Ad-hoc Management Capabilities for Distributed Business Processes*

Sonja Zaplata, Dirk Bade, Kristof Hamann, Winfried Lamersdorf
Computer Science Department, University of Hamburg
{zaplata,bade,hamann,lamersdorf}@informatik.uni-hamburg.de

Daniel Straßenburg[1], Benjamin Wunderlich[2]
[1]CoreMedia AG, [2]Geoflags GmbH, Hamburg, Germany
[1]Daniel.Strassenburg@coremedia.com, [2]b.wunderlich@geoflags.de

Abstract: Advanced business processes are mostly distributed and require highly flexible management capabilities. In such scenarios, process parts often leave their initiator's direct sphere of influence – while management requires both *monitoring* as well as *instant reaction capabilities* anytime during the overall execution of the process. However, realizing such functions is often difficult, e.g. due to the heterogeneity and temporal disconnectivity of participating execution systems. Therefore, this contribution proposes a two-tier concept for monitoring and controlling distributed processes by representing a process management system as a *manageable resource* according to the WSDM standard. Based on a minimal shared model of management capabilities it allows to define customized events and processing rules for influencing business processes executed on a remote (and even on a temporarily disconnected) process management system. Applicability is demonstrated by a scenario-based evaluation on distributed WS-BPEL and XPDL processes.

1 Motivation

Today's competitive business collaborations highly benefit from transparency and visibility of the status of their business process networks. Within a single organization, *business activity monitoring (BAM)* technologies support real-time analytics about running business transactions and allow for the correlation of events for causalities, aggregates, thresholds, and alerts based on user-defined preferences. The analyzed information is delivered in (near) real time and provides an important basis to detect failures and non-compliances, to react to them accordingly and in sufficient time and, thus, to optimize the execution of processes in whole or in part.

However, to stay competitive and provide new value-added products and services, often also *cross-organizational collaborations* become necessary which span business processes between several organizations and different process management systems. Here, not only atomic resources such as employees, machines and services, but also the execution of a process instance itself can be distributed. Figure 1 shows examples for

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

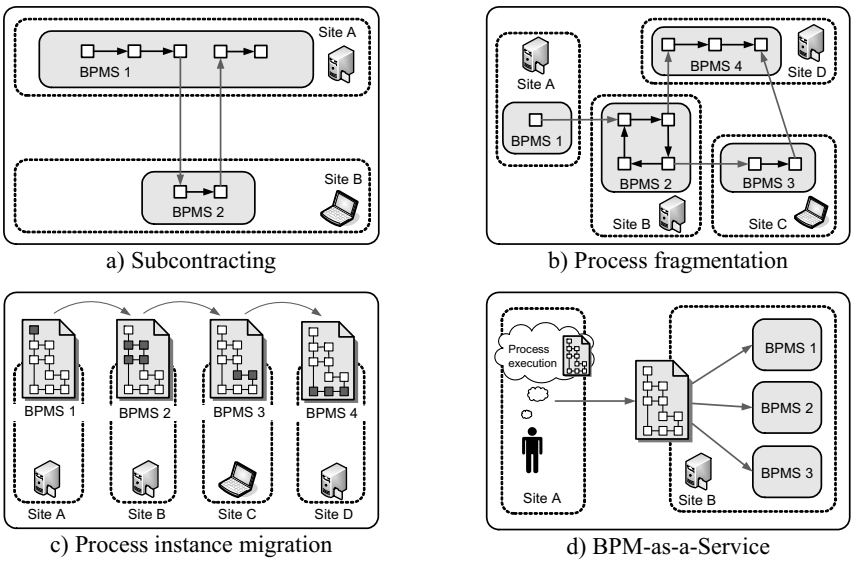


Figure 1: Examples for the distribution of process execution [vdA00,ZK+10]

such distribution, realized as *subcontracting* of single process parts, *fragmentation* of processes, *process instance migration* from one workplace to another, and *BPM-as-a-Service* where the execution of a business process is fully operated by an external provider [vdA00, ZK+10]. Considering resulting *cross-organizational processes*, it is even more relevant to gather information about the execution on the remote system, because such participants may be dynamically selected or exchanged on the basis of their workload, context or QoS parameters. Cross-organizational monitoring and controlling capabilities are thus important to support the controllability of active process parts and – based on the collected information and experiences – optimize the distribution and execution of upcoming processes.

