Towards a Model-driven Development of Monitored Processes

Christof Momm, Robert Malec, Sebastian Abeck

Cooperation & Management
Institut für Telematik

Universität Karlsruhe (TH)
76128 Karlsruhe
{momm,malec,abeck}@cm-tm.uka.de

Abstract

An integrated management of business processes demands a strictly process-oriented development of the supporting IT. Process-orientation is especially promoted by Service-Oriented Architectures (SOA), where loosely coupled business services are being composed to executable processes. In this paper we present a model-driven methodology for a top-down development of a process-oriented IT support based on a SOA. In contrary to existing approaches we also include the monitoring required for business process controlling and introduce meta-models for the specification of process performance indicators in conjunction with the necessary monitoring. Furthermore, we show how these models are transformed to executable process definitions extended by the required monitoring activities.

1 Introduction

Today, companies demand IT support that is strongly aligned with their business processes, which in turn are compliant with their (strategic) business goals. For achieving this, goal-driven approaches to an integrated Business Process Management (BPM) have been proposed [AaHW03; MuRo04]. For controlling goal achievement in business processes, solutions are required that allow a continuous, “real-time” monitoring of the performance within the IT support based on quantitative process performance indicators (PPI). This aspect is also referred to as “Business Activity Monitoring” (BAM) in relevant literature. The discussed BAM architectures [JeSC03; McSc04] have in common that they abstract from the IT systems implementing the business processes by introducing management relevant business events. The PPIs are defined and evaluated on the basis of these business events, which are generated by event adapters triggered by an instrumentation of the underlying IT systems. Because the business events and their associated PPIs as well as the existing IT support are extremely company-specific, the implementation of the required instrumentation is itself very time consuming and is elongated if facing an extremely heterogeneous IT support.

The heterogeneity of the IT support can be significantly reduced by establishing a company-wide Service-Oriented Architecture (SOA) in which the business processes are consequently realized through orchestrations on the business process layer of a SOA [Le03; LeRS02]. This is accomplished by either using the Business Process Execution Language (BPEL) [ACDG03] and web services or other SOA platforms, for example CORBA and an appropriate workflow engine. Unfortunately, a uniform methodology for realizing the required monitoring – including the instrumentation - does not as yet exist. It is still very specific to the employed SOA platform. Accordingly, a solution for specifying the monitoring in a platform-independent way and a clear methodology for breaking down PPIs into appropriate measuring points within the orchestrations or queries on logging data is necessary [HaRa01].

Taking these drawbacks into account, in this paper we propose a top-down approach for developing a uniform IT support based on a SOA in conjunction with the monitoring aspects required for processing the PPIs. To enable the support of different SOA platforms as well as an automated generation of the required instrumentation and monitoring infrastructure, we decided to build the approach on the principles of the Model Driven Architecture (MDA) proposed by the OMG [MiMu01]. The approach is demonstrated using a concrete business process taken from the field of higher education. So far, the target platform is limited to most common SOA platform based on BPEL and web services.

2 Related Work

For developing a service-oriented IT support tightly aligned with the underlying business processes, various model-driven approaches have been proposed [BaMR04; KHSW05]. Thereby, the business processes are specified by means of computation-independent business process models (CIM), for instance based on Petri nets, Event-driven Process Chains (EPCs) or the

Business Process Modeling Notation (BPMN). These process models are systematically refined and transformed into platform-independent models (PIM) of executable business processes (i.e. orchestrations). Finally, these PIMs are transformed to platform-specific models (PSM), in particular to executable process definitions which are mainly based on BPEL. So far, the presented approaches deliver solutions to the development of the functional aspects but do not consider monitoring and control (i.e. management) aspects.

For component-based software development this aspect has already been addressed. [PAVB04] present an approach which integrates Quality of Service (QoS) aspects into a model-driven development process of component-based applications and allows for an automated generation of the required monitoring infrastructure and component instrumentation. As the service-orientation leads to a significant reduction of complexness, some essential adaptations are necessary to seamlessly integrate monitoring aspects into a model-driven SOA development process.

To enable BAM on the orchestrations, the generic solution proposed by [JeSC03] can be employed. As already pointed out, in this case the instrumentation of the involved IT systems would be very time consuming as it has to be accomplished manually for each orchestration and execution engine. Furthermore, the presented implementation is based on proprietary technologies, which particularly complicates the monitoring of cross-enterprise business processes. Taking especially this shortcoming into account [McSc04] propose a framework for analyzing and measuring business performance on the basis of web services. In doing so, the whole BAM system is encapsulated in a Solution Manager Service providing a standardized interface for the instrumented IT systems. The information required for evaluating the PPIs is transferred to the Solution Manager Service by extending the BPEL process definition by management calls. [Mc03] amplifies this BPEL instrumentation, but does not address the questions how to systematically develop such instrumented orchestration in a top-down fashion and how to automate the generation of these artifacts.

3 Approach to a Model-Driven Orchestration Development

Within this section we introduce our general idea for a model-driven development of monitored orchestrations. Figure 1 provides an overview of our approach.

