Monitoring and Analyzing Influential Factors of Business Process Performance

Branimir Wetzstein* Philipp Leitner† Florian Rosenberg† Ivona Brandic† Schahram Dustdar† Frank Leymann*

*Institute of Architecture of Application Systems
University of Stuttgart
Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

†Distributed Systems Group
Vienna University of Technology
Vienna, Austria
lastname@infosys.tuwien.ac.at

***Abstract*—Business activity monitoring enables continuous observation of key performance indicators (KPIs). However, if things go wrong, a deeper analysis of process performance becomes necessary. Business analysts want to learn about the factors that influence the performance of business processes and most often contribute to the violation of KPI target values, and how they relate to each other. We provide a framework for performance monitoring and analysis of WS-BPEL processes, which consolidates process events and Quality of Service measurements. The framework uses machine learning techniques in order to construct tree structures, which represent the dependencies of a KPI on process and QoS metrics. These dependency trees allow business analysts to analyze how the process KPIs depend on lower-level process metrics and QoS characterisitics of the IT infrastructure. Deeper knowledge about the structure of dependencies can be gained by drill-down analysis of single factors of influence.**

I. Introduction

Business Process Management (BPM) encompasses a set of methods, techniques, and tools for modeling, executing and analyzing business processes of an organization [1]. Recently, BPM has been supported by a set of tools which have been integrated in order to support the business process lifecycle in a unified manner. Thereby, business analysts create a business process model, which is then refined by IT engineers to an executable model. The executable process model is deployed to a process engine, which executes the process by delegating tasks to humans and services. The execution of processes is often based on a Service Oriented Architecture [2] (SOA). In that case, the business process model is typically implemented as a service composition, for example in WS-BPEL [3].

An important aspect of the BPM lifecycle is the continuous supervision of business goals and timely measurement of business process performance. This is typically supported by business activity monitoring (BAM) technology, which enables continuous, near real-time monitoring of processes based on an eventing infrastructure [4]. Analysts define Key Performance Indicators (KPIs) and their target values based on business goals (e.g., "order fulfillment lead time $<$ 3 days"). KPIs are influenced by a set of Process Performance Metrics (PPM) [5], which are metrics based on process runtime data (e.g., "number of orders which can be served from inhouse stock"). PPMs are on a different level of granularity than KPIs: a KPI measures the success of the process as a whole, while a PPM captures only a single facet of the process, which is usually not interesting in isolation. Additionally, KPIs are influenced by technical parameters, i.e., the Quality of Service (QoS) metrics of the SOA (e.g., the availability of the process engine or the response time of Web services).

Business Activity Monitoring provides useful information on KPI achievement. However, the focus is set on the "what" rather than the "why" question. When KPIs do not meet target values the business analysts are interested in factors that cause these deviations. Since KPIs potentially depend on numerous lower-level PPMs and QoS metrics, these causes can be manifold, and are rarely obvious even to domain experts. In this paper we present an integrated framework for run-time monitoring and analysis of the performance of WS-BPEL processes. Our main contribution is the presentation of a framework for dependency analysis, a machine learning based analysis of PPMs and QoS metrics, with the ultimate goal of discovering the main factors of influence of process performance (i.e., KPI adherence). These factors are represented in an easy-to-interpret decision tree (dependency tree). We present the general concepts of our analysis framework, and provide experimental results based on a purchase order scenario, identify cases when dependency trees do not show expected results, and explain strategies how these problems can be coped with.

The rest of the paper is organized as follows. In Section II we present a scenario which we use for explaining our concepts and for experimentation and explain the research issues that this paper deals with in more detail. Section III explains the main ideas of our framework for runtime monitoring and dependency analysis. Section IV explains in detail the monitoring of influential factors, which is then followed by the description of the dependency analysis in Section V. Section VI describes the implementation of our prototype based on the scenario and experimental results, which are also extensively interpreted. Section VII discusses important related work and Section VIII finally concludes the paper and presents some future research directions.