As a current drawback, today's BPM systems mostly consider monitoring and controlling of single centralized process executions, are often heterogeneous and do not provide standardized runtime monitoring or management APIs [vL+08]. Therefore, an integration of runtime monitoring information from different source systems is hardly possible yet. Required possibilities to also take influence on a remote process execution and to react to the observed behavior of the process (preferably in real time) are still challenging. This paper therefore aims at a concept and supporting infrastructure to flexibly collect information about the execution of process parts running on a remote system, to automatically process this information and to predefine and execute timely reactions to detected complex situations where ever necessary. Therefore, the rest of this paper is organized as follows: Section 2 motivates the need for a cross-organizational management approach for distributed business processes based on existing and related work. Section 3 presents a two-tier middleware extension for process management systems in order to support the provision and utilization of ad-hoc management capabilities. Section 4 analyzes applicability of the proposed concepts and Section 5 concludes the paper.

2 Background and Related Work

Research in the area of monitoring constitutes an important part of the management of distributed processes. Relevant previous approaches use the idea of extending existing process models by weaving in additional activities which call-back to a central monitoring system (e.g. [BGG04, BG05]). Thereby, the user initiating the process can build up his own monitoring system according to his individual preferences, e.g. accessing the status of the process instances, the duration of activities or the actual navigation of control flow. Neither agreements nor the adaptation of the partner system are required. Nevertheless, many information interesting for distribution, such as current system properties (e.g. the location where the process is executed, or the current workload of the engine), the number and type of deployed (but uninitiated) process models or the occurrence of internal process instance events (e.g. errors) are not visible. Furthermore, inserting additional activities after each functional task may result in a huge monitoring overhead – in the worst case expanding the original process description up to its double size, potentially decreasing execution performance considerably and mixing business logic and technical management logic in an undesired way [MWL08]. If the monitoring system becomes unavailable, the execution of the functional process is delayed or is likely to fail. Furthermore, appropriate actions depending on the results of the gathered information are limited, because running process activities cannot be influenced, e.g. cancelled, as the remote system does not provide an interface for such operations.

In order to preserve efficiency of process execution while at the same time allow quick and adequate reactions to predefined situations, the subscription to *process-related events* and their corresponding processing are attached a high importance [vA09]. While a *primitive event* simply represents some relevant change of a certain property (e.g. change of a process's status, a workload shift, etc.) *complex events* represent some arbitrarily complex inference of information from one or more primitive or other complex events [Luc02]. This is achieved by so called *Complex Event Processing (CEP)* and *Event Stream Processing* techniques. As an example, the ESPER project[1] addresses business process management and automation, i.e. process monitoring, BAM, reporting exceptions and operational intelligence. It uses an SQL-based query language to express rules and provides a rule engine for complex event processing. In order to address the heterogeneity of possible event sources, common agreements and standards for the representation of events (e.g. IBM's *Common Base Event*) as well as for the specification of complex event inference statements are required. The specification of reactions to (complex) events is equally important. Rule-based approaches, especially *Event-Condition-Action* rules, are widely used but due to manifold application domains neither the event, nor the condition or action representations are commonly agreed on. Wetzstein et al. [WK+10] therefore present an approach to support monitoring in service choreographies based on agreements about events to be shared with other partners and using complex event processing to derive key performance indicators for the overall process execution. However, this approach still only focuses on the subscription to events, but neither offers the possibility for requesting monitoring information on demand nor for initiating ad-hoc management actions.

