Measuring Performance Metrics of WS-BPEL Service Compositions

Branimir Wetzstein, Steve Strauch, Frank Leymann

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{wetzstein, strauch, leymann}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{Wetzstein2009,
  author    = {Branimir Wetzstein and Steve Strauch and Frank Leymann},
  title     = {Measuring Performance Metrics of WS-BPEL Service Compositions},
  booktitle = {Proceedings of the 5th International Conference on Networking and
               Services (ICNS 2009)},
  year      = {2009},
  pages     = {49--56},
  doi       = {10.1109/ICNS.2009.80},
  publisher = {IEEE Computer Society}
}
```

Measuring Performance Metrics of WS-BPEL Service Compositions

Branimir Wetzstein, Steve Strauch, Frank Leymann
Institute of Architecture of Application Systems
University of Stuttgart
Universitaetsstr. 38, 70569 Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

Abstract

In this paper we present an approach to the development of monitoring solutions for processes implemented as WS-BPEL service compositions. The approach allows modeling of process performance metrics in a platform-independent manner and then generating an event-based monitor model for a specific WS-BPEL process engine. We create a metamodel which enables modeling of different types of process performance metrics. In particular, our approach supports modeling of metrics related to correlated processes. In the deployment phase, we generate a monitor model based on a proprietary event metamodel of a process engine. In addition, we determine which events are needed for the calculation of PPMs, and generate corresponding deployment information for the process engine.

1. Introduction

Business Process Management encompasses methods, techniques, and tools that allow organizing, executing, and measuring the processes of an organization [12]. Typically, the business process lifecycle begins with the business analyst analyzing business processes in the company and creating process models using a process modeling tool. When the process is to be automated, it is translated by IT engineers to an executable process model, which is run on a process engine. When BPM is layered over a Service Oriented Architecture (SOA), services are used for implementing activities of business processes. In the context of SOA, business processes are modeled and executed using the (WS-) BPEL language [8], which is a workflow language for orchestration of services. The BPEL process engine executes the BPEL process model by delegating the process tasks to Web services.

For controlling the performance of business processes, business activity monitoring (BAM) technology enables continuous, near real-time monitoring of processes based on process performance metrics (PPMs). Business people define PPMs based on business goals. These PPMs are then translated to monitor models by IT engineers. At process execution time, the process engine publishes events as the process is executed. A business process monitoring tool subscribes to these events, calculates the PPM values and displays them in dashboards to business users. In case of severe deviations from planned values, alerts are raised and notifications are sent to responsible people.

When it comes to developing monitoring solutions for business processes, there exist several challenges that we want to address in this paper: (i) While there is a standard language for specifying and executing processes in the context of Web services, namely BPEL, there is no corresponding approach on how to model performance metrics for business processes. Typically, monitor models, which serve as the input to the BAM tool and describe how PPMs are calculated, are created by developers based on the events the process engine emits for a business process, i.e. on a lower abstraction level. (ii) As event metamodels of process engines are proprietary, metric definitions are tightly coupled to a specific process engine reflecting their specific event metamodel.