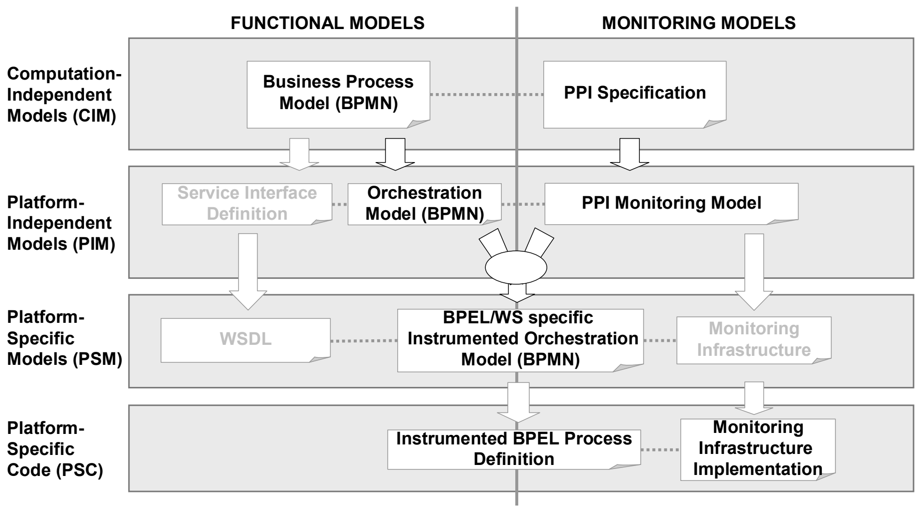

FUNCTIONAL MODELS
MONITORING MODELS
Computation-Independent Models (CIM)
Platform-Independent Models (PIM)
Platform-Specific Models (PSM)
Platform-Specific Code (PSC)
Business Process Model (BPMN)
PPI Specification
Service Interface Definition
Orchestration Model (BPMN)
PPI Monitoring Model
WSDL
BPEL/WS specific Instrumented Orchestration Model (BPMN)
Monitoring Infrastructure
Instrumented BPEL Process Definition
Monitoring Infrastructure Implementation

Figure 1: Model-Driven Orchestration Design

According to this we distinguish between functional and monitoring models on three different layers of abstraction (CIM, PIM and PSM). The target platform [MiMu01] for the functional parts is a specific SOA platform. These specifics are abstracted by the PIM. The same holds for the monitoring. There are several existing solutions, how monitoring can be realized. The instrumentation for instance can be implemented by adding sensors to the orchestrations or querying the audit trail. In practice, the vendors of a BPEL or other workflow engine provide their own, platform-specific monitoring solutions. Our goal is to support the different approaches to implementing the monitoring of orchestrations by means of a platform-independent monitoring model, which can be transformed (automatically) into specific models. This paper focuses on the generation of the specific BPEL process definitions (or code) extended by sensors as a first step towards this objective.

The CIMs are used for specifying the business processes along with the goals in a way that is independent from the IT support. Thereby, various modeling notations are available for modeling business processes. As pointed out in [BaMR04] an MDA approach allows supporting all kinds of notations by transforming them into an appropriate meta notation like the Business Process Definition Metamodel (BPDM) [FrGJ04]. Due to the fact that the BPMN standard defines mapping rules for a BPMN-to-BPEL transformation and is already supported by a couple of development tools, it currently represents one of the most appropriate platform-independent models [EmWA06]. Thus, we decided to also employ it for modeling the functional aspects on the CIM level.

On the PIM layer the functional models of the CIM layer are transformed into BPMN-based orchestration models describing the (machine) executable business processes as well as the external services invoked within the orchestration.

The high-level PPI specifications on the other hand are transformed into PPI monitoring models. Basically, these monitoring models define components for measuring the specified PPIs on the basis of metrics and monitoring information, which are derived from the functional model.

The orchestration models along with the monitoring models are transformed into a platform-specific instrumented orchestration model. Concretely, the applied transformation adds sensors to the orchestration model which are required for evaluating the specified PPIs. And furthermore, the specifics of the selected SOA implementation are added to the instrumented orchestration models. Finally, the platform-specific code, namely the executable BPEL process definitions, is generated from platform-specific model. These process definitions are extended by monitoring sensors which pass the information on to a static monitoring infrastructure (MI). This MI provides a uniform interface to a BAM system and offers the required monitoring information for evaluating the specified business process goals.

The MI comprises several monitoring agents, which are possibly arranged in a hierarchical way. Furthermore, several existing technologies, like for instance Web-Based Enterprise Management (WBEM) could be employed for implementing this infrastructure. In this paper, we limit the scope to a simple and static MI consisting of one monitoring agent for each specified PPI. A more flexible design would require the creation of adequate models for describing the details of the MI and the usage of the employed platform.

4 Computation-independent Modeling of Business Processes and Mapping to Platform-independent Orchestration Models

For the formal modeling of business processes various notations are available. As pointed out in section 3, we decided on BPMN for modeling business processes in a computation-independent way. Thereby, the BPMN defines both the (graphical) notation and the semantics of a process through the definition of a so-called Business Process Diagram (BPD) [EmWA06].

Figure 2 provides an overview of the underlying meta-model. The full specification is available in [Wh04].

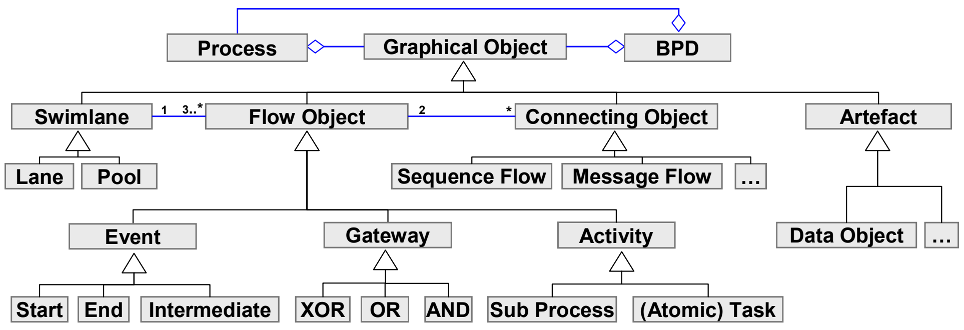


Figure 2: BPMN Meta-Model for Defining Business Processes in a Computation-Independent Way

In this section, the elements of a BPD which are important for understanding the monitoring models presented in the following sections are briefly introduced. In general, a BPD is comprised of activities performed by a certain organizational unit or role, a control flow between the contained activities, artifacts, like for instance data objects, which are processed within the activities, and events that may occur during process execution. The control flow is modeled by means of *Connecting Object* elements, especially the *Sequence Flow* along with *Gateway* elements for modeling parallel flows and conditioned branches. The process participants are modeled through the construct *Swimlane*. A *Pool* indicates that the containing process is owned by an independent organizational unit, whereas a *Lane* within a *Pool* specifies that a certain role is responsible for the covered process parts. An exchange of messages between two organizational units is described through a *Connecting Object* of type *Message Flow*. By means of the element *Sub Process* a process may be further segmented.