[1] http://esper.codehaus.org/

As a foundation for the interoperability of heterogeneous workflow management systems, the Workflow Management Coalition (WfMC) has issued the *Workflow Reference Model* which also contains administration and monitoring tools for the management of users, resources and processes [WfM98]. An overview of management operations is proposed here, especially for user and role management (e.g. changing privileges of users), audit management (e.g. querying logs and audit trails), resource control (e.g. concurrency levels, thresholds), process supervisory functions (e.g. termination of process instances) and process status functions (e.g. fetching information about process instances). Associated specifications for achieving workflow interoperability (e.g. Wf-XML [SPG04]) are more detailed, but still focus on sending, installing and retrieving process definitions to/from a remote process engine. However, the idea of exchanging process management related information based on a common model by using standard web services is very interesting. The *Web Services Distributed Management (WSDM)* standard develops this idea a bit further. It allows specifying an arbitrary resource (e.g. a printer) as a so-called *manageable resource* which offers a set of resource-dependent properties accessible by a self describing service interface [OAS06b]. Providing such resource properties requires specifying a model as a mutual understanding of the resource to be managed. However, only a model supporting the *management of web services (MOWS)* [OAS06a] themselves has been developed. The first part of the work presented here therefore proposes a model to exchange basic information and control options for business process management systems involved in cross-organizational collaborations. A similar basis has been proposed by van Lessen et al. [vL+08] for WS-BPEL process instances. In this paper, however, we are extending this idea by also including relevant process model and process engine properties as well as related events, and, as the second part of this work, presenting a loosely coupled management component in order to analyze and process the received information either on-site or remotely.

3 A Two-Tier Process Management Middleware

The approach presented here proposes a *service-based common management interface* and uses *complex event processing* in order to specify user-defined management rules and actions. Therefore, a business process management system is considered as a *manageable resource* according to the understanding of WSDM. Defining the elements and properties of this manageable resource, relevant functionalities such as data retrieval, event subscription and control options are exposed as services and can be integrated in a standard registry and thus in existing and future applications. Based on that, an additional component uses the resulting management services and events in order to specify user-defined monitoring and management (re-)actions.

3.1 Tier 1: Process Management System as a Manageable Resource

In order to find an adequate basis for a common understanding of the elements and attributes relevant for distributed process management, an analysis of several current practical and theoretical approaches and systems as well as abstract models and concrete products for traditional and distributed business process management has been carried

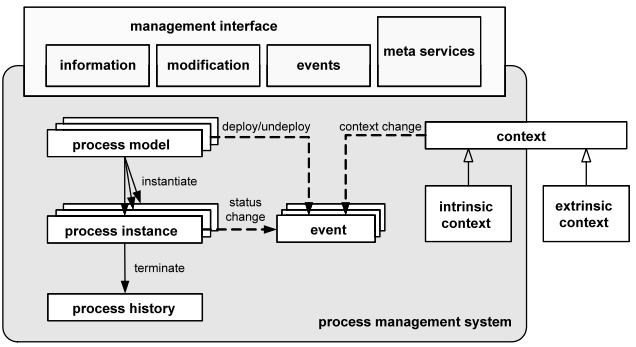


Figure 2: Process management system as manageable resource

out. The analysis has lead to the identification of most relevant management entities and a resulting basic model which is shown in Figure 2. It holds the process management system as a manageable resource which can be accessed by a service-based management interface either by pulling read-only information about its entities (*information interface*), by asking for manipulation of entity values (*modification interface*) or for receiving events emitted by the entities (*event interface*). To provide information about the management itself, a *meta interface* allows to access information about capabilities and operations for the configuration of the three aforementioned interfaces.

In the context of distributed process management, the proposed entities of a process management system include the *process models* which are deployed to the process engine, the *process instances* which are instantiations of these models (representing the processes which are currently running), and the *process histories* which contain information about processes which have already been finished (cp. Figure 2). Furthermore, to consider the special characteristics of distributed process management (such as mobility, cooperation and dynamic assignment) the process management system has a relevant *context* comprised of the *intrinsic context* of the process engine (e.g. system properties such as workload or service availability), and the *extrinsic context* (e.g. location or weather). Both types of context can either be static (e.g. the identity of the system's owner) or dynamic (e.g. the current workload). Generic context models which can be customized for such application are e.g. proposed by [KZTL08].