In this paper we address these challenges by introducing a top-down development approach for monitoring of PPMs for WS-BPEL based business processes, as shown in Figure 1. For the definition of PPMs, the user specifies PPM attributes such as name, type, data unit, analysis period, target value, deviations and alerts, and defines how PPMs are calculated using a domain specific language. He therefore uses predefined functions such as "Duration", "State", and "ProcessVariableValue" (for accessing business objects), which can be combined using arithmetic, logical and relational operators. These functions use process probes as parameters, which are pointers to process elements (typically activities and variables) specifiying at which state (e.g., "started", "completed", "modified") the information is needed. Therefore, an engine-independent event metamodel for BPEL processes is used. The created PPM model is thus inde-

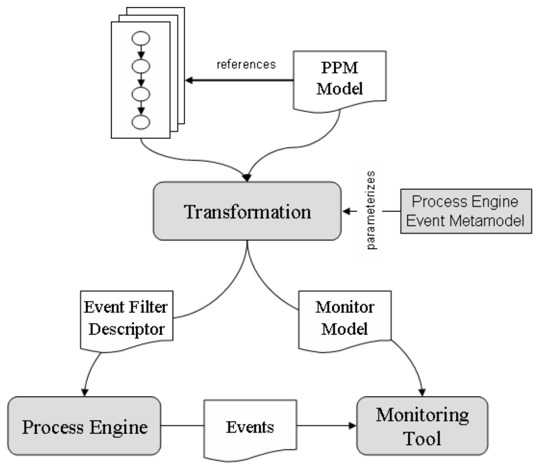

Figure 1. Overview of the Approach

pendent of a certain BPEL engine implementation. In the next step, based on the event metamodel of the target BPEL engine(s), we generate a monitor model and an event filter. The event filter is part of deployment information of the BPEL engine specifying which events it needs to publish. The monitor model specifies how to calculate the PPMs based on the runtime events. At deployment time, the monitor model is deployed to the monitoring tool. The process models are deployed to one or several process engines. In addition, the monitoring tool subscribes to events needed by the process engines. At process runtime, the process engine executes the process and publishes events as specified in the deployment descriptor to a publish/subscribe infrastructure. The business process monitor receives these events, computes the corresponding PPM values, and displays them in dashboards.

The contributions of the paper are as follows. We classify PPMs and in particular identify the need for PPMs which are measured across correlated processes. We create a metamodel which enables modeling of PPMs independently of a specific event metamodel supported by a process engine. We then show how this PPM model can be transformed into a monitoring model based on a specific event metamodel and how metrics are calculated at process execution time. The modeling approach is independent of process engine implementation and thus flexible in that definition of PPMs is separated from its implementation specifics.

The rest of the paper is organized as follows. In Section 2 we present a case study which we use to present examples for our concepts and which we have implemented to evaluate our approach. Section 3 describes how process performance metrics are modeled. Section 4 shows then how PPM models are transformed to event-based monitor models. In Section 5 we present related work, and finally, in Section 6 we outline our future work.

2. Case Study

For evaluating our approach we have developed a purchase order (PO) scenario in the B2B area. The use case consists of a customer, a reseller, and a shipping service. The reseller offers certain products to its customers. It holds a certain part of the products in stock and orders missing products if necessary; this ordering part is however left out for space reasons. In the implemented scenario, the customer sends a PO request with details about the required products and needed amounts to the reseller. The latter checks whether all products are available in stock in the warehouse. If some products are not in stock and cannot be ordered from suppliers, the reseller notifies the customer. As long as customer and manufacturer do not agree on the order, this process is repeated. When a mutual agreement has been achieved, the reseller sends the purchase order to the warehouse for shipment. The warehouse packages the products and hands them over to the shipment service. Finally, the reseller sends a purchase order response back to the customer. The shipment service delivers the products to the customer and confirms the shipment. The interactions of the participants in our scenario are illustrated in the BPMN diagram shown in Figure 2.

The scenario has been implemented as follows: the PO process and the warehouse process are implemented as two different BPEL process models. The shipment service is implemented in Java and exposed as a WSDL Web service. The PO BPEL process and the warehouse BPEL process have been deployed on an Apache ODE BPEL engine.

For evaluating the performance of the processes defined in our scenario, we have chosen PPMs from the SCOR model [3]. The SCOR model specifies reference processes and PPMs in the supply-chain domain. We define the following PPMs from the point of view of the reseller: (i) Average duration of the PO process (before products are handed over to the shipping service); (ii) Percentage of POs which were delivered ”in time”; (iii) Percentage of purchase orders which were delivered ”in full” as requested by customer. More details on these PPMs, such as target values and more detailed definitions, will be given in Section 3.3 when explaining how they are modeled.