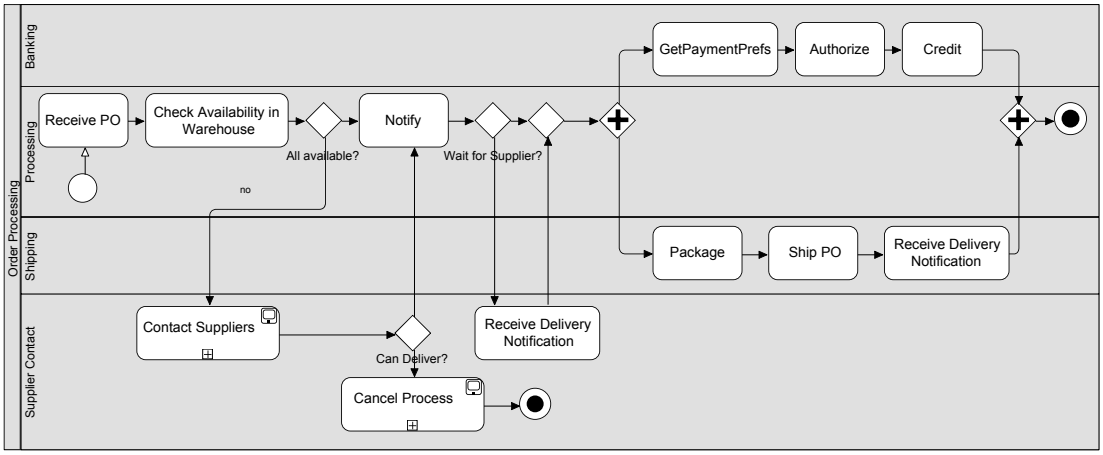

Fig. 1. Reseller Process Model in BPMN

II. Scenario

In this section we present a scenario which we use in the following sections for explaining our concepts and which we have implemented and used for experimentation purposes. We have chosen a purchase order scenario consisting of a customer, a reseller, its two suppliers, a banking service, and a shipping service. The business process of the reseller is illustrated in the BPMN diagram shown in Figure 1. The reseller offers certain products to its customers. It holds a certain part of the products in stock and orders missing products from suppliers if necessary. The customer sends a purchase order request with details about the required products and needed amounts to the reseller. The latter checks whether all products are available in stock. If some products are not in stock, they are ordered from suppliers. Note that the second supplier is contacted only if the first (preferred) supplier is not able to deliver. If the purchase order can be satisfied, the customer receives a confirmation, otherwise the order is rejected. The reseller waits, if needed, for the supplier to deliver the needed products. When all products are in place, the warehouse packages the products and hands them over to the shipment service, which delivers the order to the customer, and finally notifies the reseller about the shipment. In parallel to the packaging and shipment, the payment subprocess is performed. For that, the customer decides on the payment style and gives its payment details. The reseller contacts a banking service which authorizes the customer and credits the agreed amount. From the point of view of the reseller, a typical KPI is the *order fulfillment lead time* (duration from receiving the customer order until shipment is received by the customer), as defined in the Supply-Chain Reference Model (SCOR) [6].

Assuming that this process is implemented using WS-BPEL, the KPI *order fulfillment lead time* is potentially influenced by a number of technical and non-technical factors, such as the response time and availability of Web services, the customer, or the products ordered (Figure 2). In Table I, we have provided an (incomplete) list of potential factors of influence for the KPI from our scenario. Factors can include simple facts from the business process instance, such as a customer

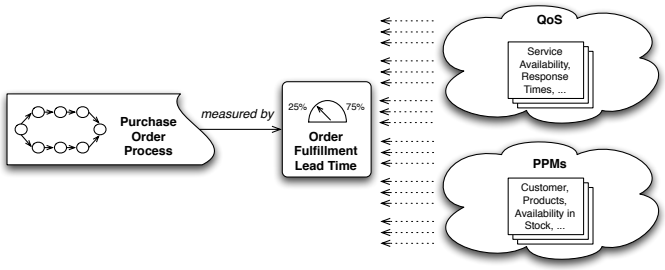

Fig. 2. KPIs, PPMs and QoS metrics

identifier, a product type, or information about which branch of a process has been executed (e.g., whether the alternative branch "ordering from external suppliers" needed to be executed). All these facts are accessible from the process instance, therefore, we have not given a calculation formula for these PPMs in the table. However, PPMs on a different level of granularity are also possible, such as the duration of a whole subprocess. Finally, we have given a few simple examples of QoS metrics, which may influence the KPI performance, such as the availability of the process engine or single services that the process relies on, or the response time of these services. A full discussion of possible QoS metrics is out of scope of this paper. The interested reader may refer e.g., to [7] for a more complete description of possible QoS metrics and their measurement. For completeness, we have also provided the possible range for these example influential factors. Generally, factors of influence can either be nominal values (i.e., take on one of a finite number of predefined values), or numeric values (e.g., integer or real values).

However, it is not obvious even to experts which of these factors actually influence the KPI most, and what the structure of the dependencies between factors of influence is (i.e., some factors are in turn influenced by others, such as the duration of the payment subprocess which is again influenced by service response times). These questions are not answered sufficiently by today's BAM dashboards – they can only provide status

Name	Type	Calculation Formula	Range
Customer ID	PPM		$\{Customer_1, Customer_2, \ldots\}$
Product Type	PPM		$\{Product_1, Product_2, \ldots\}$
Shipped from stock	PPM		$\{true, false\}$
Duration of Payment Subprocess	PPM	$t_{end} - t_{begin}$	$[0; \infty]$
Availability Process Engine	QoS	$\frac{\#available}{\#checks}$	$[0; 1]$
Availability Banking Service	QoS	$\frac{\#available}{\#checks}$	$[0; 1]$
Response Time Banking Service	QoS	$t_{end} - t_{begin}$	$[0; \infty]$

TABLE I
POTENTIAL INFLUENTIAL FACTORS OF KPI PERFORMANCE

information about KPIs, but do not allow further analysis of the main causes for violations. Our approach supports this kind of analysis, which we refer to as dependency analysis (i.e., the analysis of dependencies of KPIs to PPMs and QoS metrics). Furthermore, more detailed information about internal dependencies between factors of influence can be gained by drill-down analysis, i.e., recursively applying dependency analysis to single factors of influence.

III. FRAMEWORK OVERVIEW

In this section we describe the concepts of our framework for monitoring and analyzing factors of influence of business process performance. A high-level overview of the main components is given in Figure 3. In our framework we distinguish three different layers.

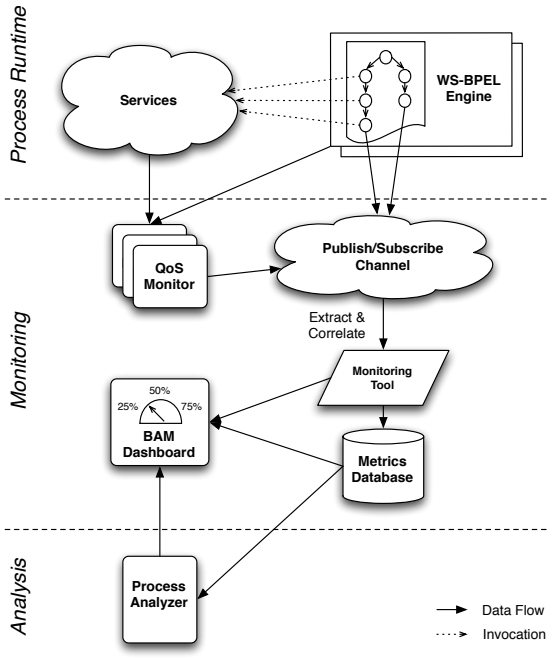

Fig. 3. Monitoring and Analysis Framework Overview

In the *process runtime* layer, a WS-BPEL business process is defined and executed. The process can be executed in a standard WS-BPEL compliant engine, as long as the engine is able to emit the process information necessary for calculating PPMs in form of process events.

