they can retrieve monitoring data from other sources as well, regardless where the monitoring actually takes place. An example for this is a situation where web services inside the BPEL process are monitored by the web service providers themselves and the monitored data is stored into a database different from the database where the data from the monitored BPEL *process* is stored. This has also been depicted in figure 2.

7 Evaluation

As example application, we have developed a *Bookshop* business process which is implemented with WS-BPEL. The *Bookshop* process is a regular business process for purchasing books at an online book shop. It interacts with four web services: the *Stock Service*, the *Distributor*, the *Accounting* and the *Bank Service*. It runs on the Oracle BPEL Process Manager and the web services are installed on the Oracle Application Server OC4J. The monitoring data, provided by the Oracle BPEL engine (e.g. instance data, sensor data) is automatically inserted into the Oracle Database. The assessment data is written into a custom MySQL database.

We performed several evaluation tests on the *Bookshop* process. The test system is a Lenovo R60 notebook with Intel Core 2 Duo processor T5600 (2x1,83GHz) with 2GB memory and Windows XP Professional Version 2002, Service Pack 2.

For the *Bookshop* process, we defined a section consisting of 20 elements, while the entire process consists of 46 elements. We measured the time consumption for the generic aggregation algorithm, by iteratively increasing the

- nr: number of BPEL instances for which responsetime was aggregated
- time: the time needed for the aggregations, expressed in ms

Fig. 4. Example Business Process Aggregation

number of process invocations. Figure 4 represents the time in ms needed for the response time aggregation for the entire process. We observe a linear growth of time in relation to the number of instances that were aggregated. Since we wanted to test how our algorithm behaves if the memory is limited, we set the Java Heap size at 90 MB. When the entire memory was allocated (at about 150000 of process instances), we also observe an abrupt increasing in the time consumption. This is an expected result and shows that the algorithm performs well but will fail if the resources of the system it is deployed on are exhausted.

8   Related Work

In his thesis [3], Cardoso describes a framework for estimating, predicting and analyzing QoS in workflows. For this QoS computation, he presents a mathematical model and a Stochastic Workflow Reduction (SWR) algorithm. The SWR algorithm uses six reduction rules: sequential, parallel, conditional, fault-tolerant, loop and network. These rules are iteratively applied on the workflow until one atomic task remains. The QoS value of this remaining task represents the QoS value of the workflow. Even if the rules apply on most workflows, for specific cases that may apply in a BPEL process, new rules have to be developed.

Different from Cardoso's work, we do not focus on deriving a statistical model or to predict QoS values but only consider their measurement on a running system. Thus, our approach would be a possible input source from which estimators for future process behavior could be built.

In [4], Zeng et al. present a middleware platform for web service compositions from the QoS perspective. There are two approaches described for the selection of web services by satisfying the constraints set by the user on QoS attributes. Their approach is based on state charts and execution plans represented as DAGs (direct acyclic graphs). The authors define aggregation functions for price, duration, reputation, success rate and availability.

In [7], M. Jaeger presents a method of QoS aggregation for service compositions based on composition patterns. There were nine composition patterns identified that might occur in a workflow. A workflow structure is represented as a graph which is collapsed step by step due to the composition patterns that were identified in the workflow until one statement remains. The aggregations are performed on the level of composition patterns. Dependent on the pattern and QoS characteristic, aggregation rules are defined. Yet, the identification of the composition patterns on the workflow graph is not a trivial task.

Mukherjee et al. focus in their work [5] on the QoS computation in BPEL processes. They address three QoS dimensions response time, cost and reliability. They also utilize fault tolerance techniques (e.g. Recovery blocks, N-version programming) to study the impact on QoS. The computations are performed on an activity graph, which nodes represent BPEL activities and handlers.