3. Modeling Process Performance Metrics

In this paper we concentrate solely on metrics which evaluate the efficiency and effectiveness of business processes and which can directly be derived and computed based on the runtime data of service compositions; we call this type of metrics process performance metrics (PPMs). We do not consider information from external data sources such as operational databases (e.g., CRM database) or services not implemented as service compositions. Further-

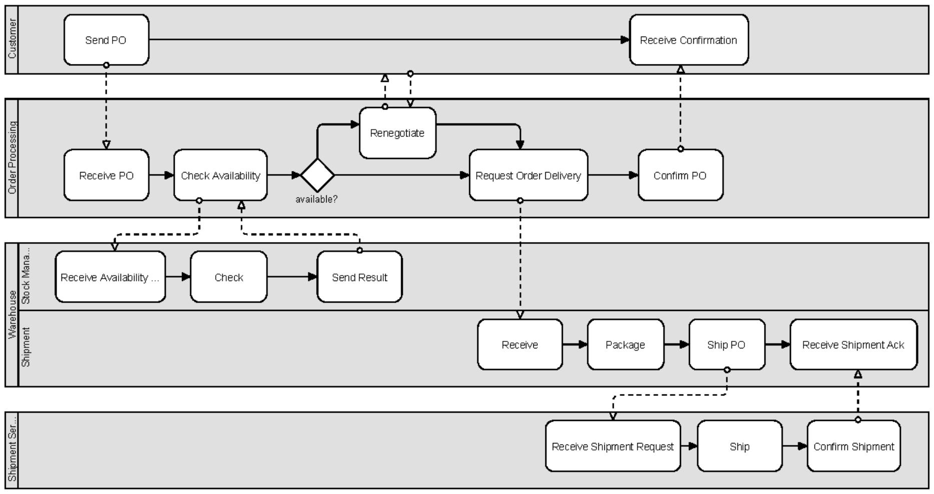

Figure 2. Case Study: Purchase Order Processing Scenario

more, we do not consider IT-level metrics such as the availability of the process engine, which of course also have an impact on business process performance. Adding support for external data sources, and IT level metrics is part of our future work.

In order to come up with a metamodel for modeling PPMs, we have analyzed the process metric catalogue of SCOR Supply Chain Metrics [3] and related literature [10]. A PPM can be seen as a metric, which is evaluated in a certain analysis period and has a target value which should be reached or preserved within the analysis period indicating the achievement of predefined business goals [10]. Typically, in addition to a target value, allowed ranges of values are defined. In case of deviations, notifications can be sent. An example for a PPM is ”percentage of purchase orders that were processed in time in the next 3 months should be greater than 90”, the analysis period being 3 months and the target value 90. The calculation formula of PPMs can be specified based on following information: duration of the process and its activities, cost of the process and its activities, business objects and their states, and execution flow of process activities. In addition, it is important to be able to specify PPMs across several business process models.

PPM computation is often based on business objects which are dealt with in the process to be measured. Such business objects (a.k.a. business items, business documents or business data) are for example ”purchase order”, or ”shipment” in our case study. Typically, PPMs contain a condition on the state of these business objects or query their attributes (e.g., ”purchase order processed in time and in full”). Some metrics are based on more than one business process, e.g. the PPM ”average duration for processing a PO” needs data both from the purchase order process and the warehouse process.

The PPM metamodel (not shown as UML class diagram for space reasons) defines a PPM as consisting of the following attributes: name, description, data type, unit, target value, analysis period, deviation value range and corresponding alerts, and finally calculation formula. The calculation formula is the ”heart” of the PPM definition, as it defines how a PPM is measured based on other PPMs. For specifying the calculation formula we have defined a domain specific language. When specifying the calculation of PPMs, one can distinguish between instance metrics and aggregate metrics. Instance metrics are based on one single process instance, while aggregate metrics aggregate values of instance metrics for several process instances by using aggregation functions.

3.1 Instance Metrics

For specification of simple instance metrics we have defined several functions as shown in Table 1. Thereby, one function defines one simple instance metric. These functions enable retrieving basic metrics on process instances. The parameters of these functions (Process, Activity, Variable) are ”process probes”, i.e. pointers to corresponding artifacts of concrete process models (Section 3.1.2).