In the *monitoring layer*, information about the running business process and the services it interacts with is collected in order to monitor KPIs, PPMs, and QoS metrics. Note that we assume that the user has defined a set of potential influential factors he wants to monitor for a KPI in order to ensure that corresponding metric data is available later on for analysis. KPIs and their potential infuential factors, consisting of both process performance metrics (PPMs) and QoS metrics, are modeled as part of a Process Metrics Definition Model (PMDM). Those metric definitions are deployed to the corresponding monitoring infrastructure components: QoS metric measurement directives to a QoS monitor(s); event definitions needed for PPM calculation to the WS-BPEL engine; and the whole PMDM to the monitoring tool. During process execution time, the QoS monitor and WS-BPEL process engine publish events to a publish/subscribe channel which the monitoring tool is subscribed to. The PPM and QoS metric values are calculated, stored in the metrics database for later analysis, and displayed in the BAM dashboard.

In the *process analysis layer*, the collected metrics information is analyzed by the process analyzer component. When the user is interested in performing a dependency analysis of KPIs, i.e., analyze the influential factors, the process analyzer gathers the needed metric data from the metrics database, prepares it for data mining, and uses a decision tree algorithm to generate a dependency tree which shows the influential factors of the KPI. Outcomes of the analysis are again displayed in the dashboard to the users of the system, who can use this resulting information to optimize the business process.

IV. MONITORING OF INFLUENTIAL FACTORS

We distinguish in our approach between PPMs and QoS metrics, which are supported by different monitoring mechanisms. PPMs are measured based on business events (e.g., OrderReceivedEvent) which are published by the WS-BPEL engine and other involved systems as the process instances are executed (Section IV-A). KPIs are specified over PPMs by providing a predicate (boolean-valued function) over their

values which evaluates to true if the target value is achieved, e.g., *order fulfillment lead time* < 2 days. QoS metrics are used for measuring the IT characteristics of the involved systems (e.g., availability and response time of Web service operations) and are provided either by dedicated QoS monitors or instrumentations of involved systems (Section IV-B).

A. PPM Monitoring

PPMs are metrics defined based on runtime events of processes. In the following, we focus on runtime events of WS-BPEL service orchestrations, but in general our approach supports arbitrary events of information systems participating in the business process.

PPM monitoring encompasses three phases: modeling, deployment, and monitoring. In the modeling phase, PPM definitions are specified in an XML file as part of the PMDM. A PPM definition specifies the name, description, unit, data type, and value calculation of the metric. When specifying PPMs we can distinguish between *atomic* and *composite* metrics. Atomic metrics are specified based on events which are published by the WS-BPEL engine. The metric value calculation of an atomic metric specifies how the value of the metric is retrieved from process event data (e.g., timestamp or a process data element). Composite metrics are calculated using diverse functions (arithmetic, aggregation, relational) based on atomic metrics and other composite metrics. When the PMDM is to be deployed to the monitoring tool, we generate *event filters* which subscribe to the events emitted by the process engine. Most engines support an event publishing mechanism, which in most cases is also to some extent configurable on which events to publish. During process execution, the PPM monitor receives events via event listeners which are subscribed to the publish/subscribe channel which the process engine uses for publishing events, and then calculates atomic metrics using event filters. After calculation of an atomic metric, its value is saved in the metrics database with a reference to the corresponding process instance. This reference is needed for later analysis. In the next step, all composite metrics which use that atomic metric are calculated in a recursive fashion.

In the following, we explain the PPM monitoring concepts based on a sample PPM from our scenario. Listing 4 shows a definition of an atomic metric in the PMDM which retrieves the information from a process variable on whether the supplier can deliver ordered products. To define an atomic PPM the user can use a set of predefined functions for measuring duration of activities and between activities, count the occurrence of activity executions, extracting the state of an activity or process, or access process data variables. The parameters of these predefined functions are linked to elements of process models such as a process activity, a data variable, or the process itself, and specify in addition the state of the process element when it is to be measured (e.g., started, halted, completed). In our example, we use a `variable` function which references an element containing that information in the corresponding process variable `orderItemsResponseMessage` and the activity `orderItemsFromSupplier1` at which the variable value

```
1  <ppm id="Supplier1_can_Deliver">
2   <name>Supplier1_can_Deliver</name
3   <dataType>boolean</dataType>
4   <calculation>
5    <calc:variable activity="orderItemsFromSupplier1"
6     variable="orderItemsResponseMessage" />
7   </calculation>
8   <attachments>
9    <activityAttachment
10    parameterName="orderItemsFromSupplier1"
11    xlink:type="simple"
12    xlink:href="PurchaseOrderProcess.bpel#xpointer(
13    //*[@name=&quot;orderItemsFromSupplier1&quot;])">
14    <activityStatus>completed</activityStatus>
15   </activityAttachment>
16   <variableAttachment
17    parameterName="orderItemsResponseMessage"
18    xlink:type="simple"
19    xlink:href="loanApprovalProcess.bpel#xpointer(
20    //*[@name=&quot;orderItemsResponseMessage&quot;])">
21    <variablePart>deliveryPossible</variablePart>
22   </variableAttachment>
23  </attachments>
24 </ppm>
```

Fig. 4. Sample PPM Definition

is to be read (lines 5-6). In the `attachments` block the links to corresponding WS-BPEL process elements are defined (lines 8-23). Thereby, we use XLink[1] and XPointer[2] to point to the XML elements in the WS-BPEL file. For example, we reference the activity with name `orderItemsFromSupplier1` in the `PurchaseOrderProcess.bpel` XML file (lines 12-13). In addition, we specify that at the state `completed` of that activity the corresponding event should be gathered (line 14). Note that we neglect here that activities can be performed several times per process instance if they are part of a loop; in that case one would have to additionally specify in which loop execution one is interested in.

