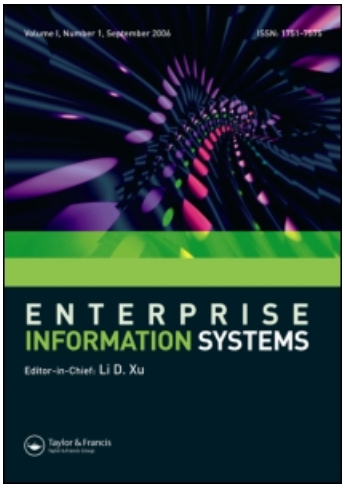

Enterprise Information Systems
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t748254467

Identifying influential factors of business process performance using dependency analysis

Branimir Wetzstein[a]; Philipp Leitner[b]; Florian Rosenberg[c]; Schahram Dustdar[b]; Frank Leymann[a]
[a] Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany [b] Distributed Systems Group, Vienna University of Technology, Vienna, Austria [c] CSIRO ICT Centre, Canberra, ACT, Australia

Online publication date: 13 December 2010

To cite this Article Wetzstein, Branimir , Leitner, Philipp , Rosenberg, Florian , Dustdar, Schahram and Leymann, Frank(2011) 'Identifying influential factors of business process performance using dependency analysis', Enterprise Information Systems, 5: 1, 79 — 98
To link to this Article: DOI: 10.1080/17517575.2010.493956
URL: http://dx.doi.org/10.1080/17517575.2010.493956

Enterprise Information Systems
Vol. 5, No. 1, February 2011, 79–98

Taylor & Francis
Taylor & Francis Group

Identifying influential factors of business process performance using dependency analysis

Branimir Wetzstein[a]*, Philipp Leitner[b], Florian Rosenberg[c], Schahram Dustdar[b] and Frank Leymann[a]

[a]*Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany;* [b]*Distributed Systems Group, Vienna University of Technology, Vienna, Austria;* [c]*CSIRO ICT Centre, GPO Box 664, Canberra, ACT 2601, Australia*

(*Received 1 March 2010; final version received 13 May 2010*)

We present a comprehensive framework for identifying influential factors of business process performance. In particular, our approach combines monitoring of process events and Quality of Service (QoS) measurements with dependency analysis to effectively identify influential factors. The framework uses data mining techniques to construct tree structures to represent dependencies of a key performance indicator (KPI) on process and QoS metrics. These dependency trees allow business analysts to determine how process KPIs depend on lower-level process metrics and QoS characteristics of the IT infrastructure. The structure of the dependencies enables a drill-down analysis of single factors of influence to gain a deeper knowledge why certain KPI targets are not met.

Keywords: process performance monitoring; service composition; KPI; QoS; data mining; decision tree

1. Introduction

Enterprise systems typically implement core business processes by using different design methods and process technologies. This requires an effective alignment between business processes and IT systems to be competitive on the market. Business process management (BPM) provides a set of methods, techniques and tools for modeling, executing and analysing business processes of an organization (Weske 2007). Recently, BPM has been supported by an integrated set of tools supporting the process of lifecycle in an unified manner. Thereby, business analysts create a process model, which is then refined by IT engineers to an executable model. The executable business process model is deployed to a process engine, which executes it by delegating tasks to both humans and services. For example, business processes can be modeled using BPMN (Business Process Modeling Notation) (OMG 2009) and then mapped to WS-BPEL (Web Service Business Process Execution Language) (OASIS 2007) for execution. Adopting WS-BPEL typically implies the service oriented architecture (SOA) paradigm (Papazoglou *et al*. 2007). In SOA, core functionality is exposed as services and orchestrated into composite services, which constitute the enterprise business processes.

*Corresponding author. Email: wetzstein@iaas.uni-stuttgart.de

ISSN 1751-7575 print/ISSN 1751-7583 online

DOI: 10.1080/17517575.2010.493956
http://www.informaworld.com

Monitoring of business goals and timely measurement of business process performance are important aspects of the BPM lifecycle. Such goals are typically expressed by defining a number of key performance indicators (KPIs) and their target values. For example, in a purchase order process an example KPI could be the 'order fulfillment lead time' with a target value 'less than 3 days'. Whenever a KPI does not meet its target value, a business analyst wants to know what when wrong, and how to address the issue. This task is supported by business activity monitoring (BAM) technology. It enables continuous, near real-time monitoring of processes based on eventing (Wahli *et al*. 2007). However, in BAM the focus is currently set on the 'what' rather than the 'why' question. BAM does not reveal the 'hidden' factors that caused deviations from target KPI values. In this article, we specifically look a two important factors. On the one hand we consider process performance metrics (PPM), which define metrics based on process runtime data (Wetzstein *et al*. 2009), e.g. 'the 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). On the other hand, we also take technical parameters, such as quality of service (QoS), e.g. the availability of the process engine or the response time of used Web services) into account. In sum, a very large number of possible causes exists, and it is rarely obvious even to domain experts which of those possible factors of influence is most important for business process performance, and what dependencies exist between those factors. Unrevealing those dependencies in a structured way increases a chance of detecting them, and subsequently trigger corrective actions.

We propose a comprehensive end-to-end framework for monitoring and analysis of the performance of business processes based on WS-BPEL. In particular, we use dependency analysis, i.e. data mining based analysis of PPMs and QoS metrics, with the goal of discovering the main factors of influence of process performance. These factors are represented as easy-to-interpret decision trees (dependency trees). We present the basic concepts of our analysis framework, and provide experimental results based on a purchase order scenario. In addition, we identify cases where dependency trees do not show all expected results, and explain strategies to deal with these problems.

The rest of the article is organized as follows. Section 2 presents an illustrative scenario for explaining the basic concepts and the core research issues addressed in this article. Section 3 explains the main ideas of our framework for runtime monitoring and dependency analysis. Section 4 details the monitoring of influential factors followed by the description of the dependency analysis in Section 5. Section 6 describes the implementation of our prototype based on the scenario and presents some experimental results. Section 7 discusses important related work and Section 8 presents some concluding remarks and highlights the most important future work.