The function ”duration” refers to the whole process instance, a process fragment, or a single activity, depending on which parameters are used. The function ”count” retrieves the number of executions of an activity in a loop. The function ”state” returns a set of states containing all states the specified process or activity have been in during process instance execution. This function is needed to be able to test e.g. whether a certain activity has been executed, or whether a process instance terminated. The function ”time” retrieves the timestamp when an activity or

Category	Function
Duration	duration(Process)
	duration(Activity1, Activity2)
	duration(Activity)
Count	count(Activity)
State	state(Process)
	state(Activity)
Time	time(Process)
	time(Activity)
Process Data	processVariableValue(Activity,Variable)

Table 1. Instance Metric Functions

process reached a certain state. Finally, ”processVariable-Value” enables retrieving of data values of business objects at a certain activity. Note that the function ”duration” is a convenience function, which could also be specified using operations over ”time” functions. For calculating cost of processes and its activities, one could introduce a ”process cost” function, similarly to the ”duration” function. That function would, however, rely on the definition of a cost model, which is out of scope of this paper.

Predefined functions enable the specification of simple instance metrics. Composed instance metrics can be used to combine several (simple or composed) instance metrics into a new metric using arithmetic operators (e.g., duration(A, B) + duration(C,D)) or specifying a logical condition using logical and relational operators (e.g., processVariableValue(”Receive PO”, ”PurchaseOrder.totalCost”) $>$ 1000 AND state(”Renegotiate”) = ”started”).

3.1.1 Cross-Process Metrics

A special type of instance metric is the cross-process instance metric. It supports the definition of metrics which are based on two process instances from two different process models. For example, for measuring the ”average duration of the PO process” in the case study, one needs to measure the duration between two activities from two different process models (Section 3.3). Thereby, the problem arises that one has to correlate two process instances which belong together (i.e., to the same purchase order). To be able to correlate two instances of these processes, one has to additionally specify a correlation token, in this case ”purchase order ID”, when specifying correlated instance metrics. The correlation token is based on one or several attributes of process data variables, and has to be specified by the user by pointing to the corresponding variable attribute(s) in both process models. It should be noted that the two process instances do not have to be part of process models which ”communicate” with each other as in the example above. E.g., a PPM which measures how often ”orders have been returned after they have not been delivered in time” involves the returns process and the PO process which are not communicating with each other in terms of WS calls.

3.1.2 Specifying Process Probes

Simple Instance Metrics are based on functions which contain ”Process”, ”Activity”, and ”Variable” as parameters. These parameters are pointers to corresponding artifacts of concrete process models. They serve as process probes. When attaching the parameters to the corresponding process model elements, in addition to specifying the identifier of the corresponding element, the user is able to specify the state of the process or activity the parameter relates to. In addition, in case of cross-process metrics, correlation tokens have to be attached to process variables, as explained in the previous section.

When defining functions, we assume certain default values concerning process and activity states (e.g., in case of duration(A, B), we assume ”A started” and ”B completed”). However the user has the possibility to override these default settings by defining other process or activity states. The process state metamodel and the activity state metamodel are strongly related to the event metamodel of a process engine, which publishes events when state changes occur. In our approach, we use the engine-independent BPEL state metamodel as proposed in [5]. For most metrics the states ”started” and ”completed” suffice.

As functions have ”Process”, ”Activity” and ”Variable” as parameters, they have to reference corresponding process models and activities. Since BPEL does not prescribe unique identifiers for processes, activities as well as variables, first, unique identifiers have to be inserted either manually by the user (or automatically if supported by the BPEL editor) in the BPEL file. This can be done by setting an attribute ”id” in the corresponding XML elements in the BPEL file. Then, we can reference BPEL processes, BPEL activities and BPEL variables out of the PPM model by using the XML Linking Language (XLink) and the XPointer xpointer() Scheme. Listing 1 shows how the attachment mechanism is used.

3.2 Aggregate Metrics

Aggregate metrics aggregate values of instance metrics using aggregation functions ”sum”, ”max”, ”min”, ”avg”, and ”qty”, as shown in Table 2. In addition, to the instance metric which is aggregated, aggregate metrics can get a logical condition as a second parameter, defined by a composed instance metric. In that case, only those instances are aggregated which fulfill the logical condition.