When the PPM definition is to be deployed to the monitoring tool, assuming the usage of the Apache ODE BPEL Engine[3] (which we have also used for our scenario implementation), an event filter for the events `ActivityExecEndEvent` and `VariableModificationEvent` is generated. The first event is published when the corresponding activity has completed; it contains the process model name, the activity name, and identifiers of the corresponding process instance and activity instance. The second event is sent whenever a WS-BPEL variable has changed and contains the process variable data, the name of the process model, the name of the variable, and a process instance identifier.

The task of an event filter is to calculate an atomic metric as specified in the PPM definition based on received event data. Therefore, it has to deal with *event correlation* and *process instance management*. In the example above, for each process instance, the event filter collects (potentially more than one) `VariableModificationEvent` until it receives an `ActivityExecEndEvent`, and then chooses the *last* received `VariableModificationEvent` in order

[1] http://www.w3.org/TR/xlink/
[2] http://www.w3.org/TR/xptr-xpointer/
[3] http://ode.apache.org/

```
<qm id="PO_Process_Availability">
  <name>PO_Process_Availability</name
  <dataType>integer</dataType>
  <calculation>
   <availability>
    <endpoint>
      http://localhost:8082/.../poProcess?wsdl
    </endpoint>
    <testFrequencyPerMinute>20</testFrequencyPerMinute>
    <startTimePpm idref="PO_Process_Started_Time" />
    <endTimePpm idref="PO_Process_Completed_Time" />
   </availability>
  </calculation>
</qm>
```

Fig. 5. Sample QoS Metric Definition

to retrieve the most recent content of the needed process data variable. Thereby, those events are correlated using the process instance identifier which is part of both types of events. Note that in case of non-BPEL events the correlation could not be done based on the technical process identifiers (assigned by the BPEL engine), but based on a business identifier such as the `purchase order identifier`.

B. QoS Metrics Monitoring

The business processes we focus on in this paper are implemented as service compositions running on top of a SOA. Such processes have several dependencies on IT components and their QoS characteristics, which potentially influence business process performance. In our context there are three possibilities of measuring QoS: (1) probing by a separate QoS monitor, such as the one described in [7], (2) instrumentation of the WS-BPEL engine, or (3) instrumentation of the WS-BPEL process (evaluating QoS parameters using PPMs, e.g., response times of Web services can be estimated through WS-BPEL activity durations). In our scenario implementation, we use an external QoS monitor for measuring the availability of the process engine and partner Web services of the WS-BPEL process. The QoS monitor polls the corresponding endpoints and emits QoS events which contain information on their availability at a certain point in time. Response time is estimated based on the duration of the corresponding WS-BPEL `invoke` activity.

Just like PPMs, QoS metrics are defined in the PMDM. Listing 5 shows the definition of the availability metric for the purchase order process from our scenario. Thereby, we assume that availability is measured by an external QoS monitor by polling the corresponding Web service `endpoint` with a certain `testFrequencyPerMinute`. The QoS monitor will thus emit 20 QoS events per minute specifying whether the process was available. As the case for PPMs, an important aspect in QoS metric definition is their *correlation* with process instances. When a QoS metric is evaluated, it has to be assigned to process instances and/or activity instances it affects. In this context, there is a technical difference between an external QoS monitor and the instrumentation approaches considering the correlation of process instances and QoS events. As the instrumentation is internal to the engine, it has access to the context of the process instance. Thus, it is possible to write the process instance (and if needed activity instance) identifier as an attribute into the QoS event which can then be used to correlate the QoS event with the process instance. For example, in case of a response time measurement for an invoke activity, the engine instrumentation can include the process instance identifier and the activity instance identifier into the QoS measurement event. In case of an external QoS monitor this is not possible. Thus, in order to be able to correlate QoS measurements with the affected process instances for our sample metric, we specify in addition which QoS measurements (namely those between the start time `PO_Process_Started_Time` and completion time `PO_Process_Completed_Time` of the process instance) are to be taken into account when calculating the availability for a specific process instance.

V. Analysis of Influential Factors

The main idea of dependency analysis is to use historical process instances to determine the most important factors that dictate whether a process instance is going to violate its KPIs or not. The input of this analysis are stored metric values of process instances, which are available in the metrics database. The output of dependency analysis is a decision tree that incorporates the most important factors of influence of process performance. We refer to this tree as *dependency tree*, because it represents the main dependencies of the business process on technical and process metrics, i.e., the metrics which contribute "most often" to the failure or success of a process instance in respect to a KPI.

A. Background

In our approach we use decision tree learning, a well-established machine learning technique [8] for construction of dependency trees. Decision tree classifiers are a standard technique for supervised learning (i.e., concepts are learned from historical classifications, in our case dependency information is learned from monitoring of previous process instances). Decision trees follow a "divide and conquer" approach to learning concepts – they iteratively construct a tree of decision nodes, each consisting of a test; leaf nodes typically represent a classification to a category. In our case, only two categories exist (KPI has been violated, or not). One big advantage of decision tree algorithms in our context is their non-parametric nature: decision trees need only a very limited set of parameters (in the simplest case none) to work correctly, and can therefore be expected to provide useful results from the first run, without the need for extensive experiments with different parameter sets. Therefore, learning of dependency trees is completely automated and transparent to the user, allowing a business analyst to carry out dependency analysis with good results out of the box. We use the well-known C4.5 [9] and alternate decision tree (ADTree) [10] techniques, since their quality is well established in the community. For decision tree training we use 10-fold cross validation [8] to avoid having to split our historic process data into training, test and validation sets, and to estimate the classification error of the decision tree.

The classification error allows the business analyst to measure the quality of the dependency tree, i.e., how exact the tree represents the actual structure of real-world dependencies.

B. Creation of Dependency Trees

The inputs and outputs of dependency analysis are sketched in Figure 6. The analysis takes a set of historical process instance metric values as input (training set), and produces a tree representation of the internal dependencies.