Using these modeling elements a business analyst is able to model business processes from a business perspective without regarding the involved IT. These models are refined and restructured to platform-independent orchestration models. Thereby, each activity is broken down to an executable task. In fact, the orchestration model must not contain a non-executable activity. The BPMN specifies the following concepts for defining orchestrations (

Figure 3).

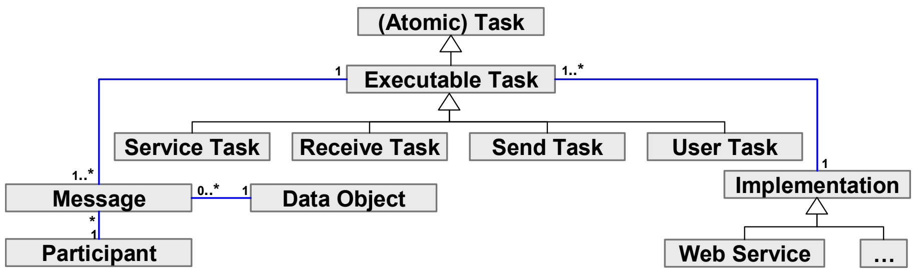


Figure 3: BPMN Meta-Model for Defining Orchestrations

Hence, an *Executable Task* generally involves the exchange of one or more *Messages* associated with a *Participant*. Furthermore, an *Implementation* (*Web Service* or other adequate implementations) is specified. The BPMN standard then distinguishes between four different kinds of executable tasks. A *Service Task* involves a *request-response* or *one-way* invocation of an operation provided by an external service. A *Receive Task* on the other hand awaits a message from an external client offered as a service operation by the orchestration itself. In case such an operation is of the type *request-response* a *Send Task* is used for returning the reply message to the requestor. A *User Task* comes into play if the orchestration involves human interaction. Within these tasks, a task message is assembled and delivered to an external task manager, which amongst other things allocates the tasks to a responsible employee, provides a user interface for the processing of the task and returns a task to the respective process as soon as it is finished. The standardization of this mechanism is currently being tackled by WS-BPEL4People initiative [KK+05]. The BPMN elements previously introduced for the CIM are also used within the PIM to model the orchestration's control flow. Note that the orchestration model may be very different from a computation-independent model. Therefore, the transformation can from our point of view not be automated.

5 Specification of the Process Performance Indicators and the PPI Monitoring Model

This section introduces newly developed meta-models for specifying PPIs for a process in a computation-independent way as well as a platform-independent PPI monitoring model which additionally defines how the specified PPIs are measured within the respective orchestration. The meta-models are based on existing approaches presented in [BKPS04; PAVB04]. In contrast to [BKPS04] we limited the scope to the specification of measurable, quantitative indicators and disregarded qualitative process or business goals. For the evaluation of the associated goals, an external BAM system could be employed

Figure 4 shows our meta-model for the specification of PPIs at the CIM level.

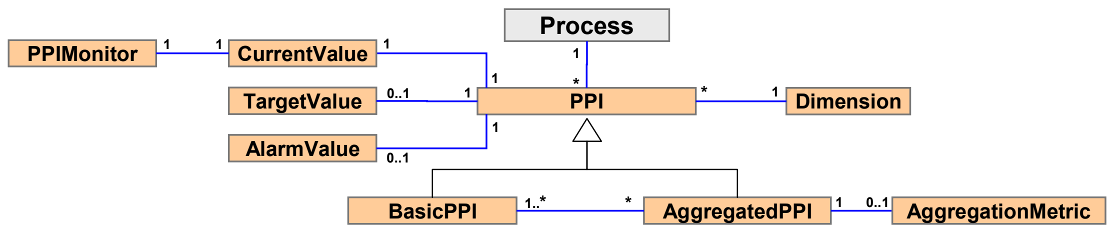


Figure 4: Meta-model for Performance Indicator Specification

A PPI is attached to the concept *Process* as part of the computation-independent process model. Optionally a *TargetValue* indicating the objective as well as an *AlarmValue* defining a threshold for an intervention may be specified. The PPI is further characterized by assigning a *Dimension*. Thereby information like the data type, the direction (e.g. ascending or descending) and the unit of the value are specified. The calculation of the mandatory *CurrentValue* on the basis of run-time information provided by the underlying orchestration is handled by the *PPIMonitor*. This aspect is tackled within the scope of the PPI monitoring model. Furthermore, we distinguish between basic and aggregated PPIs. A *BasicPPI* represents an atomic indicator, which can be measured within a single process instance whereas an *AggregatedPPI* spans multiple instances and is either evaluated through an *AggregationMetric* operating on basic PPIs (e.g. mean or variance) or directly calculated by the respective *PPIMonitor*.

Having the PPIs specified on the CIM level as a next step, the platform-independent *PPIMonitor* has to be defined in order to obtain a full PPI monitoring model tailored to the monitoring of orchestrations. The meta-model presented in Figure 5 has to be seen as an extension of the PPI specification meta-model.