In difference to the other aggregation functions, the aggregate metric ”quantity” has only one parameter, namely

Category	Function
Summation	sum(InstanceMetric)
	sum(InstanceMetric, ComposedInstanceMetric)
Average	avg(InstanceMetric)
	avg(InstanceMetric, ComposedInstanceMetric)
Maximum	max(InstanceMetric)
	max(InstanceMetric, ComposedInstanceMetric)
Minimum	min(InstanceMetric)
	min(InstanceMetric, ComposedInstanceMetric)
Quantity	qty(ComposedInstanceMetric)

Table 2. Aggregation Functions

a composed instance metric specifying a logical condition. It determines the number of instances fulfilling this logical condition (e.g., qty(”purchase order has been processed in time”). Aggregate metrics can be composed using arithmetic operators.

3.3 DSL for PPM Specification

For specifying the calculation of PPMs we have defined a domain specific language (DSL) based on the functions specified in Tables 1 and 2 which can be combined using arithmetic, relational, and logical operators. We have implemented an Eclipse plugin for a BPEL 2.0 editor, which enables modeling of PPMs. It produces a PPM model serialized in XML. The XML model references BPEL process models.

Table 3 shows the use of the DSL for specifying the calculation formula of PPMs of the case study. The first metric is a simple aggregate metric which aggregates a simple cross-process instance metric based on the function ”duration(Activity1, Activity2)”. The correlation token is specified during attachment (see Listing 1). The second metric is a composed aggregate metric which calculates the percentage value based on two simple aggregate metrics. Both aggregate metrics count (qty) the number of process instances which fulfill the corresponding logical expressions (composed instance metrics). For testing whether the ordered products were delivered in time, we assume that the ”Purchase Order” business object contains the attribute ”deliveryDeadline”. In the logical condition we then compare the date in the attribute ”deliveryDeadline” with the value of the ”receiptDate” attribute of the ”Shipment” business object. Note that this composed instance metric is a cross-process metric, as the two activities ”Receive Shipment Ack” and ”Request Order Delivery” are part of different process models. Finally, with the second aggregate metric we count the process instances which have completed. The third PPM is specified in a similar way as the second one. A purchase order has not been processed ”in full” if renegotiating is needed. That is the case if the activity ”Renegotiate” has been started (state(”Renegotiate”) = ”started”).

The language presented in Table 3 enables a top-down specification of PPMs. Thereby, only the calculation of the ”root” PPM is specified, as shown also in Listing 1. The lower-level metrics which are composed or aggregated for the calculation of the PPM are automatically detected when parsing the calculation expression. This detection is needed later for monitoring, as the PPM is then calculated bottom-up, starting with instance metrics which are calculated based on runtime events.

```
<ppmModel>
  <ppm name="Average Duration of PO Process until
       Shipment">
    <unit>hours</unit><targetValue>8</targetValue>
    <calculation>
      <avg><duration><fromActivity name="<Receive PO>"><
           toActivity name="<Ship PO>"></duration></avg>
    </calculation>
    <attachments>
      <activityAttachment parameterName="Receive PO"
           xlink:type="simple" xlink:href="POProcess.
           bpel#xpointer(//*[@name=\&quot;ReceiveOrder\&
           quot;])" status="started"/>
      <activityAttachment parameterName="Ship PO"
           xlink:type="simple" xlink:href="
           WarehouseProcess.bpel#xpointer(//*[@name=\&
           quot;ShipOrder\&quot;])" status="completed"/>
    </attachments>
    <correlation>
      <correlationKey>
        <source>
          <variableAttachment xlink:type="simple"
               xlink:href="POProcess.bpel#xpointer(//*[
               @name=\&quot;PurchaseOrder.Id\&quot;])"/>
          <activityAttachment xlink:type="simple"
               xlink:href="POProcess.bpel#xpointer(//*[
               @id=\&quot;ReceiveOrder\&quot;])" status=
               "completed"/>
        </source>
        <target>
          <variableAttachment xlink:type="simple"
               xlink:href="WarehouseProcess.bpel#
               xpointer(//*[@name=\&quot;Shipment.poId\&
               quot;])"/>
          <activityAttachment xlink:type="simple"
               xlink:href="WarehouseProcess.bpel#
               xpointer(//*[@id=\&quot;ShipOrder\&quot
               ;])" status="completed"/>
        </target>
      <correlationKey>
    </correlation>
  </ppm>
</ppmModel>
```

Listing 1. PPM Definition with Attachment and Correlation

4 Monitoring Process Performance Metrics

The result of PPM modeling consists of PPM definitions which are based on one or more business process models.