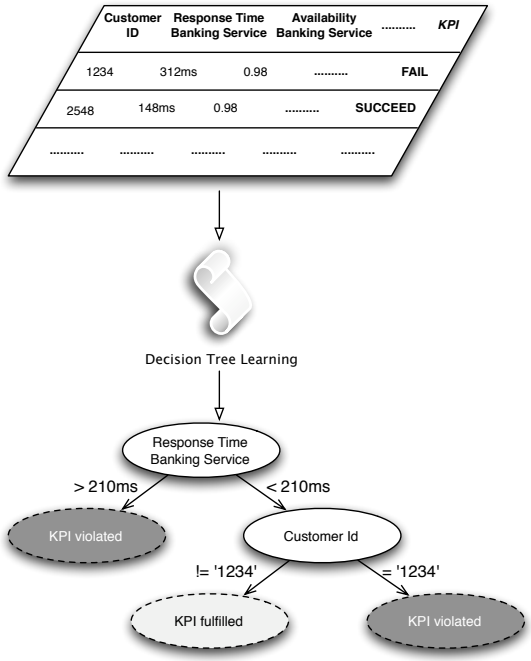

Fig. 6. Dependency Analysis Input and Output

The analysis consists of the following phases: (1) KPI selection and optional adjustment of analysis parameters, (2) creation of the training set, (3) decision tree learning, (4) displaying the result tree in the dashboard. All phases but the first one are performed automatically. In the first phase, the user chooses a KPI (from the PMDM) he wants to analyze. Optionally he can adjust the following parameters (or alternatively use default values): the *KPI target value* (and corresponding predicate); the *analysis period* and/or how many process instances in that period should be analyzed (e.g., last 1000); a subset of metric types from the PMDM which should be used as potential influential factors (default value: all); the decision tree algorithm which should be used. In the second phase, the creation of the training set is then performed automatically as follows: for each *process instance* which has begun and finished in the analysis period, the corresponding PPM and QoS metric values are gathered from the metrics database and used as attributes of a record of the training set. The predicate of the KPI metric value is evaluated and according to the result, the record is classified as "KPI fulfilled" or "KPI violated". An example training set is shown in Figure 6. Each row (record) contains the metric values (representing potential influential factors) of a process instance, whereby the last column specifies whether the KPI target value predicate (order fulfillment time $<$ target value) is fulfilled or violated for that process instance.

As a result of decision tree learning a dependency tree is constructed as shown in Figure 6 and displayed in the dashboard. In this (simple) example, the most influential factor is the response time of the banking service, since a delay in this service generally leads to a violated KPI. However, even if the banking services response time is acceptable (below 210 time units in this example), the KPIs are still often violated if the order is placed by the customer with the ID '1234'. Business analysts can use the dependency tree to learn about the "hot spots" of the process, and inform themselves about possible corrective actions if a process underperforms. For example, considering the example in Figure 6, a business analyst can take the corrective action to replace the "Banking Service" against a service with better response time, if such a service is available. However, note that the "first" metric used in the dependency tree is not necessarily the most important one – to find out about the most important metrics one needs to look at the whole tree and find out which decisions lead to the most failed process instances. For metrics which have been identified as factors of influence, a further "drill down" analysis can be performed. For this, one of the factors of influence (e.g., the response time of the banking service) is selected, a target value is specified, and another dependency analysis is launched. This identifies the more detailed dependencies that influence this specific factor of the overall process performance (e.g., one could find out that way that the response time of the banking service strongly depends on the type of the banking account).

VI. Experimentation

In this section we describe our prototype implementation of the framework and experimental results based on the example scenario.

A. Experiment Setup and Implementation

We have implemented the scenario as presented in Section II using a Java-based prototype. We use Apache ODE as our business process execution engine. ODE is open source software, and implements the WS-BPEL standard for Web service orchestration. One of the features of ODE is that it can trigger execution events (e.g., `ActivityExecStartEvent`), which we use as process events as described in Section III. The eventing features of ODE demand for a JMS implementation to take care of the transportation of events to subscribers (the monitoring tool in our case). We chose to use the open source message queue Apache ActiveMQ[4], however, any other JMS implementation could be used as well. The purchase order process has been implemented as a WS-BPEL process which interacts with six Web services including the client of the process. These Web services have been implemented in Java using Apache CXF[5] and simulate certain influential

[4]http://activemq.apache.org/
[5]http://cxf.apache.org/

factors. One can, for example, configure the response time, availability, and outputs of a service over time and dependant on business process data. The metrics database is implemented as a standard MySQL[6] database. Because of the limited size of the scenario we did not use advanced features such as clustering or load balancing. The Monitoring Tool for evaluation of PPMs has been implemented in Java as described in [5]. We have additionally implemented support for correlation of PPMs and QoS metrics. The Dashboard component is implemented as an standalone Swing application. The process analyzer is a standalone Web service, which is accessible over a RESTful interface. The foundation of this component is the WEKA toolkit[7], which implements many high-quality machine learning schemes, including the decision tree based classifiers that we used in this paper. We have transparently integrated WEKA into our process analyzer component using the WEKA Java API. Finally, we have implemented a simple QoS monitor, which can non-intrusively check the availability of Web services through periodic polling. This QoS data is again provided to the monitoring dashboard through a RESTful interface. For experimentation, we have deployed all these components on a single desktop PC, mainly to prevent external influences such as network latency to influence our experimentation results. However, the scenario is designed in such a way that physically distributed experiments can be run without any modifications.

B. Experimental Results

The procedure of experimentation is as follows. We create a configuration which simulates certain influential factors and define a set of potential influential metrics. We then execute the process a certain number of times (100, 400, and 1000 times) by triggering the process using a simulated client. During execution, the process is monitored and metrics are saved in the metrics database. We then perform dependency analysis of the KPI and compare the result of the generated dependency tree with our configured influential metrics. In the following we present the results of two experimental runs. For both of them, we have used the same configuration consisting of the KPI *Order Fulfillment Lead Time* and a set of 31 potential influential factors (a subset is shown in Table I).