2. Scenario

In this section, we present a scenario which will be used throught the remainder of this article. Additionally, we have implemented and used this scenario for experimentation purposes. The scenario consists of a customer, a reseller, its two suppliers, a banking service, and a shipping service. The reseller offers certain products to its customers. It holds certain products in stock, and orders others directly from suppliers. The customer

sends a purchase order request with details about the required products and required amounts to the reseller. The latter checks whether all products are available in stock. If some products are not, they are ordered externally. Note that the second external 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 necessary, for the supplier to deliver the needed products. When all products are available, the warehouse packages them and hands them over to shipping. The shipping service delivers the order to the customer, and finally notifies the reseller about the shipment. In parallel to packaging and shipment, the payment subprocess is performed. The customer decides on the payment style and specifies 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*, i.e. duration from receiving the customer order until shipment is received by the customer, as defined in the supply chain operations reference model (SCOR)[1].

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 ordered products. We have provided an (incomplete) list of potential factors of influence for the KPI from our scenario in Table 1. Factors include simple facts from the business process instance, such as a customer 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, no calculation formula is required. 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. For example, 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 article. Additionally, 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 which of these factors actually influence the KPI most, and what the structure of the dependencies between sfactors is (i.e. some factors are in turn influenced by others, such as the duration of the payment subprocess which is

Table 1. Potential influential factors of KPI performance.

Name	Type	Calculation formula	Range
Customer ID	PPM		$\{\text{Customer}_1, \text{Customer}_2, \dots\}$
Product type	PPM		$\{\text{Product}_1, \text{Product}_2, \dots\}$
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}	$[0;\infty]$

influenced by service response times). These questions are not answered sufficiently by today's BAM dashboards – they can only provide status 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.

3. Framework overview and methodology

In this section, we give an overview of our framework and the corresponding methodology for monitoring and analysing factors of influence of business process performance. Our approach can be can be described in terms of four phases: design, deployment, monitoring and analysis phase.

In the *design* phase, the user specifies one or several KPIs for an already defined executable WS-BPEL process. KPIs are key metrics with target values which are to be measured for this process. For the specified KPIs, in the next step a set of further metric definitions (PPMs and QoS metrics) needs to be specified. To support this step we provide tooling to semi-automatically generate a number of potential metrics for monitoring. The metric definitions model will serve as input to KPI dependency analysis later in the analysis phase. All metric definitions together constitute the process metrics definition model (PMDM).

In the the *deployment* phase, the PMDM model is deployed to the monitoring infrastructure and *monitoring* is started. The monitoring infrastructure consists of several components: an event listener in the BPEL engine which publishes resource events during process execution, a QoS monitor which publishes QoS events as result of QoS measurement, and a CEP engine which recursively aggregates those events to complex events thus calculating PPMs and QoS metrics. PPMs and QoS metrics are saved in a metrics database and displayed in related dashboards.

When the user is interested in performing a dependency *analysis* of KPIs, the process analyser gathers the needed metric data from the metrics database, prepares it for mining, and uses a decision tree algorithm to generate a dependency tree, which depicts 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.

A high-level overview of the main components which support the described methodology is given in Figure 1. In our framework we distinguish three different layers.

In the *process runtime* layer, a WS-BPEL business process is defined and executed. It orchestrates a set of Web services, and is exposed itself as a Web service to service requesters. In the *monitoring layer*, information about the running business process and the services it interacts with is collected to monitor KPIs, PPMs, and QoS metrics. 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 analysed by the process analyser component.

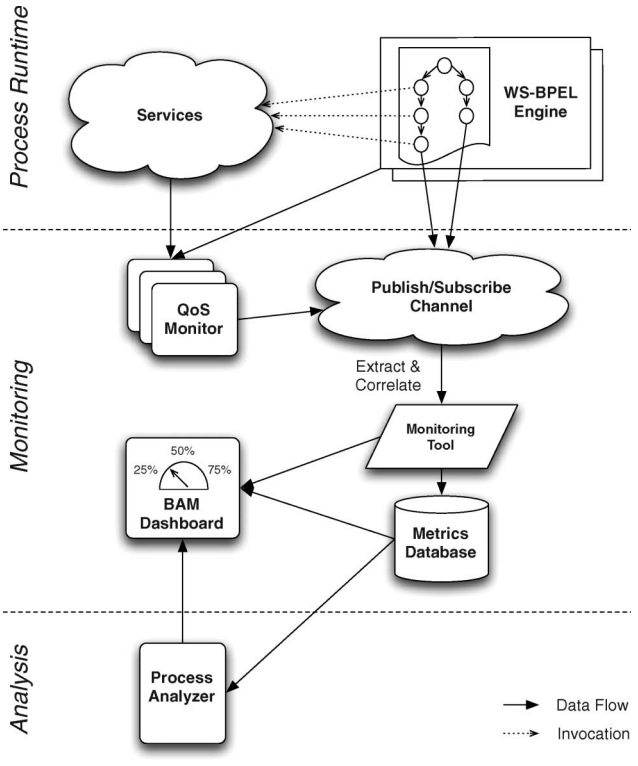

Figure 1. Monitoring and analysis framework overview.

4. Monitoring of influential factors

The goal of monitoring in our approach is (1) to obtain KPI values and check whether they meet specified targets and (2) to provide metrics for factors which could potentially influence the KPI performance and are thus input to later dependency analysis. In this phase we deal with two challenges:

- *Monitoring both on process level and service level:* The business processes we focus on in this paper are implemented as WS-BPEL 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. This is why we support in our approach both PPMs and QoS metrics and their correlation (Sections 4.1 and 4.2). supported by different monitoring mechanisms.
- *Semi-automated creation of metric definitions:* One input to our approach is a comprehensive set of metrics which should be monitored. Even if the user knows which metrics she wants to monitor, the manual creation of these definitions can be a tediuos and error-prone task. Therefore, there is a need for a semi-automated approach to creation of potentially interesting
- metrics (Section 4.3), which supports the user by proposing metrics which may be important to consider for monitoring.