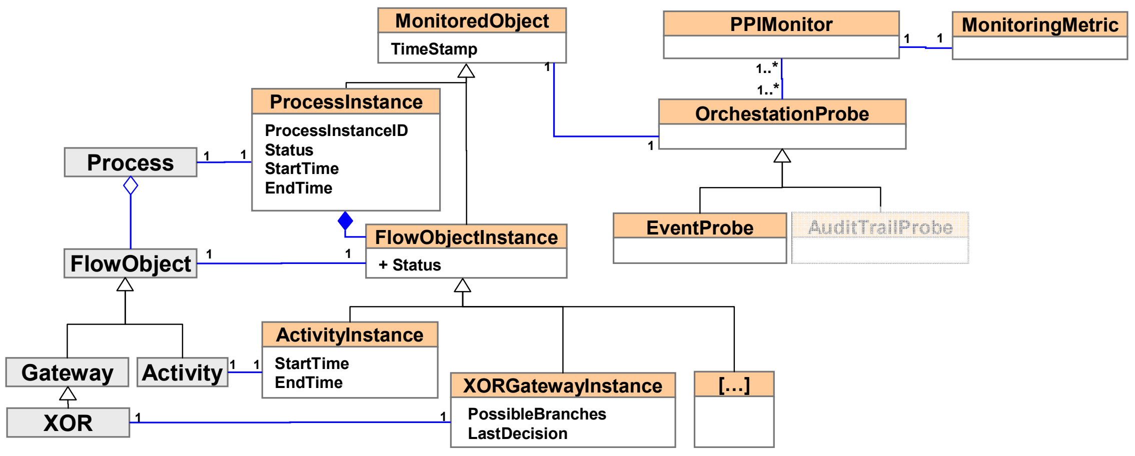


Figure 5: Meta-Model for Specifying the PPI Monitoring Model

Basically, a *PPIMonitor* operates on one or more managed objects of the respective orchestration and determines the desired PPI as-is value by means of a predefined *MonitoringMetric*. The managed objects thereby represent a management view on the process and hence capsulate information relevant for management [HeAN99]. As in our case the management functionality is limited to the monitoring of PPIs; the concept is termed *MonitoredObject.* The *MonitoredObject* of type *ProcessInstance* for example delivers information about the running process instance, like its current *Status* (e.g. "active" or "completed"), its cycle time (*StartTime*, *EndTime*) and the *ProcessInstanceID*, which is in orchestrations usually determined on the basis of a predefined *Correlation Set*. Within the scope of a process instance the monitoring can be further extended or refined to *FlowObjects* the process contains. Hence, a *MonitoredObject* of type *FlowObjectInstance* is introduced which may not exist without a *ProcessInstance*. The monitoring information required for a *FlowObject* depends on its concrete type. Hence, for each monitoring-relevant *FlowObject* a correspondent *MonitoredObject* is defined, as for example *ActivityInstance* or *XORGatewayInstance*. Whereas in the case of an *Activity* from a monitoring perspective the cycle time is of interest, for a *Gateway* of type *XOR* we would i.e. like to know the last decision. Depending on the PPIs that should be monitored, this information model for processes has to be further extended.

To retrieve the desired monitoring information (e.g. state updates) for a specified *MonitoredObject* from the underlying orchestration engine, an adequate instrumentation is required. The instrumentation is realized by means of *OrchestrationProbes.* Thereby, the information can be either gathered on the basis of the audit trail provided by the engine or events that are fired within the orchestration itself. Thus, a general distinction can be made between an *EventProbe* and an *AuditTrailProbe*. In the next section, the necessary BPMN extensions for defining and realizing an instrumentation based on *EventProbes* as well as the corresponding transformation (i.e. model merge) of the orchestration model along with the PPI monitoring model will be discussed. The generation of an *AuditTrailProbe* is not taken into consideration within this paper.

6 Transformation of the Orchestration and PPI Monitoring Model into an Instrumented Orchestration Model

To obtain monitoring information by means of *EventProbes* an extension of the BPMN-based orchestration meta-model is required which allows for the specification of the accordant instrumentation (Figure 6).

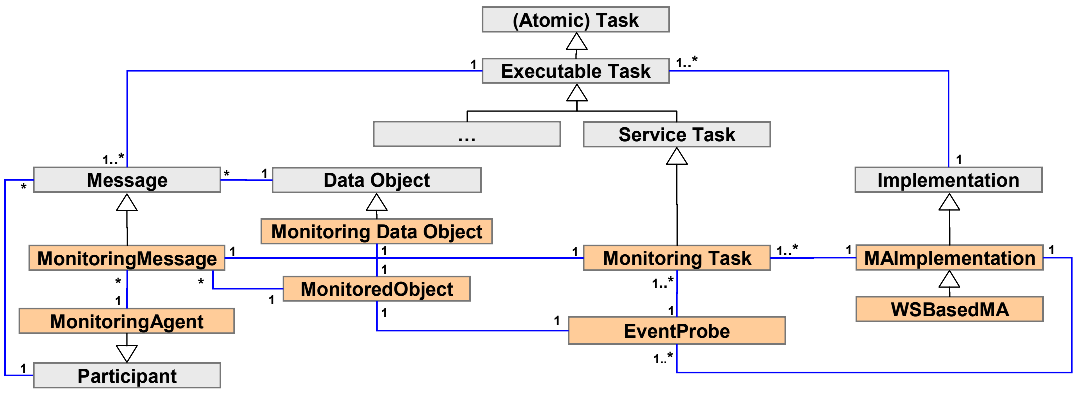


Figure 6: Extended BPMN Meta-Model for Specifying Instrumented Orchestrations

Thus, the monitoring information of a *MonitoredObject* provided by an *EventProbe* is gathered on basis of *MonitoringMessages* delivered by *MonitoringTasks*. A *MonitoringMessage* thereby contains a *Monitoring Data Object,* which only represents the BPMN version of the *MonitoredObject* (see Figure 5) and holds information about the current state. A *Monitoring Task* is a special kind of *Service Task.* But in contrast to those it only provides a *one-way* communication to the associated monitoring agent (MA), which is implemented by means of the *MAImplementation.* This implementation particularly realizes one or more *EventProbes,* meaning that it receives *MonitoringMessages* sent by a process instance through *Monitoring-Tasks* that belong to a distinct *EventProbe*. The probe update is then passed on to all associated *PPIMonitors*, which instantly calculate their *CurrentValue* on basis of the *MonitoredObject's* state information provided by the probe by applying the predefined *MonitoringMetric.* The *MAImplementation* may rely on web services or other technologies. As we focus on a SOA implementation on the basis of BPEL and web services, we target a *WSBasedMA.*