Canfora et al. [1] adopt a similar approach to Cardoso [3] for QoS computation. They apply the same aggregation functions as Cardoso, except for loops. In our approach we used the same functions as Canfora but, as already stated, focus on measurement instead of prediction. Thus, probabilities for a certain control flow do not exist in our approach or, from another perspective, are always 1 or 0, since we aggregate the data a posteriori. In their work, Canfora et al. propose a solution with genetic algorithms for the service selection problem in service compositions.

In [10] Baresi et al. are also concerned with the monitoring of WS-BPEL processes. They add monitoring rules to the BPEL process by inserting them as comments to the source code. For monitoring purpose, we profit from the utilization of sensors that are directly triggered by the BPEL engine, and do not have to insert extra monitoring artifacts.

In our approach, we do not apply workflow patterns or reduction rules like [7], [3] and [1]. We represent the BPEL process as a tree and by only applying aggregation functions on the nodes values we are able to compute the QoS value of the entire business process. We also provide a generic algorithm that is applicable for any QoS dimension if appropriate aggregation functions are provided.

9   Conclusion

In this paper, we presented a flexible approach for the monitoring and computation of QoS in business processes. We have demonstrated the feasibility of our method on a business process that we implemented with WS-BPEL. We have developed a generic algorithm that performs the computation of any QoS dimension if appropriate aggregation functions are available. The algorithm may be applied both at runtime or after a business process terminated its execution. By executing the algorithm at runtime, the process is additionally observed whether it behaves as expected. Otherwise, appropriate management actions may be triggered to improve the process behavior. We also support QoS monitoring and computation on sections of the process, which permits detecting the sections of the process that cause problems and might lead to undesired QoS values of the process.

We take advantage of the utilization of sensors for the monitoring purpose. Sensors are dynamically associated with each activity of the BPEL process and the BPEL process description is not affected by extra monitoring artifacts. New QoS dimensions can be integrated with minimal effort by only specifying the aggregation functions.

References

[1] Canfora, G., Penta, M., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 1069–1075. ACM, Washington DC (2005)

[2] Canfora, G., Penta, M., Esposito, R., Villani, M.L.: A Lightweight Approach for QoS-Aware Service Composition. Technical report, Research Centre on Software Technology University of Sannio (2004)

[3] Cardoso, J.: Quality of Service and Semantic Composition of Workflows, PhD thesis, University of Georgia, Georgia (2002).

[4] Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. In: IEEE Transactions on Software Engineering, pp. 311–327. IEEE Press, (2004)

[5] Mukherjee, D., Jalote, P., Nanda, M. : Determining QoS of WS-BPEL Compositions. In: Proceedings Service-Oriented Computing ICSOC 2008, pp. 378–393. Springer, Heidelberg (2008)

[6] Comes, D., Bleul, S., Zapf, M.: Management of the BPRules Language in Service Oriented Computing. In: 16th Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009, WowKiVS, Electronic Communications of the EASST, Kassel (2009)

[7] Jaeger, M.: Optimising Quality of Service for the Composition of Electronic Services, PhD thesis, University of Berlin, Berlin (2007).

[8] Charfi, A., Schmeling, B., Heizenreder, A., Mezini, M.: Reliable, Secure, and Transacted Web Service Compositions with AO4BPEL. In: Proceedings of the European Conference on Web Services (ECOWS'06), pp. 23–34. IEEE Computer Society, (2006)

[9] Baresi, L., Ghezzi, C., Guinea S.: Smart monitors for composed services. In: Proceedings of the 2nd international conference on Service oriented computing, pp. 193–202. ACM, New York (2004)

[10] Baresi, L., Guinea, S.: Towards Dynamic Monitoring of WS-BPEL Processes. In: Proceedings of the Third International Conference, Service-Oriented Computing - ICSOC 2005 , pp. 269–282. Springer, Heidelberg(2005)

[11] Web Services Business Process Execution Language Version 2.0, OASIS standard, 2007, `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`

[12] Oracle BPEL Process Manager, 2008, `http://www.oracle.com/technology/products/ias/bpel/index.html`