4.1. Monitoring of process performance metrics

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. We distinguish between resource events and complex events. Resource event definitions are specified based on the process instance resources and complex events are defined (recursively) based on other events. Those events are used for PPM calculation.

4.1.1. WS-BPEL resource events

A resource event definition specifies the following three elements:

- *Monitored resource:* First, we have to specify which process resource should be monitored and for which state of the resource the event should be published. The resource is identified by pointing to the corresponding WS-BPEL elements. Monitored resources we are interested in are the instances of the WS-BPEL process, activity, scop, and variable. The state models (e.g. *started, completed, terminated, compensated* and corresponding transitions) for these resources are not standardized. In our work, we use the state models defined in Karastoyanova *et al.* (2006).
- *Process data:* Optionally, one can specify which process data (defined as WS-BPEL variable) is to be part of the event. The data is read when the event is published.
- *Target message queue or pub/sub topic:* Finally, one has to specify a message queue or a pub/sub topic to which the event is to be published.

Listing 1 shows a *resource event definition* for the `OrderReceivedEvent` resource event. It is specified by pointing to the `Receive PO` activity in the reseller WS-BPEL process model. The event is to be published when the corresponding activity is `completed`. In addition, the event should contain the data from the `purchaseOrder` variable. It is published to the specified message queue. The resource identification (specified in the `monitoredResource` element) will result at process runtime in corresponding resource identifiers (i.e. process instance ID, `ppid`, and an activity instance ID, `aiid`, which is needed if an activity is executed in a loop). These identifiers are transported as part of the event, and are needed for event correlation.

```
1  <resourceEventDefinition name="OrderReceivedEvent">
2   <monitoredResource
3     process="reseller:PurchaseOrderProcess"
4     scope="process"
5     activity="reseller:ReceivePO"
6     state="completed"/>
7   <data>
8    <processVariable name="order" variable="purchaseOrder"/>
9   </data>
10  <publish>
11   <queue name="PurchaseOrderProcess.ResourceEvents"  />
12  </publish>
13 </resourceEventDefinition>
```

Listing 1. Resource event definition.

This resource event definition is deployed to an event listener, which receives internal events by the process engine, filters only the events of the specified resource and includes process data if needed. The listener then publishes the event to the queue or topic, as specified in the definition.

4.1.2. Complex events and PPMs

Complex events are specified by correlating and aggregating existing events (resource and complex events). Event correlation and aggregation is a well-known topic in the area of complex event processing (CEP), and there are different languages available for the specification of complex events (Luckham 2002). In our case, we have decided to use the language of ESPER[2], which is the CEP implementation we have used in our prototype (Section 6.1). However, alternatively, any other language could be used instead. Note that we use the term *complex event* for an event which results from using a CEP statement over one or more events. We do not further distinguish between more fine-grained meanings of complex, composite, and derived events as in some other works.

A complex event definition specifies the following three elements:

- *Consumed events:* First, we have to specify the names of the source queue(s) and/or topic(s) from which the events should be aggregated.
- *Event aggregation statement:* This is a CEP statement which correlates and aggregates the consumed events to a new event. In case the created complex event calculates a metric, the metric value is saved in the metricValue field of the event.
- *Target message queue or pub/sub topic:* Finally, one has to specify a message queue or a pub/sub topic to which the new event is to be published.

In Listing 2 we define a complex event `OrderFulfillmentTimeEvent` (lines 1–14), which contains the corresponding metric value in the attribute `metricValue`. In addition it contains the attribute `unit` and the process instance identifier. The metric value is calculated by correlating two events already defined, namely `OrderReceivedEvent` (Listing 1) and `ReceivedDeliveryNotificationEvent`. These

```
1  <complexEventDefinition name="OrderFulfillmentTimeEvent">
2   <consume><queue name="PurchaseOrderProcess.ResourceEvents" /></consume>
3   <eventAggregation>
4    <statement><![CDATA[
5     SELECT abs(b.timestamp - a.timestamp) AS metricValue, "ms" AS unit, a.piid AS piid
6     FROM PATTERN [EVERY a = ResourceEvent(name="OrderReceivedEvent")
7      -> b = ResourceEvent(name="ReceivedDeliveryNotificationEvent"
8         AND piid = a.piid) ]
9    ]]><statement>
10  </eventAggregation>
11  <publish>
12   <topic name="PurchaseOrderProcess.metrics" />
13  </publish>
14 </complexEventDefinition>
15
16 <ppm id="OrderFulfillmentTimeMetric">
17  <name>Order Fulfillment Time</name>
18  <unit>ms</unit>
19  <calculation>
20    <event name="OrderFulfillmentTimeEvent" />
21  </calculation>
22 </ppm>
```

Listing 2. Complex event for PPM computation.

events are correlated based on the piid, and then their timestamps are subtracted. The result event is published to the corresponding topic. Note that obviously such a definition results in one result event per process instance, i.e. an event stream. Finally, we can define the PPM OrderFulfillmentTimeMetric (lines 16–22), which is calculated based on the corresponding previuosly defined complex event. Complex event definitions are deployed to the CEP engine. The CEP statement specified in the definition is registered in the CEP engine, the consumed events needed are retrived from the queues or topics as specified. There is a special listener for complex events for which a PPM has been defined. Such a listener retrieves the complex events from the topic, stores the metric value in the metric database and updates the dashboard view if needed. The record saved in the metric database consists of the metric name, metric value and the corresponding resource identifiers (piid and aiid).

4.2. *Monitoring of QoS metrics*

In our context, QoS can be measured in three different ways: (1) probing by a separate QoS monitor, such as the one described in (Rosenberg *et al.* 2006), (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. Response time is estimated based on the duration of the corresponding WS-BPEL invoke activity.