The *MonitoringsTasks* required for an *EventProbe* have to be placed at appropriate positions in the existing orchestration model. These positions depend on the concrete type of the *MonitoredObject* the probe is responsible for. In the following we will explain the basic idea of how the instrumented BPEL orchestration model is created from the orchestration model along with the PPI monitoring model (Figure 7). The approach is exemplified using the simple case of monitoring an activity.

The upper pool depicts a very simple orchestration model comprising of two activities of type *Executable Task* (*et1* and *et2*) which are executed in a sequence. The activity *et1* should be monitored on basis of an *EventProbe* providing the respective *MonitoredObject* of type *ActivityInstance.*

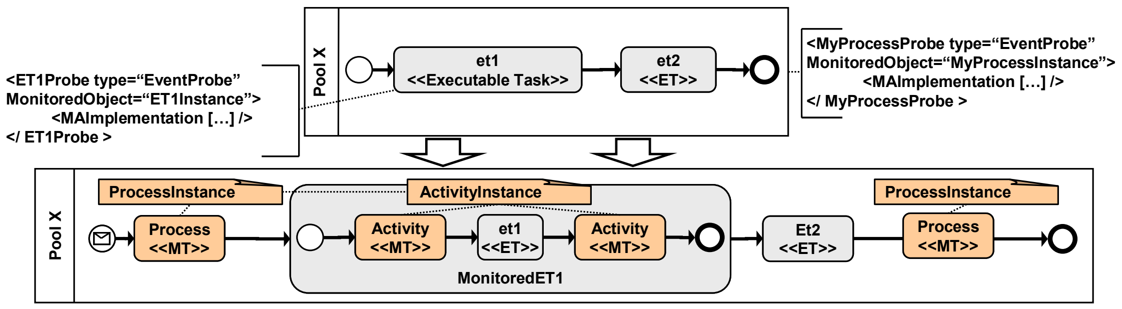


Figure 7: Mapping to Instrumented BPEL Orchestration Model

As an *ActivityInstance* may not exist without a *ProcessInstance* (see Figure 5) and the identifier of the running process instance is required for correlating the probes with the associated PPI, respective *EventProbes* for the whole process (*MyProcess*) as well as for the executable task *et1* have to be defined on basis of the PPI monitoring model and the instrumented orchestration model. As the evaluated modelling tools that support BPMN do not allow for an extension of the underlying BPMN meta-model, we decided to realize the association between concepts of the two models by means of BPMN annotations holding an XML-based definition of the *EventProbe*. This definition comprises all aspects needed for (automatically) creating the BPMN instrumentation on basis of *Management Tasks*. These annotations have to fully match the *EventProbes* specified within scope of the PPI monitoring model.

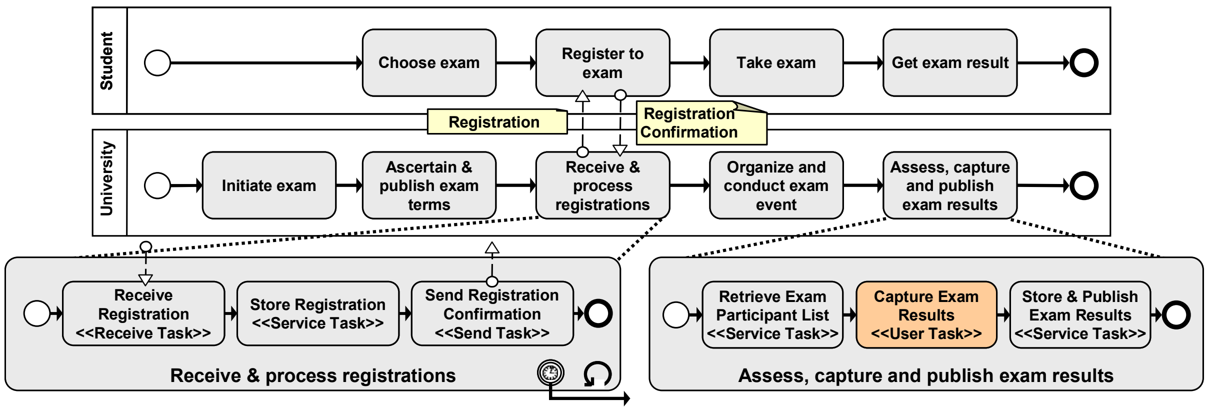
The transformation of the annotated orchestration model works as follows: In case of the *EventProbe* for the whole process two *MonitoringTasks* are added to the orchestrations model, namely one right after the *StartEvent* and one just before the *EndEvent*. The first *Monitoring-Task* provides information about the determined process instance identifier and the starting time whereas the second one only adds the end time.

The instrumentation of an activity in performed in a similar manner. The only difference is that instead of the activity, a new sub process holding the activity itself along with the required *MonitoringTasks* is created and inserted into the orchestration model. Within the scope of the *MonitoringTasks* added before and after the actual activity, an *ActivityInstance* object is assembled or updated and in each case sent to the responsible MA. As indicated by the association between the *ActivityInstance* object and the *ProcessInstance* object, the process instance information is also delivered within the according *MonitoringMessage*. Furthermore, the associated *EventProbe* has to be included. This is accomplished by providing a fixed message part within each *MonitoringMessage* holding this meta-information.