Figures 3 and 4 show refinements of the entities *process model* and *process instance* as manageable resources. Here, a process consists of *activities* which are connected by *transitions* to define a control flow (potentially restricted by a *condition*), and a set of *data fields* using a certain *data type* also defined within the process model. Furthermore, participants can be predefined as required *performers* for specific activities which are especially important in the context of distributed process management as this element contributes to the selection of partner systems. A process instance (cp. Figure 4) extends its process model by implementing the associated runtime information. Most importantly, this involves the *status of the running process* (e.g. executing, suspended, in error), the specific *values of the data fields*, the *status of the activities* (e.g. running, skipped), the *evaluation* of transition conditions (true/false) and the *actual performers* who are finally executing the activities. Finally, *process histories* reflect the entities of

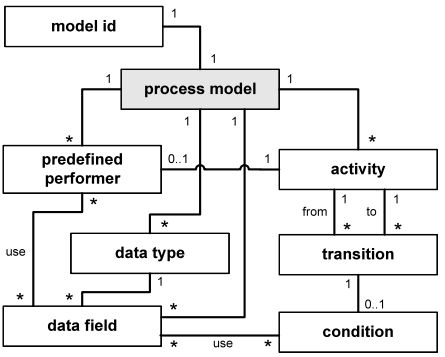


Figure 3: Process model as manageable resource

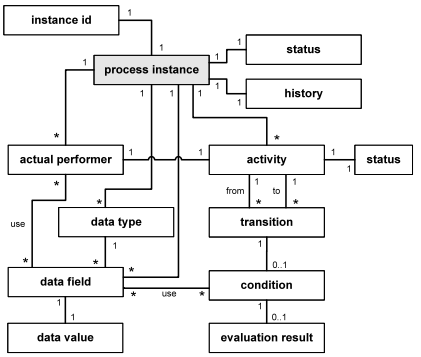


Figure 4: Process instance as manageable resource

the terminated process instances in a static way (not depicted). In order to enable a distributed execution and a respective management, all three entities need to have a unique identifier for the correlation of requests.

According to existing approaches such as Wf-XML [SPG04], all entities contain a number of sub entities and individual atomic properties, e.g. a process engine has a current workload expressed as the number of running process instances and CPU load, or a process instance activity has a start time, a duration and an end time. Creation of entity instances and changes of their properties' values are effecting the associated *events*. In order to allow manageability, exchanging information about a resource property requires a uniform and unambiguous representation and interpretation of values, e.g. represented as standard or complex data types, and a metric. Furthermore, the *modifiability* (e.g. read or read-write), the *availability* of the property (e.g. before, during or after execution of the process or the activity) and the *mutability* and *frequency of updates* should be specified. Due to space limitations and similarity to existing meta models, the enumeration of relevant entity properties and events should, however, not be part of this paper.

3.2 Tier 2: Management Component for Complex Event Processing

Providing informational and manipulative services and the possibility to subscribe to events based on a common understanding such as established in Section 3.1, arbitrary management applications can be composed in order to collect information and react to even complex situations in a user-defined way. The general methodology to support such operations is based on a loosely-coupled management component which is depicted in Figure 5: The user who is initiating a controlled distributed execution of a process (in the following called the *customer*) takes the original process description to be executed and creates an additional document (*management document*) which holds the customer's requirements for the management of this process (*management rules*). Here, the term *management* subsumes all objects, situations and operations which are, from the customer's perspective, relevant for the correct execution and administration of the distributed process and are not covered by the functional business process description. Relevant objects are the entities of the model presented in Section 3.1, e.g. process models, instances and data objects. Situations and operations are described within the manage-