4.2.1. *QoS events*

The QoS monitor polls the corresponding endpoints and emits QoS events which contain information on the monitored services current availability. Listing 3 shows the definition of a QoS event which is to be provided by the QoS monitor. We define that the QoS monitor should poll the corresponding endpoint with a certain testFrequencyPerMinute. The QoS monitor will thus emit 20 ProcessEndpointAvailableEvents per minute specifying whether the process endpoint was available to the specified queue.

4.2.2. *Correlation of process events and QoS events*

QoS events as defined above are not enough to evaluate the process instance availability QoS metric. They do not contain information on process instances which

```
1  <QoSEventDefinition name="ProcessEndpointAvailableEvent">
2   <availabilityProbe>
3     <endpoint>
4       http://localhost:8082/.../poProcess?wsdl
5     </endpoint>
6     <testFrequencyPerMinute>20</testFrequencyPerMinute>
7   </availabilityProbe>
8   <publish>
9    <queue name="PurchaseOrderProcess.QoSEvents"  />
10  </publish>
11 </QoSEventDefinition>
```

Listing 3. QoS event definition.

have been running while the availability check has been performed. Thus, these events have to be *correlated* with corresponding process events, in this case the start and end (resource) event of a process instance. In Listing 4, we construct a new complex event `ProcessInstanceAvailableEvent` which results from each QoSEvent `ProcessEndpointAvailableEvent` (as defined above). The latter is published while that process instance has been executed, namely between the WS-BPEL resource events `OrderProcessingStartedEvent` and `OrderProcessingCompletedEvent`. The resulting complex event `ProcessInstanceAvailableEvent` now contains the `piid` of the process instance. It can be used for calculating the availability metric for a process instance (not shown for space reasons) by counting those events with `available = true` and dividing that number through all events. Note that while QoS metric calculations result in rather long statements, they do not have to be written manually by the user, but can be generated as will be explained in the next section.

4.3. *Generation of metric definitions*

The core of our approach is the availability of a meaningful and complete set of PPM and QoS metrics for monitoring and analysis. Unfortunately, defining these metrics manually is rather cumbersome and time-consuming. We have therefore devised a rule-based approach to identify potential metrics in a semi-automated manner. A human operator can then take over parts of or all metrics from this generated temporary metric set, as well as define entirely new metrics.

Our general approach is to provide a number of rules, which identify elements of a WS-BPEL process (such as invoke activities, branches or loops), extract some basic information from the element (such as the endpoint or service name for invoke activities) and generate one PPM or QoS metric definition per element. This is done by filling an XML template with the information extracted from the element. We have sketched a template for availability metrics in Listing 5. In this templates, variables are indicated by `[$]{...}` (e.g. `[$]{endpoint}`). The concrete values for these variables are retrieved from the WS-BPEL element. We used XML stylesheet transformations (XSLT) to implement metric generation in our tool.

We currently support rules to generate the following metrics:

- For every WS-BPEL invoke activity we generate an availability metric (using the template sketched in Listing 5).
- For every WS-BPEL invoke activity which is not part of a loop (i.e. which is executed 0 or 1 times in every instance) we generate a metric representing the

```
1  <complexEventDefinition name="ProcessInstanceAvailableEvent">
2   <consume>...</consume>
3   <eventAggregation>
4    <statement><![CDATA[
5     SELECT begin.piid as piid, q.available as available from PATTERN [
6      begin=ResourceEvent(name="OrderProcessingStartedEvent") -> q=QoSEvent(name="
           ProcessEndpointAvailableEvent")
7      UNTIL end=ResourceEvent(name ="OrderProcessingCompletedEvent" AND piid = begin.piid)
8    ]]><statement>
9   </eventAggregation>
10  <publish>
11   <topic name="PurchaseOrderProcess.metrics" />
12  </publish>
13 </complexEventDefinition>
```

Listing 4. Correlation of process and QoS events.

```
<QoSEventDefinition name="${service}AvailableEvent">
  <availabilityProbe>
    <endpoint> ${endpoint} </endpoint>
    <testFrequencyPerMinute>20</testFrequencyPerMinute>
  </availabilityProbe>
  <publish>
    <queue name="${processName}.QoSEvents" />
  </publish>
</QoSEventDefinition>
```

Listing 5. Availability metrics template.

execution time of the activity (i.e. the time between starting and finishing the activity).

- For every WS-BPEL invoke activity which is part of a loop (i.e. which is potentially executed several times in an instance) we generate a metric representing the average execution time of the activity. Additionally, we generate metrics representing the number of times the activity has been executed in this instance, and the minimum and maximum execution time of the activity.
- For every loop, we generate a metric representing the number of iterations in this instance.
- For every branching activity, we generate a metric representing the branch that has been executed.
- Finally, we generate a metric representing the callback time of every asynchronous activity in the WS-BPEL process (e.g. the duration between ordering via an external supplier and the callback from the supplier signaling that the order has been shipped).

Each of these rules can be individually turned on or off. In addition, every generated metric can be discarded after generation. However, please note that this approach can only provide a partial list of metrics, since some domain-specific metrics (e.g. a customer identifier) cannot be identified this way. These domain-specific metrics still need to be provided by a human domain expert manually.

5. Analysis of influential factors

Dependency analysis uses historical process data to determine the most important factors that dictate whether a process instance is going to violate its KPIs. The output of dependency analysis is an easily visualizable model of the internal dependencies of the business process. It identifies the most important factors of influence of process performance. In our work, we generally use decision trees (Witten and Frank 2005) as dependency model. We refer to these tree models as *dependency trees*, because they represent 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. 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 data). Decision trees use a 'divide and conquer' approach to learning concepts. They iteratively construct a tree of decision nodes, each consisting of a test, such as whether a given numerical attribute is smaller than a given threshold. Leaf

nodes in the tree typically represent a classification to a category. In our case, only two such categories exist (KPI has been violated, or not), i.e. dependency trees are binary decision trees. One big advantage of decision tree algorithms, espcially so in our context, is their non-parametric nature. They need only a very limited set of parameters (in the simplest case none) as input, and can therefore be expected to provide useful results from the first run, without the need for extensive experiments with different parameter sets, which we argue makes our approach suitable for business analysts, who are generally no experts in data mining. However, please note that in our apprach the used mining algorithms can still be customized by data mining savy users, which may lead to better results in many cases.