For the first run, we have created a configuration which simulates the following factors: (i) the warehouse availability check (*order in stock*) returns a negative result for certain *product types* based on certain probabilities; *order in stock* is an important influential factor of the overall duration of the process as it decides whether products have to be requested from suppliers which increases the overall process duration substantially (ii) supplier 1 has in average a higher than expected *supplier delivery time*; (iii) average *shipment delivery time* is high in relation to the overall duration of the process instance. Based on this configuration, we expect the KPI to be mainly influenced by *order in stock*, *product type*, *supplier 1 delivery time*, and *shipment delivery time*. Other metrics (in particular response times of services) also influence the KPI value, but in a marginal way.

The generated decision tree is shown in Figure 7a. It has been generated using J48 (the WEKA implementation of C4.5 [9]) based on 100 process instances. The most influential factor is the *shipment delivery time*; if it is above 95 time units all process instances lead to KPI violations ("red"), otherwise they depend further on the *order in stock* metric and *supplier 1 delivery time*. The leaves of the tree show the number of instances which are classified as "red" or "green".

The dependency tree shows three of the four influential factors we have configured. Interestingly, the fourth factor, the *product type*, is not shown. The reason for this is that *product type* directly influences *order in stock*, which again influences the KPI value which is shown in the tree; as both metrics influence the KPI value in the same way, only one of them is shown in the tree. This particular result is unsatisfactory, as it hides the root cause, namely *product type* in this case. The user can deal with this problem using two approaches: (i) he can drill down and request the analysis of the *order in stock* metric. A second tree is generated which explains when ordered products are not in stock as shown in Figure 7b. This tree now clearly shows how the unavailability depends on *product type* and *ordered product quantity*. (ii) The user can also remove the *order in stock* metric temporarily from the analyzed metric set. Now, the algorithm will search for alternative metrics which classify the instances in a similar way as *order in stock*. In that case, as shown in the experiments, the algorithm finds and displays *product type* in the tree (not shown in the figure).

Table II shows the more detailed results of the first experimental run. We have experimented with two algorithms: J48 (based on C4.5 [9]) and ADTree (alternating decision tree [10]). Both of them show very similar results concerning the displayed influential metrics. Typically there is only one or at most two (marginal) metrics which differ. For the same precision (correctly classified instances in the training set, as shown in the last column), the algorithms also generate trees of about the same size. The usage of parameters has lead to only marginal changes in our experiments (for example, J48 -U with no pruning). The only parameter that turned out useful in our experimentation was the "reduced error pruning" (J48 -R) [8] as it reduced the size of the tree, loosing accuracy only marginally. This parameter is useful as the experiments show that the tree is getting bigger (column "Leaves/Nodes") with the number of process instances. For example, J48 generated for 400 instances a tree with 11 nodes, for 1000 instances a tree with 18 nodes, while the precision improved only by 1%. In particular, when the tree gets bigger, factors are shown in the tree which have only marginal influence and thus make the tree less readable; column "Displayed Metrics" shows how many distinct metrics are displayed in the tree, the first number thereby depiciting the number of expected metrics. In the case of too many undesirable (marginal) metrics, one can try to improve the result by simply removing those metrics from the analyzed metric set and repeating the analysis. Finally,

[6]http://www.mysql.com/

[7]http://www.cs.waikato.ac.nz/ml/weka/

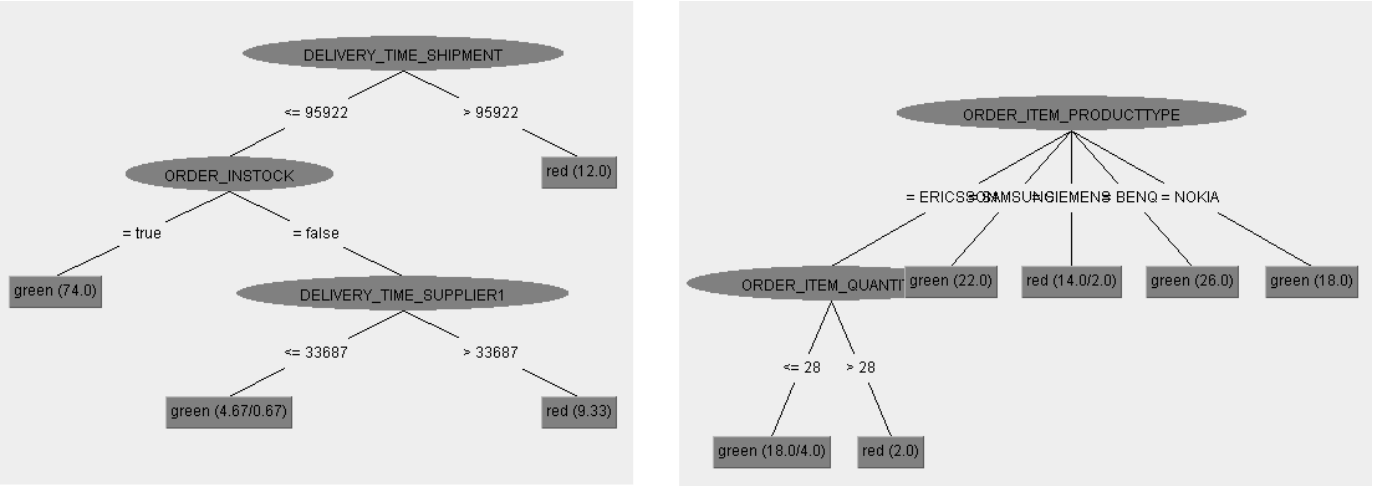

Fig. 7. Generated Trees for (a) *Order Fulfillment Time*, (b) *Order in Stock*

concerning the analysis duration, in our setting on a standard laptop computer a decision tree generation based on 1000 instances takes about 30 seconds.