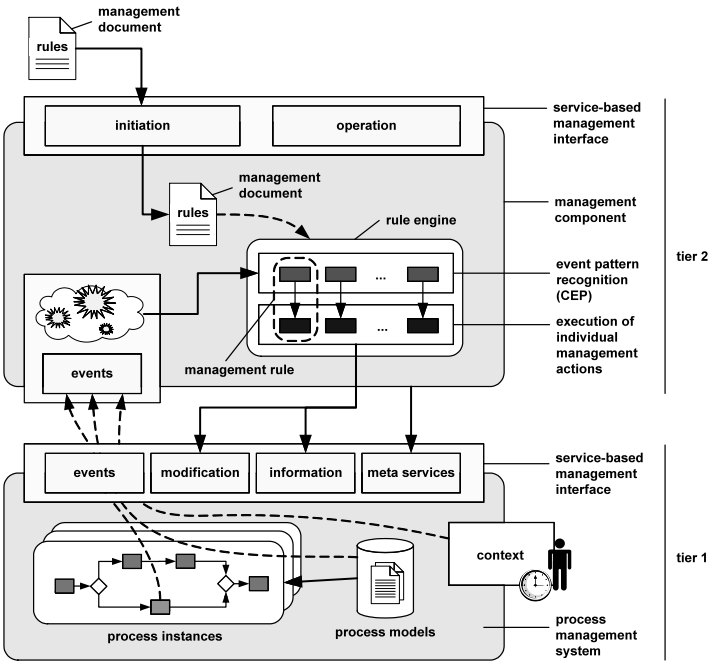


Figure 5: Management component to support customized management actions

ment document as complex situations and actions. An example for such a situation-action pair is monitoring the duration of a specific activity (*object*) and, in case a specified amount of time has passed and no progress becomes visible (*situation*), to restart the activity (*action*). However, also monitoring rules that do not influence the execution of the process are possible (e.g. after each activity, its performer, duration and current location should be logged) or distribution decisions can be supported (e.g. if the workload exceeds a specified threshold, the process should be transferred to a process engine with a better capacity).

Listing 1 shows the general syntax of a management rule as a part of the management document. For administration purposes, each management rule has a *name* and, optionally, a *description*. The *rule pattern* holds an event pattern to determine if a complex situation has occurred or not. The *rule action* specifies the service to be executed, including parameters for the service call if necessary. Encapsulated as a composite service, even complex actions can be defined. Furthermore, arbitrary system-external services, such as sending an email, can be referenced.

```
1  MANAGEMENT-RULES
2    MANAGEMENT-RULE
3       NAME         : <String>
4       [DESCRIPTION : <String>]
5       RULE-PATTERN : <Event-Pattern>
6       RULE-ACTION  : <Service-Invocation>
7    END MANAGEMENT-RULE
8  ...
9  END MANAGEMENT-RULES
```
Listing 1: Structure of management rules

The management document is passed to the management component and the system returns a management identifier as a reference to the management document. Thus, the customer can adapt the management rules later if necessary. Interpretation and processing of the management rules are executed by a rule engine (cp. Section 3.3). Relevant events are subscribed and event notifications are passed to the rule engine in order to perform the pattern matching. If a specified pattern is recognized, the rule engine initiates the execution of the corresponding actions.

Besides the management rules (line 18 of Listing 2), the management document holds additional information required for correlation, assignment and execution of processes and rules. Therefore, *general information* (lines 3-7 of Listing 2) contains a *management endpoint* as the unique identifier of the process management system to be supervised, and a *management mode* which defines when the management should end. As management starts with passing the management document (i.e. the rule engine starts listening to the specified events) it can either terminate automatically when all process instances are finished (management mode = "system") or it can explicitly be terminated by the customer (management mode ="user"). The latter is relevant if the management should also observe the process engine as a candidate for further distributions, if process history data is required later (i.e. for evaluation) or if the process model can be instantiated again from the outside (which is e.g. the case for WS-BPEL processes which deploy their own service interface for the initiation of new process instances). In case of an automatic termination, the customer can optionally specify a *notification endpoint*. Termination of the management plays an important role, because here all rule patterns have to be removed from the rule engine, and the associated event subscriptions have to be cancelled.

```
1  MANAGEMENT-DOCUMENT
2
3    GENERAL-INFORMATION
4        MANAGEMENT-ENDPOINT        : <URL>
5        MANAGEMENT-MODE            : "system"|"user"
6        [NOTIFICATION-ENDPOINT     : <URL>]
7    END GENERAL-INFORMATION
8
9    INSTANTIATION-INFORMATION
10      PROCESS-MODEL-REFERENCE    : <STRING>
11      LOCAL-INSTANCE-REFERENCE   : <STRING>
12      [INSTANTIATION-TIME        : <DATE>]
13      [INSTANTIATION-DELAY       : <INTEGER>]
14      [INSTANTIATION-PARAMETERS]
15      [BLOCKING-EVENT-TYPES]
16   END INSTANTIATION-INFORMATION
17
18   MANAGEMENT-RULES
19
20 END MANAGEMENT-DOCUMENT
```