Our main motivation for using decision trees is that they are easy to depict graphically. This is important if results should be presented to non-experts. An additional benefit is that many well-researched algorithms to construct decision trees from data exist, such as the C4.5 (Quinlan 1993) or the alternate decision tree, ADTree, (Freund and Mason 1999) algorithms. For tree learning, we use 10-fold cross validation, which is a standard technique in data mining to avoid having to split the input data into training, test and validation sets. Additionally, cross validation allows to estimate the classification error of the decision tree. The classification error enables 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.

5.1. *Creation of dependency trees*

The general process of dependency analysis is a 4-step-procedure. We have depicted these steps in Figure 2.

The analysis consists of the following steps:

(1) The first step is KPI selection and optional adjustment of analysis parameters, which is done by a human business analyst. She chooses a KPI (from the PMDM) she wants to analyse. Optionally she 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 analysed (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.

(2) The second step is creation of the training set, which is 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 2 (in Step 2). 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.

(3) The third step is decision tree learning. Basically, in this step a decision tree is trained from the training set using a standard decision tree algorithm like C4.5 or ADTree.

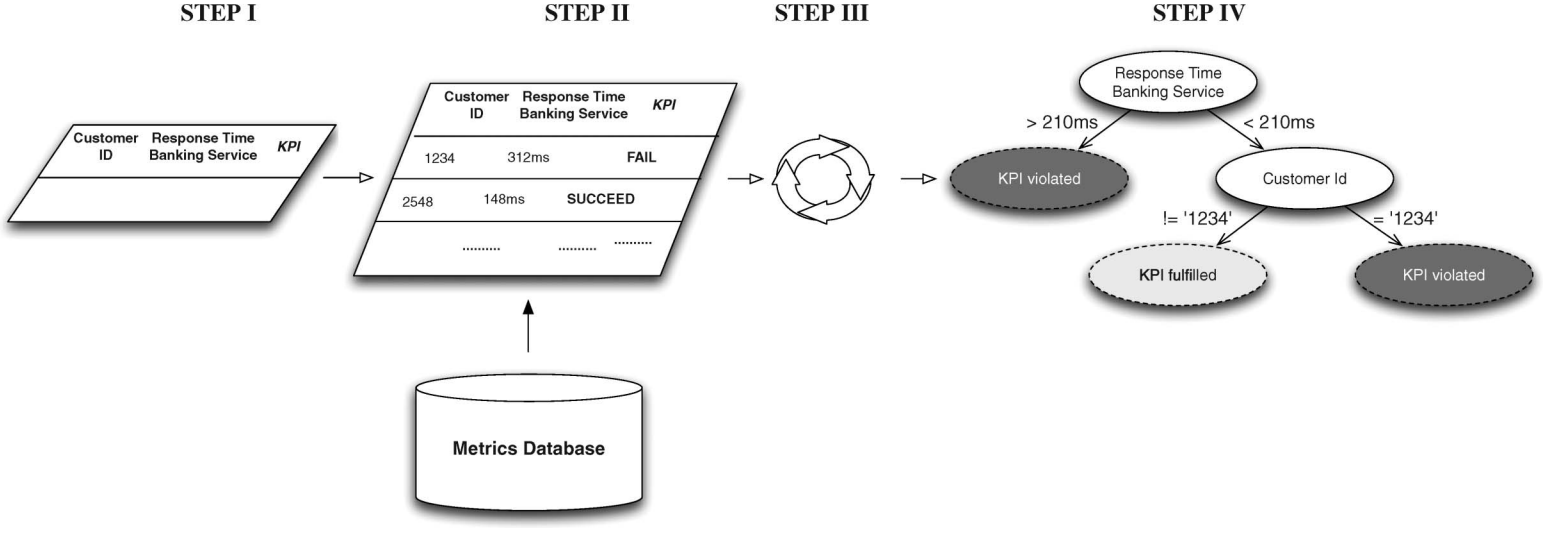

Figure 2. Dependency analysis.

(4) Finally, in the fourth step the resulting tree is displayed in the dashboard as outcome of the dependency analysis.

As a result of these four steps, a dependency tree is constructed as shown in Figure 2 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 service's 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. Considering the example in Figure 2, a business analyst can take the corrective action to replace the 'Banking Service' with 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 important factors of influence, a further 'drill-down' analysis can be performed. For this, one of the factors (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).

6. Experimentation

This section describes our prototype implementation of the framework and experimental results based on the example scenario.

6.1. Implementation and experiment setup

Our prototype uses Apache ODE as business process execution engine. ODE is open source software, and implements the WS-BPEL standard for Web service orchestration. We have implemented an ODE event listener for publishing of WS-BPEL resource events as described in Section 4.1. The events are published by the listener to JMS-based queues and topics. We chose to use the open source Apache ActiveMQ[3] JMS implementation. We have also implemented a simple QoS monitor, which can non-intrusively check the availability of Web services through periodic polling. It publishes QoS events to corresponding JMS queues or topics (as described in Section 4.2). Complex event processing has been implemented based on ESPER.[4] Complex event definitions (Section 4.1) are deployed as ESPER statements. An inbound and an outbound JMS adapter have been implemented for access to JMS queues and topics. For storing metrics in the database, a special event listener has been implemented which subscribes only to those complex events which are used for metric calculation. Metrics are saved in a standard MySQL[5] database. Because of the limited size of the scenario we did not use advanced features such as clustering or load balancing. The dashboard component is implemented as an standalone swing application. The process analyser is a standalone Web service, which is accessible

over a RESTful Web service interface. The foundation of this component is the WEKA toolkit[6], which implements many high-quality data mining schemes, including the decision tree based classifiers that we used in this article. We have transparently integrated WEKA into our process analyser component using the WEKA Java API.