Description	Calculation Expression
Average Duration of PO Process	avg(duration(”Receive PO”, ”Ship PO”))
Percentage of POs delivered ”in time”	qty(processVariableValue(”Receive Shipment Ack”, Shipment.receiptDate) $<$ processVariableValue(”Request Order Delivery”, PurchaseOrder.deliveryDeadline)) $\div$ qty(state(”Purchase Order Process”) = ”completed”) $\times$ 100 %
Percentage of POs delivered ”in full”	100% $-$ (qty(state(”Renegotiate”) = ”started”) $\div$ qty(state(”Purchase Order Process”) = ”completed” AND state(”Warehouse Process”) = ”completed”)) $\times$ 100 %)

Table 3. Calculation Specifications for PPMs from the Case Study

In order to enable monitoring of PPMs at process execution time, two actions have to be performed: (i) information has to be generated for the process engine about which events need to be published for measurement of PPMs; (ii) the PPM model has to be transformed to a monitor model which is deployed to the monitoring tool.

Many process engines support publishing of events during process execution. Events thereby denote a state change of a process entity as the process instance is executed, e.g., completion of an activity or modification of a variable value. Although there is a standard specification for the workflow audit trail in the reference model of the Workflow Management Coalition [13], it is not supported by most process engines. Thus, in practice, process engines differ in event metamodels they implement. Besides different event metamodels, there are also differences between engines in terms of how to configure which events to publish for a particular process. This kind of configuration is typically done in the deployment descriptor of the process model to be deployed.

The granularity of configuration differs again between engines. For example, Apache ODE [4] supports filtering of events only on scope level, while IBM Process Server supports specifying for each BPEL element which types of events to publish and to some extent also which content the events should contain [11]. The possibility of configuration which events should be published is important, because simply logging all possible events, i.e. every state change of each process, activity, and variable, would unnecessarily degrade the performance of the process execution infrastructure [6]. In particular for the evaluation of process performance metrics only a small subset of all possible events is needed.

4.1 Determining Needed Events

As shown in Figure 1, the transformation takes the PPM model and the event metamodel of the process engine as input and then generates the monitor model and event filtering information for the process engine. The main task here is to map the instance metrics in the PPM model to events which are needed for their calculation. Other types of metrics are not relevant, because they are not calculated based on events but based on other metrics.

Function	Required Events
duration(Process)	ProcessInstanceStartedEvent, ProcessCompletionEvent
duration(Activity)	ActivityExecStartEvent, ActivityExecEndEvent
duration(Act1, Act2)	ActivityExecStartEvent, ActivityExecEndEvent
count(Activity)	ActivityExecEndEvent
time(Process)	ProcessCompletionEvent
time(Activity)	ActivityExecEndEvent
state(Process)	ProcessInstanceStateChangeEvent
state(Activity)	Activity*Event (for all states of the Activity)
processVariableValue (Act,Var)	ActivityExecEndEvent, VariableModificationEvent

Table 4. Mapping Instance Metrics to Events

When determining events for a calculation of a PPM, three aspects have to be addressed: (i) All simple instance metrics (based on functions shown in Table 1) which are used for the calculation of a PPM are calculated based on runtime events; each function with its attachments can be mapped to a set of events needed for the calculation of this function (Table 4 shows a mapping based on the event metamodel of the Apache Ode BPEL Engine [4]). (ii) For cross-process metrics in addition correlation tokens have to be mapped to additional events; (iii) For each instance metric additional termination events are needed (see below).

When it comes to configurability there are greater differences between process engines. We assume that we can configure for the process and for each activity of the process which events we are interested in. If a process engine does not support that level of configuration, then there will be more events published than needed for the evaluation of the PPMs. In that case, the business process monitor has to simply discard those events when they are received.