Instances	Algorithm	Leaves/ Nodes	Displayed Metrics Expected/All	Correctly Classfied
100	J48	4/7	3/4	95,0 %
100	ADTree	11/16	2/4	98,0 %
400	J48	6/11	3/4	97,8 %
400	ADTree	17/26	4/5	99,0 %
1000	J48	11/18	3/6	98,8 %
1000	J48 -R	6/11	3/4	97,9 %
1000	J48 -U	13/22	4/9	99,2 %
1000	ADTree	19/28	3/6	99,4 %

TABLE II
EXPERIMENTAL RESULTS

For the second run, we have created a configuration which, among others, simulates QoS influential factors: (i) the warehouse Web service and the shipment Web service are unavailable with the probability of 15%; the BPEL process contains fault handlers when trying to invoke partner services; in case of unavailability it waits for a certain time frame and retries; we have defined *response time* metrics (measured based on process events) for each invoke-activity which ”include” the retries in case of unavailability (ii) the warehouse availability check (*order in stock*) returns now a negative result only with the probability of 5%; (iii) *shipment delivery time* is still very influential in relation to the duration of other activities of the process. Based on this configuration, we expect the KPI to be mainly influenced by the *availability* of warehouse Web service and shipment Web service, *order in stock*, and *shipment delivery time*.

The generated decision tree is shown in Figure 8a. It is a J48 tree based on 1000 instances using reduced error pruning. The tree shows the *response time warehouse*, *delivery time shipment*, and *order in stock* as the main influential factors. Completely missing, however, are the expected dependencies on the availability of the warehouse Web service and the shipment Web service. We suspect that *availability of warehouse Web service* is hidden by *response time warehouse*, which could be analyzed by drilling down. However, we take now another approach and perform an analysis of the KPI only in relation to availability metrics of all services involved in the process, i.e., we remove all other metrics from the analyzed metric set. Effectively, we analyze the impact of availability on the KPI. The result is shown in Figure 8b. It clearly shows that (only) availability of the shipment and warehouse Web services have an impact on the KPI value, as expected.

Overall, we can draw the following conclusions from the experiments. In general, the generated trees show the expected influential metrics in a satisfactory manner. As expected, the non-parametric nature of decision tree algorithms makes dependency analysis produce suitable results “out of the box”. We argue that this renders our approach suitable for non-IT personnel. However, this claim has yet to be verified through real-life evaluation. Concerning the influential factors displayed in the tree, we have identified two problems: (i) as the tree gets bigger it contains often more metrics than expected, i.e. metrics which have only marginal influence and thus only “blur the picture”; in that case one can try to tune the algorithm by using, for example, reduced error pruning, or one can simply remove those metrics from the analyzed metric set and repeat the analysis; both techniques lead to more satisfactory results; (ii) the tree does not show some of the expected metrics: we have shown that this is often the case when there are “multi-level” dependencies between metrics; in that case further analysis (drill down) of lower-level metrics may help to find further influential factors. While in the general analysis case, the user does not need to have any special domain knowledge on metric dependencies, in the drilldown case, we assume that the user *suspects* that there could be further dependencies behind a

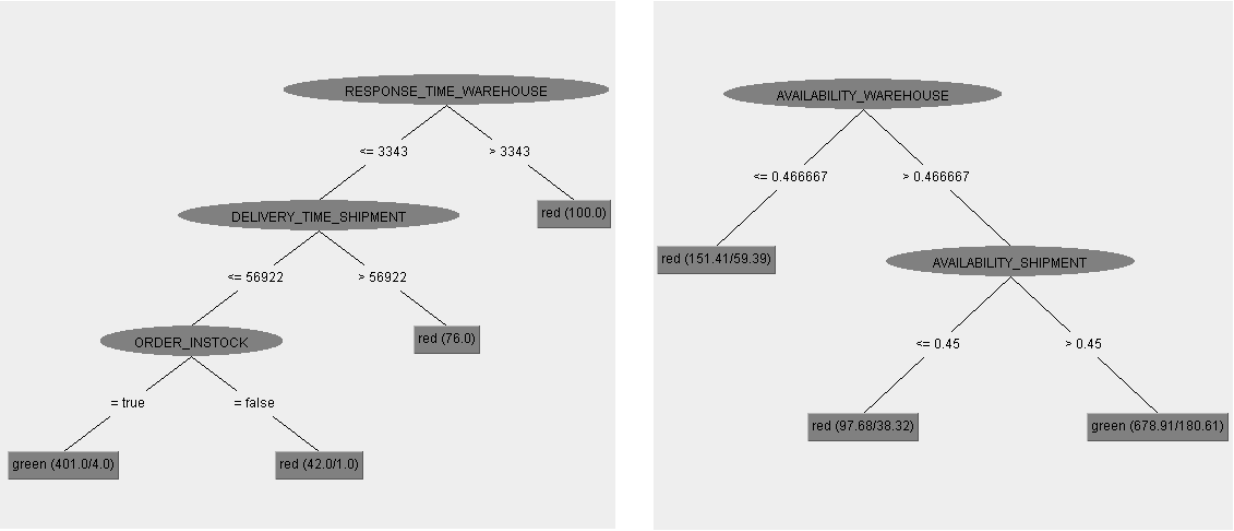

Fig. 8. Generated Trees for (a) All Factors, (b) Availability of IT Infrastructure

lower-level metric.

VII. Related Work

There are several approaches that deal with monitoring of service compositions. They differ mostly in monitoring goals, i.e., what is monitored, and the monitoring mechanisms. IBM's approach integrates performance management tightly into the business process lifecycle and supports it through its WebSphere family of products [11]. Thereby, process metrics are modeled and monitored based on events published by the Process Server BPEL engine. Our approach is similar to IBM's approach in that we use process events published by the process engine. We, however, also support monitoring of QoS metrics and automated dependency analysis. Baresi et al. [12] deal with monitoring of WS-BPEL processes focusing on runtime validation. The goal is thereby not to monitor process performance metrics, but to detect partner services which deliver unexpected results concerning functional expectations. Traverso et al. [13] describe a monitoring approach for WS-BPEL processes which supports run-time checking of assumptions under which the partner services are supposed to participate in the process and the conditions that the process is expected to satisfy. The approach supports also collecting statistical and timing information. All of these approaches have in common that they concentrate only on monitoring of business processes. In particular, they do not deal with QoS metrics integration and dependency analysis.