We have implemented the scenario from Section 2 as a WS-BPEL process interacting with six Web services. Additionally, we provide a simple Java client for this process. The Web services have also been implemented in Java, using the Apache CXF[7] framework, and simulate certain influential factors. For example, the user can configure the response time, availability, and outputs of a service over time and dependent on business process data.

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.

6.2. *Experimental results*

The experimentation has been performed using the purchase order process already discussed in Section 2. We define *Order Fulfillment Lead Time* as the KPI to be analysed and create a set of 31 potential influential factors. The corresponding metric definitions have been generated as described in Section 4.3. In addition, we have manually defined several domain-specific metrics such as *product types, number of ordered products, customer type, order in stock*, etc.

The experimentation is performed as follows. We create a setting which simulates certain influential factors by configuring the behaviour of services invoked by the process. We then trigger the execution of process instances using a test client. During process instance execution, the previously specified metrics are measured and saved in the metrics database. Finally, we perform the dependency analysis on the KPI and compare the result of the created dependency tree with the configured influential metrics.

The first configuration we have created simulates the following factors: (1) the warehouse service returns a negative result for certain *product types* based on given probabilities; this should have a major impact on process duration as some products have to first be ordered from suppliers; (2) 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, and can vary. On the basis of this configuration, we expect the dependency tree to show that the KPI is mainly influenced by *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 3. It has been generated using J48 (the WEKA implementation of C4.5) based on 400 monitored process instances. It shows that when *shipment delivery time* was above 96 time units all process instances lead to KPI violations ('red'), otherwise the outcome of the KPI depends further on the *order in stock* metric, then again on *shipment delivery time, supplier 1 delivery time* and finally *response time supplier 1*. The leaves of the tree show the number of instances which are classified as 'red' or 'green'. That means, for example, that 267 process instances (out of 400) had the *shipment delivery time* below 96 time units and *order in stock* = true, and met the KPI target value.

The dependency tree shows two of the three influential factors we have configured and expected (*shipment delivery time* and *supplier 1 delivery time*). Interestingly, the third factor, the *product type*, is not shown. Instead *order in stock* has been chosen by the decision tree algorithm. The reason for this is that *product type* directly influences *order in stock*, which again influences the KPI value; 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: (1) he can drill-down and request the analysis the *order in stock* metric. A second tree is generated which explains when ordered products are not in stock as shown in Figure 4. This tree now clearly shows how the unavailability of ordered products in stock depends on *product type* and *ordered product quantity*. (2) The user can also remove the *order in stock* metric temporarily from the analysed metric set. Now, the algorithm will search for alternative metrics which classify the instances in a similar way as *order in stock*.

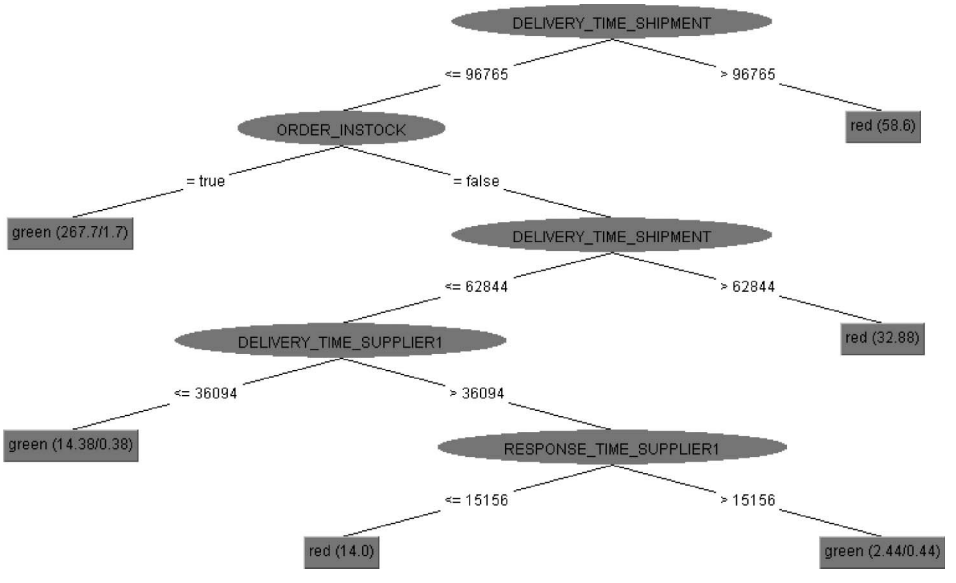

Figure 3. Generated tree for *order fulfillment time*.

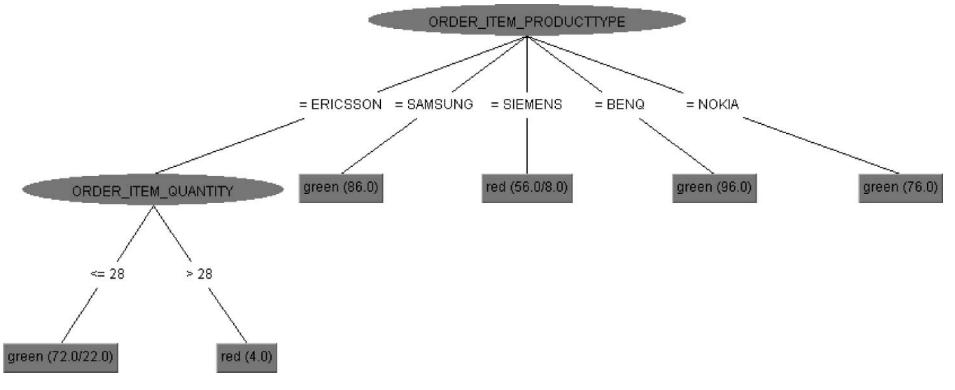

Figure 4. Generated tree for *order in stock*.

Note however that while in the general analysis case, the user does not need to have any special domain knowledge on metric dependencies, in the described two approaches, we assume that the user *suspects* that there could be further dependencies behind a lower-level metric.