4.1.1 Mapping Instance Metric Functions

Table 4 shows how Instance Metric Functions are mapped to events for the Apache ODE BPEL Engine. Each event contains at least the following attributes: creation timestamp, process instance ID, process model ID, and activity ID (for activity related events).

The function ”duration” is mapped to two events, a start event and an end event and is calculated by subtracting the timestamps of these two events. Per default these two events denote the states ”started” and ”completed”, however, also other or additional states are considered if they are explicitly modeled on PPM level. The user could for example decide that not only ”completed” but also ”terminated” or ”faulted” processes should be considered. The function ”time” simply obtains the timestamp at which an event occurred. Thus, one just has to log the event for a certain process state or activity state. The function ”count” is counting how often a state, per default ”completed”, occurs for an activity. Thus it is mapped to the corresponding event of that state. The function ”state” retrieves the states a process or an activity has been in during process instance lifetime. This function is simply mapped to all events which correspond to the state model of the process or activity. The function ”processVariableValue” is used to access the state of process variables at a certain activity. The function thus has to be mapped to the corresponding event marking the state of the activity (”completed” per default). This event however does not suffice, because the event does not carry variable value data. There is an additional event which is published when the value of a variable changes. The function is thus mapped to both the activity state relevant event and the variable modification event. When evaluating this function one thus waits for the ”completed” event of the activity and then evaluates the last variable modification event received according to the timestamp. A more optimal solution would be to send the needed variable data with the activity event, however not every process engine supports that.

4.1.2 Mapping Correlation Tokens

A cross-process instance metric specifies one or more correlation keys which are mapped to variables in two or more process models. To be able to correlate process instances, one thus has to receive those data values from each process instance. Thus, correlation keys are mapped to variable modification events for variables which contain the correlation key in each process model.

4.1.3 Termination Events

In some cases, it can happen that a metric cannot be calculated because the corresponding activity is never reached during execution of a process instance (because of a fault or simply because another alternative branch has been taken). E.g. if ”Activity2” is never executed in a paticular process instance, ”duration (Activity1, Activity2)” cannot be calculated. In that case the monitoring tool should know when to abort the calculation of the metric, otherwise it would wait ”forever” for the needed event. In order to deal with these problems, we introduce the concept of termination events, which signal that the calculation of the instance metric can be finished (if possible) or aborted. For some instance metric types, regular events can act as termination events, e.g. ”Activity 2 completed” in the example above. For others, extra events are needed. In that case, in our implementation, we use all events signalling the end of a process instance as termination events. When such an event is received, the monitoring tool deals with all yet ”unfinished” instance metrics of that process instance. In case of cross-process metrics termination events of both process instances are needed. In some cases, one could use also timeout events or more fine granular termination events, these are however out of scope of this paper.

4.2 Monitoring

The monitor model specifies how events received by the monitoring tool are processed to calculate the PPMs. The PPM model already contains definitions of different types of PPMs. Each PPM can be seen as a tree. Leafs of this tree are the functions which define simple instance metrics. The monitor model extends this PPM tree by inserting events as leafs which are needed for the calculation of the function as described in last section. The parent nodes of the events are the corresponding simple instance metrics. The monitor model effectively combines the PPM model with the corresponding event model, as shown in Figure 3 for the PPM ”average process duration” from the case study (termination events are not displayed in the Figure). At process deployment time, the monitoring tool subscribes to all events specified in this event model. At process runtime, when the process engine publishes a corresponding event, that event is received by the monitoring tool, which evaluates the metrics of the PPM tree in a bottom up fashion.