When it comes to the analysis aspect of service compositions, the idea of using dependency models for representing dependencies among different impact factors is not new and has been applied in connection with the monitoring of SLAs of service compositions in [14]. This approach focuses on dependencies of SLAs of overall service compositions on SLAs of composed services and analyzes reasons for SLA violations. Focusing on response time and cost metrics, the dependency relations and the impacts factors are identified at design time, and then later compared with monitoring results during runtime. In our approach, we do not model dependencies at design time, but solely construct the dependency model based on monitoring results using data mining. Most closely related to our work is the platform for business operation management developed by HP [15], as it supports both process monitoring, and analysis and prediction based on data mining. In [16] the authors give an overview and a classification of which data mining techniques are suitable for which analysis and prediction techniques. Thereby, also decision trees are mentioned as one possible technique, which is what we concentrate on in our approach. The platform presented allows users to define and monitor business metrics (not focused on WS-BPEL processes), perform intelligent analysis on them to understand causes of undesired metric values, and predict future values. Our approach is different in that we focus on SOA-based WS-BPEL processes, and explicitly integrate PPMs and QoS metrics for analysis purposes. We deal only with decision trees, but provide detailed experimental results. Another popular approach to process analysis is process mining. Process mining techniques operate on event logs provided by information systems and perform different kinds of analysis on them, in particular process discovery when there is no explicit process model a priori [17]. In our approach, we also operate on monitored "metric logs" during analysis phase, but focus on mining of metric dependencies using decision tree algorithms.

VIII. Conclusions

In this paper we have presented a framework that performs monitoring of both PPMs and QoS metrics of business processes runnning on top of a Service-Oriented Architecture. Besides providing up-to-date dashboard information about the current process performance, the main goal of our framework is to enable what we refer to as dependency analysis, i.e., an analysis of the main factors that influence the business process and make it violate its performance targets. The result of this analysis is represented as a decision tree. We have

presented experimental results which show that in general the generated decision trees provide explanations in a satisfactory manner, but in some cases further analysis has to be done. In that respect, we have shown how drill-down functionality and analyis based on different metric sets can influence the analysis result. One important advantage of our approach is that results are satisfactory without the need for extensive experimentation and parameter adaptation, which lets our approach seem feasible even for domain experts which are not IT-savy.

Our future work includes extending the framework presented here into various directions. Firstly, we plan to work on the runtime prediction of the outcome of process instances (i.e., whether the KPI is going to be violated or not) while they are still running. Basically, we can use the same techniques as for dependency analysis (however, we will in addition use regression trees). Secondly, we are working towards making use of the dependency analysis in the area of process adaptation – currently, dependencies are presented towards the human business analyst, who is then incorporating the gained knowledge back into the process, e.g., by exchanging service bindings. We currently think about a more automated feedback mechanism, which uses rule sets and predefined reactions to incorporate dependency knowledge back into the WS-BPEL process in a more automated way. One example would be service selection: if the dependency model of a process shows that the process outcome is sensitive to the response time of a service, then an expensive high-quality service is selected; if the response time is no important factor of influence a cheaper service is selected. Finally, we still need to test our approach in real-world settings, to further validate the claims that we have stated in this paper.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Communitys Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

REFERENCES

[1] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, vol. 11, 2007.

[3] OASIS, *Web Services Business Process Execution Language Version 2.0 – OASIS Standard*, 2007.

[4] J.-J. Jeng, J. Schiefer, and H. Chang, "An Agent-based Architecture for Analyzing Business Processes of Real-Time Enterprises," in *Proceedings of the 7th International Conference on Enterprise Distributed Object Computing (EDOC '03)*. Washington, DC, USA: IEEE Computer Society, 2003, p. 86.

[5] B. Wetzstein, S. Strauch, and F. Leymann, "Measuring Performance Metrics of WS-BPEL Service Compositions," in *The Fifth International Conference on Networking and Services (ICNS 2009), Valencia, Spain, April 20-25, 2009*. IEEE Computer Society, April 2009.

[6] S. Council, "Supply Chain Operations Reference Model Version 7.0," 2005.

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

[8] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[9] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan-Kaufmann, 1993.

[10] Y. Freund and L. Mason, "The Alternating Decision Tree Learning Algorithm," in *Proceedings of the 16th International Conference on Machine Learning (ICML '99)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 124–133.

[11] U. Wahli, V. Avula, H. Macleod, M. Saeed, and A. Vinther., *Business Process Management: Modeling Through Monitoring Using WebSphere V6.0.2 Products*. IBM, International Technical Support Organization, 2007.

[12] L. Baresi and S. Guinea, "Towards Dynamic Monitoring of WS-BPEL Processes," in *Proceedings of the 3rd International Conference of Service-Oriented Computing (ICSOC'05)*. Springer, 2005, pp. 269–282.

[13] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-Time Monitoring of Instances and Classes of Web Service Compositions," in *Proceedings of the IEEE International Conference on Web Services(ICWS'06)*, 2006, pp. 63–71.

[14] L. Bodenstaff, A. Wombacher, M. Reichert, and M. C. Jaeger, "Monitoring Dependencies for SLAs: The MoDe4SLA Approach," in *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC '08)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 21–29.

[15] M. Castellanos, F. Casati, M.-C. Shan, and U. Dayal, "iBOM: A Platform for Intelligent Business Operation Management," in *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, 2005, pp. 1084–1095.

[16] M. Castellanos, F. Casati, U. Dayal, and M.-C. Shan, "A Comprehensive and Automated Approach to Intelligent Business Processes Execution Analysis," *Distributed and Parallel Databases*, vol. 16, no. 3, pp. 239–273, 2004.

[17] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. on Knowl. and Data Eng.*, vol. 16, no. 9, pp. 1128–1142, 2004.