Another available analysis option is to selectively choose the set of potential influential factors to be analysed. Assuming that we are interested in how the KPI depends on the availability of services and infrastructure. In that case, we reduce the potential influential factor metric set to just the availability metrics. To evaluate this, we have created a configuration which simulates that the warehouse service and the shipment service are unavailable with the probability of 15%. The WS-BPEL process contains fault handlers when trying to invoke partner services; in case of unavailability it waits for a certain time frame and retries. Thus, the unavailability of a service will impact the overall duration of the process. (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. On the basis of this configuration, we expect the KPI to be mainly influenced by the *availability* of the warehouse and shipment Web services, *order in stock* and *shipment delivery time*. The generated decision tree is shown in Figure 5. It is a J48 tree based on 1000 instances. It clearly shows that (only) availability of the shipment and warehouse Web services has an impact on the KPI value, as expected.

Table 2 shows more detailed results of the experimental results. We have experimented with two algorithms, J48 and ADTree, and generated trees for different numbers of process instances (100, 400, 1000). The results show that the ADTree algorithm produces bigger trees than J48 (third column: number of leaves and nodes) for the same number of instances. However, it also reaches a higher precision (last column: correctly classified instances). The experiments further show that the trees are getting bigger 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

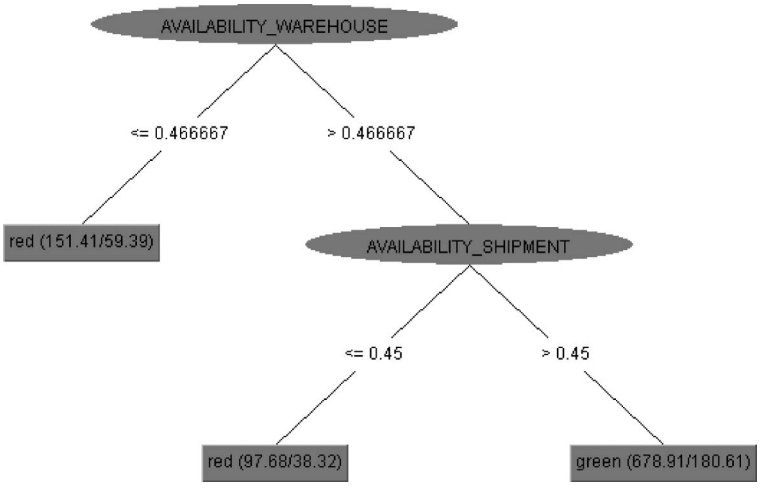

Figure 5. Generated tree for dependency of KPI on availability of IT infrastructure.

Table 2. Experimental results.

Instances	Algorithm	Leaves/nodes	Displayed metrics	Correctly classified (%)
100	J48	4/7	4	95.0
100	ADTree	11/16	4	98.0
400	J48	6/11	4	97.8
400	ADTree	17/26	5	99.0
1000	J48	11/18	6	98.8
1000	J48 -R	6/11	4	97.9
1000	J48 -U	13/22	9	99.2
1000	ADTree	19/28	6	99.4

bigger, factors are shown in the tree which have only marginal influence and thus make the tree less readable; the column 'Displayed Metrics' shows how many distinct metrics are displayed in the tree. To improve the readability, i.e. make the tree more compact, we have experimented with several parameters available for the algorithms. In general, 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 to reduce the size of the tree was 'reduced error pruning' (J48 -R) (Witten and Frank 2005). Another option, in the case of too many undesirable (marginal) metrics, is to simply remove those metrics from the analysed metric set and repeat the analysis. Finally, both algorithms show very similar results concerning the displayed influential metrics. Typically there is only one or at most two (marginal) metrics which differ. Both algorithms have displayed two of the three expected metrics (always preferring *order in stock* over *product type* as explained above). Finally, concerning the analysis duration, in our setting on a standard laptop computer a decision tree generation based on 1000 instances takes about 30.

Overall, these experiments show that the generated trees contain the expected influential metrics in a satisfactory manner and produce suitable results 'out of the box'. Therefore, we argue that the approach is suitable for non-IT personnel, even though this claim has yet to be verified through real-life evaluation. Concerning the influential factors displayed in the tree, we have identified two problems: (1) the user should be aware that the tree might hide some influential factors if there are 'multi-level' dependencies between metrics (*product type* influences *order in stock* which again influences *order fulfillment time*). In that case, further analysis (drill-down) of lower-level metrics may help to find further influential factors, (2) when the tree gets bigger it contains often some 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 marginal metrics from the analysed metric set and repeat the analysis. Both techniques often lead to more satisfactory results.

7. 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 languages and mechanisms. Baresi and Guinea (2005) deal with monitoring of WS-BPEL processes focusing on functional runtime validation. The goal is to detect partner services which deliver unexpected results concerning functional expectations. Barbon *et al.* (2006) 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 also supports collecting statistical and timing information, i.e. process performance metrics. Both approaches have in common that they concentrate on (functional) monitoring of WS-BPEL processes and use proprietary languages for expressing complex monitored properties. In contrast to those approaches, we focus on non-functional aspects and support both monitoring of process performance metrics and QoS metrics, and their correlation using a general-purpose CEP language. IBM's BAM approach described by Wahli *et al.* (2007) is similar to ours and it also supports monitoring based on arbitrary events in addition to WS-BPEL events and also evaluation of process metrics, however it does not support integration of QoS data and dependency analysis.

When it comes to the analysis aspect, most closely related to our work is iBOM, a platform for business operation management developed by HP (Castellanos *et al.* 2005). It 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 approach which explains dependencies among different impact factors is described by Bodensta *et al.* (2008). This approach focuses on dependencies of SLAs of the overall composition on SLAs of used base services, and analyses reasons for SLA violations. In contrast to our approach, the dependency relations and the impacts factors are identified at design time, and then later compared with monitoring results during runtime. The approach supports only analysis of response time and cost metrics. In our approach, we construct the dependency model based on monitoring results using data mining and support arbitrary metrics, in particular data-based metrics.