It becomes clear, that for each *MonitoredObject* a fixed procedure for adding the necessary instrumentation can be identified. Hence, the automation of these procedures can be realized by applying adequate model transformations.

7 Case Study: Development of a Monitored Orchestration for the Management of Examinations

The approach put forward in this paper has been applied to a practical scenario developed in the context of the project "Karlsruher Integriertes InformationsManagement" (KIM) [JuMa05], which targets the process and service-oriented redesign of a university's business processes along with the supporting IT. We particularly focused thereby on the business process within the scope of the examination management. Figure 8 shows a simplified computation-independent process model along with the refined platform-independent orchestration model for supporting the activities within the examination management lifecycle.

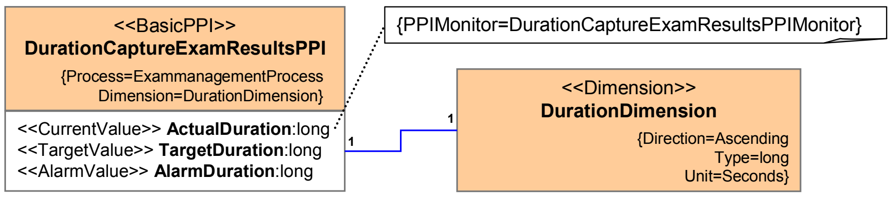


Student
Choose exam
Register to exam
Take exam
Get exam result
Registration
Registration Confirmation
University
Initiate exam
Ascertain & publish exam terms
Receive & process registrations
Organize and conduct exam event
Assess, capture and publish exam results
Receive Registration <<Receive Task>>
Store Registration <<Service Task>>
Send Registration Confirmation <<Send Task>>
Receive & process registrations
Retrieve Exam Participant List <<Service Task>>
Capture Exam Results <<User Task>>
Store & Publish Exam Results <<Service Task>>
Assess, capture and publish exam results

Figure 8: Process and Orchestration Model of the Examination Lifecycle Management

To demonstrate the approach, we limit the explanations to the activities relevant for the orchestration model. The orchestration is initiated after the university has decided to conduct an examination. In next step the terms for the exam are ascertained, transferred to the orchestration and published. Subsequently, registrations from students are received and processed by the orchestration. After the exam event has been organized and conducted, the exam results have to be assessed, captured and published. The capturing and publishing of the results is also supported by the orchestration. For this purpose, the final participant list is retrieved from the *EMService* within a *Service Task.* Afterwards, a *User Task* for the capturing of the results is initiated. As soon as all results are available they are returned to the orchestration by the employed task management service and stored through a *ServiceTask*.

Due to the fact, that the exam results are required promptly for generating certificates and evaluating preconditions within the registration process for further exams, one key performance indicator is the students' waiting time for their results. Hence, a university wide policy defines that the capturing of the exam results must not exceed 3 weeks. If the results are still not available after 2 weeks, a reminder should be sent to the person in charge. Figure 9 shows the formalized PPI based on an UML profile for the presented PPI specification model.

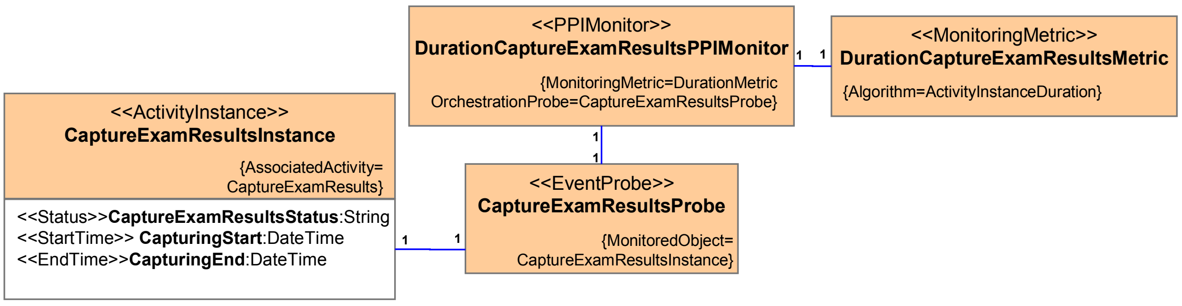


<<BasicPPI>>
DurationCaptureExamResultsPPI
{Process=ExammanagementProcess
Dimension=DurationDimension}
<<CurrentValue>> ActualDuration:long
<<TargetValue>> TargetDuration:long
<<AlarmValue>> AlarmDuration:long
{PPIMonitor=DurationCaptureExamResultsPPIMonitor}
1
1
<<Dimension>>
DurationDimension
{Direction=Ascending
Type=long
Unit=Seconds}

Figure 9: Specification of the PPI „DurationCaptureExamResults"

The assignments of a *Process*, a *Dimension* as well as the associated *PPIMonitor* for the attribute of stereotype *CurrentValue* to the specified PPI are realized through *TaggedValues*, either for on level of the stereotyped class or attribute. The target and the alarm value could also be realized through *TaggedValues*, but for the sake of flexibility we chose to define them as attributes. As soon as a concrete instance of the *DurationCaptureExamResultsPPI* is created (which has to be done for each newly created process instance) these values have to be assigned with three and two weeks.

As defined within the PPI specification the attribute *ActualDuration* is determined by the *DurationCaptureExamResultsPPIMonitor*. How this monitor works is specified by means of the PPI monitoring model is depicted on Figure 10.