5 Related Work

There are several approaches to monitoring of service compositions described in the literature and already implemented in products. IBM's approach integrates performance management tightly into the business process lifecy-

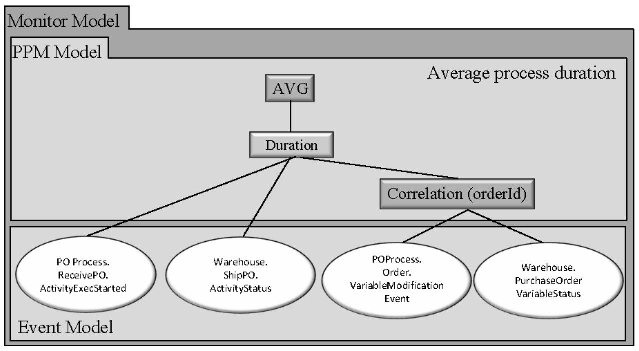
Monitor Model
PPM Model
Average process duration
AVG
Duration
Correlation (orderId)
PO Process. ReceivePO. ActivityExecStarted
Warehouse. ShipPO. ActivityStatus
POProcess. Order. VariableModification Event
Warehouse. PurchaseOrder VariableStatus
Event Model

Figure 3. Monitor Model

cle and supports it through its WebSphere family of products [11]. Thereby, PPMs are modeled based on events of the Process Server BPEL engine. In our approach, we model PPMs in a platform-independent manner and provide a DSL and support for cross-process metrics. [7] presents a model-driven approach to developing monitored business processes. The authors show how a monitoring model can be integrated with a BPMN process model, finally generating a BPEL process with special activities provide monitoring information by invoking operations on the monitoring tool. Our approach is different, in that we support a more elaborate metric metamodel and we rely on the event dispatching mechanism of the process engine providing information for monitoring. [9] also extends a BPEL process definition with auditing activities in order to publish state changes to the monitoring tool. The auditable BPEL process definition does not use proprietary event metamodels of process engines but remains compliant with the standard. The approach, however, does not deal with specification of process performance metrics. [2] deals with monitoring of 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. [1] describes a monitoring approach for BPEL processes which also distinguishes between instance and aggregate metrics. It, however, does not deal with modeling of PPMs in an engine-independent manner and does not support cross-process metrics.

6 Conclusions and Outlook

In this paper we have presented an approach to monitoring of PPMs of WS-BPEL processes. We have implemented the presented concepts as follows: modeling of PPMs is supported by an Eclipse plugin which is part of the Eclipse BPEL Designer. As output, the plugin creates a PPM model serialized in XML. The XML file is read by a transformation component implemented in Java which generates the monitor model supporting the Apache ODE BPEL engine. The monitor model is deployed on a monitoring tool which computes the PPMs at process runtime and displays the results in dashboard-like views. In our future work, we will extend our approach to support also human tasks, external data sources, cost models, and resource-level metrics.

Acknowledgment

The research leading to these results has received funding from the European Community's 7th Framework Programme under the Network of Excellence S-Cube Grant Agreement no. 215483 and the COMPAS Project Grant Agreement no. 215175.

References

[1] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-Time Monitoring of Instances and Classes of Web Service Compositions. In *ICWS*, pages 63–71, 2006.

[2] L. Baresi and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *ICSOC 2005, 3rd International Conference of Service-Oriented Computing*, pages 269–282. Springer, 2005.

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

[4] A. S. Foundation. Apache ODE User Guide. *W3C Working Draft*. http://ode.apache.org/user-guide.html.

[5] D. Karastoyanova, R. Khalaf, R. Schroth, M. Paluszek, and F. Leymann. BPEL Event Model. Technical Report 2006/10, University of Stuttgart, Germany, November 2006.

[6] F. Leymann and D. Roller. *Production Workflow – Concepts and Techniques*. Prentice Hall, 2000.

[7] C. Momm, R. Malec, and S. Abeck. Towards a Model-driven Development of Monitored Processes. *In Proceedings of 8. Internationale Tagung Wirtschaftsinformatik (WI2007), Karlsruhe, Germany*, February 2007.

[8] Organization for the Advancement of Structured Information Standards (OASIS). Web Services Business Process Execution Language Version 2.0. Comitee Specification, April 2007.

[9] H. Roth, J. Schiefer, and A. Schatten. Probing and Monitoring of WSBPEL Processes with Web Services. *In Proceedings of CEC, San Francisco*, 2006.

[10] H. Schmelzer and W. Sesselmann. *Geschäftsprozessmanagement in der Praxis*. Hanser Verlag München, 2006.

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

[13] WfMC. Audit Data Specification. *W3C Working Draft*, 1998.