Another popular approach is process mining which includes several techniques that operate on event logs provided by information systems and perform different kinds of analysis on them (van der Aalst *et al.* 2004). Process discovery techniques aim at deriving a process model out of event logs when there is no explicit process model a priori (van der Aalst *et al.* 2004). In that context, process mining deals also with discovering performance bottlenecks by analysing idle times vs. working times and flow times on the discovered process model. That kind of analysis yields information which in our approach is obtained by monitoring metrics as specified in the metrics model. Besides just obtaining basic process metrics, another approach in process mining is process completion time prediction. Dongen *et al.* (2008) present an approach to cycle time prediction of process instances based on regression techniques. The approach takes into account the partial trace of the process instance until the prediction takes place and the history logs of past process instances. Our approach can also be extended towards prediction as we have shown by Leitner *et al.* (2009). Thereby, the prediction is performed during process execution time at previuosly specified check points in the BPEL process model using neural networks.

8. Conclusions

In this article we have presented an integrated monitoring and analysis framework for KPIs of SOA-based business processes. Our monitoring approach supports

monitoring of business process performance in terms of both process performance metrics and QoS metrics. We have shown how WS-BPEL resource events and QoS events are modeled and how they can be correlated and aggregated using CEP technology. We have also explained how metric definitions can be generated semi-automatically. The analysis part of the framework is based on decision trees and enables analysing the main factors that influence the business process and make it violate its performance targets. 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 analysis based on different metric sets can influence the analysis result.

Our future work includes extending the framework towards making use of 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 manually, e.g. by exchanging service bindings. We currently think about a more automated 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 article.

Acknowledgements

The authors thank Ivona Brandic for discussions and input on an earlier version of this article. 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).

Notes

1. http://www.supply-chain.org/
2. http://esper.codehaus.org
3. http://activemq.apache.org/
4. http://esper.codehaus.org/
5. http://www.mysql.com/
6. http://www.cs.waikato.ac.nz/ml/weka/
7. http://cxf.apache.org/

References

Barbon, F., *et al*., 2006. Run-time monitoring of instances and classes of web service compositions. *In*: *Proceedings of the IEEE international conference on web services (ICWS'06)*. Washington, DC: IEEE Computer Society, 63–71.

Baresi, L. and Guinea, S., 2005. Towards dynamic monitoring of WS-BPEL processes. *In*: *Proceedings of the 3rd international conference of service-oriented computing (ICSOC'05)*. Berlin/Heidelberg: Springer, 269–282.

Bodenstaff, L., *et al*., 2008. Monitoring dependencies for SLAs: the MoDe4SLA approach. *In*: *Proceedings of the 2008 IEEE international conference on services computing (SCC '08)*. Washington, DC: IEEE Computer Society, 21–29.

Castellanos, M., *et al*., 2005. iBOM: a platform for intelligent business operation management. *In*: *Proceedings of the 21st international conference on data engineering (ICDE'05)*. Washington, DC: IEEE Computer Society, 1084–1095.

Dongen, B.F., Crooy, R.A., and Aalst, W.M., 2008. Cycle time prediction: when will this case finally be finished? *In*: *Proceedings of the OTM 2008 confederated international conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008*, Monterrey, Mexico. Berlin, Heidelberg-Springer-Verlag, 319–336.

Freund, Y. and Mason, L., 1999. The alternating decision tree learning algorithm. *In*: *Proceedings of the 16th international conference on machine learning (ICML '99)*. San Francisco, CA: Morgan Kaufmann Publishers Inc, 124–133.

Karastoyanova, D., *et al*., 2006. *BPEL event model*. Technical Report 2006/10, University of Stuttgart, Germany.

Leitner, P., *et al*., 2009. Runtime prediction of service level agreement violations for composite services. *In*: *Proceedings of the 3rd workshop on non-functional properties and SLA management in service-oriented computing (NFPSLAM-SOC'09)*. Berlin/Heidelberg: Springer-Verlag.

Luckham, D., 2002. *The power of events: an introduction to complex event processing in distributed enterprise systems*. Boston, MA: Addison-Wesley Longman Publishing Co.

OASIS, 2007. Web services business process execution language version 2.0 – OASIS standard [online]. Available from: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel [Accessed 1 March 2010].

OMG, 2009. Business process modeling notation, specification, v.2.0-beta1 [online] Available from: http://www.omg.org/cgi-bin/doc?dtc/09-08-14 [Accessed 1 March 2010].

Papazoglou, M.P., *et al*., 2007. Service-oriented computing: state of the art and research challenges. *Computer*, 40 (11), 38–45.

Quinlan, J.R., 1993. *C4.5: programs for machine learning*. San Francisco, CA: Morgan-Kaufmann.

Rosenberg, F., Platzer, C., and Dustdar, S., 2006. Bootstrapping performance and dependability attributes of web services. *In*: *Proceedings of the IEEE international conference on web services (ICWS '06)*. Washington, DC: IEEE Computer Society, 205–212.

van der Aalst, W., Weijters, T., and Maruster, L., 2004. Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16 (9), 1128–1142.

Wahli, U., *et al*., 2007. *Business process management: modeling through monitoring using WebSphere V6.0.2 products*. Riverton, NJ: IBM, International Technical Support Organization.

Weske, M., 2007. *Business process management: concepts, languages, architectures*. Secaucus, NJ: Springer-Verlag Inc.

Wetzstein, B., Strauch, S., and Leymann, F., 2009. Measuring performance metrics of WS-BPEL service compositions. *In*: *Proceedings of the 5th international conference on networking and services (ICNS 2009)*, 20–25 April, Valencia, Spain, Washington. DC: IEEE Computer Society, 49–56.

Witten, I.H. and Frank, E., 2005. *Data mining: practical machine learning tools and techniques*. 2nd ed. San Francisco, CA: Morgan Kaufmann.