<<PPIMonitor>>
DurationCaptureExamResultsPPIMonitor
{MonitoringMetric=DurationMetric
OrchestrationProbe=CaptureExamResultsProbe}
1
1
<<MonitoringMetric>>
DurationCaptureExamResultsMetric
{Algorithm=ActivityInstanceDuration}
1
1
<<ActivityInstance>>
CaptureExamResultsInstance
{AssociatedActivity=
CaptureExamResults}
<<Status>>CaptureExamResultsStatus:String
<<StartTime>> CapturingStart:DateTime
<<EndTime>>CapturingEnd:DateTime
1
1
<<EventProbe>>
CaptureExamResultsProbe
{MonitoredObject=
CaptureExamResultsInstance}

Figure 10: Monitoring Model for the PPI „DurationCaptureExamResults"

The *DurationCaptureResultsPPIMonitor* operates on a *CaptureExamResultsProbe*. This probe provides state information about the *MonitoredObject* of type *CaptureExamResultsInstance* which is associated with the process activity *Capture Exam Results* (see Figure 8). The actual value is calculated by executing the linked *MonitoringMetric*. In this case, the metric uses a generic algorithm for calculating the duration of an arbitrary *ActivityInstance*. This somewhat simple algorithm works as follows:

```
If (ActivityInstance.Status  equals  “Active”)
        ActualDuration = TimeSpan(CurrentTime, ActivityInstance.StartTime)
Else if (ActivityInstance.Status equals “Completed”)
        ActualDuration = TimeSpan(ActivityInstance.EndTime,  ActivityInstance.StartTime)
```

To retrieve the state information about the *CaptureExamResultInstances* for all running process instances, the appropriate *MonitoringsTasks* are added to the orchestration model. As described in section 6 in case an *ActivityInstance* should be monitored, a new sub-process is created containing a sequence of the actual activity along with *MonitoringTasks* before and after die activity is performed. Figure 11 shows the instrumented orchestration model for the sample process.

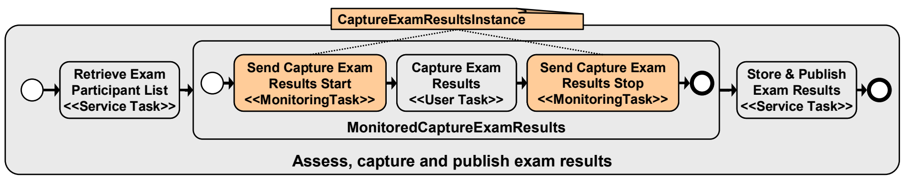

CaptureExamResultsInstance
Retrieve Exam Participant List <<Service Task>>
Send Capture Exam Results Start <<MonitoringTask>>
Capture Exam Results <<User Task>>
Send Capture Exam Results Stop <<MonitoringTask>>
MonitoredCaptureExamResults
Store & Publish Exam Results <<Service Task>>
Assess, capture and publish exam results

Figure 11: Instrumented Orchestration Model

For the added *MonitoringsTasks* some additional properties are specified, for instance the endpoint reference to the employed *MAImplementation*. To create the executable BPEL process definition we used the BPEL export functionality of the employed modelling tool (Borland Together 6.0). As, amongst other things, the required variable assignments are missing in the generated code and *UserTasks* are not supported at all, we had to manually add these aspects. For this purpose, we used the Oracle BPEL Designer along with the corresponding BPEL engine Oracle BPEL Manager. The final code for the sub process *MonitoredCaptureExamResults* is as follows:

```
<scope name=“MonitoredCaptureExamsResult“>
        [...]
    <assign name="setCaptureExamResultsInstanceStart> [...] </assign>
    <invoke name="sendCaptureExamResultsStartMessage" partnerLink="agentService"
    operation="processMonitoringMessage" inputVariable="captureExamResultsInstance" [...] "/>
    <!-- UserTask: CaptureExamResults-->
    <scope name="CaptureExamResults" […] xmlns:task="http://services.oracle.com/bpel/task">
          <partnerLinks>
                    <partnerLink name="userTask" partnerLinkType="task:TaskManager"
                                   partnerRole="TaskManager" myRole="TaskManagerRequester" [...]/>
          </partnerLinks>
          [...]
    </scope>
    <assign name="setCaptureExamResultsInstanceCompleted"> [...] </assign>
    <invoke name="sendCaptureExamResultsCompletedMessage" partnerLink="agentService"
      operation="processMonitoringMessage" inputVariable="captureExamResultsInstance" […]/>
[…]
</scope>
```

Besides the XML representation of the *MonitoredObject* (here *CaptureExamResulsInstance*) the *MonitoringMessage* contains an additional message part holding information about the process instance ID along with the process ID. This information is required by the invoked monitoring agent for correlating the messages with the respective instances of the associated probe as well as the PPI monitor.

Our implementation of the monitoring infrastructure only consists of one monitoring agent for the presented PPI which handles both, the provision of probes and the calculation of the PPI. It would also be possible to decouple the provision of probes from the PPI monitoring. In doing so, the integration of an existing BAM system would be easier. The provided probes would have to be translated into events the BAM system understands within the scope of an appropriate adapter.

8 Conclusion & Outlook

In this paper, we presented the first steps towards a model-driven development of orchestrations along with the infrastructure for the monitoring of predefined PPIs. Thereby, the presented meta-model for the specification of the PPI monitoring along with the extension of the BPMN meta-model for modeling the required instrumentation and the sketched methodology for an automated generation of this instrumentation represent the main contribution of this work. In our future research we will try to achieve a fully automated generation of the orchestration instrumentation along with the monitoring infrastructure based on UML profiles for the meta-models and an adequate transformation language. Furthermore, we aim to extend the monitoring to a larger variety of *MonitoredObjects*, including more complex transactions with embedded sub transactions, and corresponding types of PPIs, especially aggregated PPIs. In this case, the design of monitoring infrastructure would also have to be revised. In this context, we are planning on using WBEM standards (especially the Common Information Model (CIM) [BuST00]) in conjunction with WS-Management [DMTF06] for implementing the monitoring infrastructure. This would enable an integration of the underlying application management and hence allow for an integrated monitoring of business goals and the involved IT.

References

[AaHW03] *van der Aalst, Wim M. P.; ter Hofstede, A. H. M.; Weske, M.:* Business Process Management: A Survey. In: Lecture Notes in Computer Science Band 2678. Springer-Verlag, Berlin 2003, S. 1-12.

[ACDG03] Andrews, T.; Curbera, F.; Dholakia, H.; Goland, Y.; Klein, J.; Leymann, F.; Liu, K.; Roller, D.; Smith, D.; Thatte, S.; Trickovic, I.; Weerawarana, S.: Business Process Execution Language for Web Services 1.1. ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, 2003-05-05, Abruf am 2006-07-21.

[BKPS04] *Bayer, F.; Kühn, H.; Petzmann, A.; Schlossar, R.:* Service-oriented Architecture for Business Monitoring Systems. In: Panetto, H. (Hrsg.): Proceedings of the Workshop on Web Services and Interoperability (WSI'05). Hermes Science Publishing, Genf 2005, S. 127-134.

[BaMR04] *Bauer, Bernhard; Müller, Jörg P.; Roser, Stephan:* A Model-Driven Approach to Designing Cross-Enterprise Business Processes. In: Lecture Notes in Computer Science Band 3292. Springer-Verlag, Berlin 2004, S. 544–555.

[BuST00] *Bumpus, W.; Schweitzer, J. W.; Thompson, P.:* Common Information Model, John Wiley & Sons Ltd., New York, 2000.

[DMTF06] *Distributed Management Task Force (DMTF)*: Web Services for Management (WS-Management) Specification Version 1.0.0a, http://www.dmtf.org/standards/published_documents/DSP0226.pdf, 2006-04-05, Abruf am 2006-07-21.

[FrGJ04] *Frank, J. H.; Gardner, T. A.; Johnston, S. K.:* Business Process Definition Metamodel -Concepts and Overview. http://www.bpmn.org/Documents/BPDM/BPDM%20Whitepaper%202004-05-03.pdf, 2004-04-08, Abruf am 2006-07-21.

[EmWA06] *Emig, Christian; Weisser, Jochen; Abeck, Sebastian:* Development of SOA-Based Software Systems – an Evolutionary Programming Approach. In: International Conference on Internet and Web Applications and Services ICIW'06, Guadeloupe, 2006.

[HaRa01] *Hauck, R.; Radisic, I.:* Service Oriented Application Management — Do Current Techniques Meet the Requirements?. In: New Developments in Distributed Applications and Interoperable Systems, Proceedings of the 3rd IFIP International Working Conference (DAIS 2001). Kluwer Academic Publishers, Krakau 2001, S. 295–304.

[HeAN99] *Hegering, H.-G.; Abeck, Sebastian; Neumair, B.:* Integrated Management of Networked Systems: Architecture, Tools, Operational Implementation. Morgan-Kaufmann, San Francisco 1999.

[JeSC03] *Jeng, J.-J.; Schiefer, J.; Chang, H.:* An Agent-based Architecture for Analyzing Business Processes of Real-Time Enterprises. In: Proceedings Seventh IEEE International Enterprise Distributed Object Computing Conference (EDOC'03). 2003, S 86-97.

[JuMa05] *Juling, W.; Maurer, A.:* Karlsruher Integriertes InformationsManagement. In: Praxis der Informationsverarbeitung und Kommunikation (2005) 3, S. 169-175.

[KHSW05] *Koehler, J.; Hauser, R.; Sendall, S.; Wahler, M.:* Declarative Techniques for Model-driven Business Process Integration. In: IBM System Journal (2005) 44, S. 47-65.

[KKLP05] *Kloppmann, M.; Koenig, D.; Leymann, F.; Pfau, G.; Rickayzen, A.; von Riegen, C.; Schmidt, P.; Trickovic, I.:* WS-BPEL Extension for People (BPEL4People).

ftp://www6.software.ibm.com/software/developer/library/ws-bpel4people.pdf, 2005-07-01, Abruf am 2006-07-21.

[Le03] *Leymann, Frank:* Web Services - Distributed Applications without Limits, Business, Technology and Web. Leipzig 2003.

[LeRS02] *Leymann, Frank; Roller, Dieter; Schmidt, M.-T.:* Web Services and business process management. In: IBM Systems Journal (2002) 41, S. 198-211.

[Mc03] *McGregor, Carolyn:* A Method to Extend BPEL4WS to Enable Business Performance Measurement. http://www.cit.uws.edu.au/research/reports/paper/paper03/TR-CIT-15-2003.pdf, 2003-06-01, Abruf am 2006-07-21

[McSc04] *McGregor, Carolyn; Schiefer, Josef:* A Web-Service based framework for analyzing and measuring business performance. In: Information Systems and e-Business Management. Springer-Verlag, Berlin 2004, S. 89-110.

[MiMu01] *Miller, Joaquin; Mukerji, Jishnu:* Model Driven Architecture. http://www.omg.org/docs/ormsc/01-07-01.pdf, 2001-07-01, Abruf am 2006-07-21.

[MuRo04] *zur Muehlen, M.; Rosemann, M.:* Multi-Paradigm Process Management. In: Grundspenkis, Janis; Kirikova, Marite (Hrsg.): Proceedings of CAiSE'04 Workshops - 5th Workshop on Business Process Modeling, Development and Support (BPMDS 2004). Riga 2004, S. 169-175.

[PAVB04] *Pignaton, R.; Asensio, J. I.; Villagrá, V.; Berrocal, J. J.:* Developing QoS-aware Component-Based Applications Using MDA Principles. In: Eighth International Enterprise Distributed Object Computing Conference (EDOC 2004), 2004, S. 172-183

[Wh04] *White, S. A.:* Business Process Modeling Notation. BPMN 1.0. http://www.bpmn.org/Documents/BPMN%20V1-0%20May%203%202004.pdf, 2004-5-03, Abruf am 2006-07-21