Listing 2: Structure of management document

The part of the *instantiation information* (cp. lines 9-16 in Listing 2) contains relevant data about the process instances. It holds the reference to the associated process model (*process model reference*) and a placeholder for the process instances (*local instance reference*) which do not exist at the time of deployment, but which need to be referenced within the rule patterns and actions for instance management. In case only one instance is distributed the process can optionally be started immediately, at a specified point of *time* or after a specified *delay*. In this case, also the *parameters* for process instantiation have to be passed. Finally, the management document specifies which events should be

able to block the execution of a process instance (cp. line 15 in Listing 2). Such *blocking events* are required to enable immediate reactions where the process engine must not continue execution of the process. Blocking events are only relevant for the monitoring of process instances and are thus also part of the instantiation information.

In order to allow customizing the location of decision-making, processing of events and derivation of management reactions can take place on the remote system or at the customer's site. In the first case, the management document has to be transferred to the remote system, all events are caught locally and management actions are carried out by the partner system. Here, in general, management data does not have to be transferred over the network and consequently, the delay resulting from management is minimized. This option is well suited in case mobile participants are involved and network connection is temporarily unavailable. In the second case, all information and/or events are transferred to the customer and management reactions are determined here. This is required if e.g. management decisions are confidential or need approval by a human operator. However, this strategy may decrease performance of process execution. Furthermore, the customer has to run and maintain the management component in order to make decisions – which may not be desired e.g. in the case of BPM-as-a-Service scenarios (cp. Figure 1(d)). An attractive alternative is, however, the combination of both strategies, e.g. having general monitoring data collected and processed by the remote system and calling-back to the customer only in case of infrequent severe problems.

3.3 Implementation

In order to use the WSDM framework, the model of relevant characteristics and relationships of process management system, process models, instances and context has been represented as a *WSDM resource properties document* which is specified in XML. Furthermore, XML-Schema (XSD) is used to specify structure and data type of each property. According to the meta-description established in Section 3.1, an example for the representation of the property "WorkloadInfo" is depicted in Listing 3 and 4. The resource properties can be accessed using the WSDM web service operations *GetResourceProperty* and *UpdateResourceProperty* as well as a set of additional operations, e.g. for cancellation of process instances, which are altogether included in the associated web service description (WSDL). The WSDL file also contains the location where the service can be accessed, e.g. a URL. Finally, WS-Notification (WSN) topics for subscription of events have been specified and included, and for each event it is specified whether it should be allowed to block the process execution in order to enable a direct reaction.

```
1  <Property name="tns:WorkloadInfo" modifiability="read-only"
2            mutability="mutable" meta:mutfreq="meta:minutes">
3      <meta:availability>meta:always</meta:availability>
4  </Property>
```

Listing 3: Example for the meta-description of a BPM resource property within WSDM

```
1  <xsd:complexType name="WorkloadInfo">
2    <xsd:sequence>
3       <xsd:element name="RunningInstances" type="xsd:int">
4       <xsd:element name="CPULoad" type="xsd:float">
5    </xsd:sequence>
6  </xsd:complexType>
```

Listing 4: Example for the schema definition of a BPM resource property within WSDM

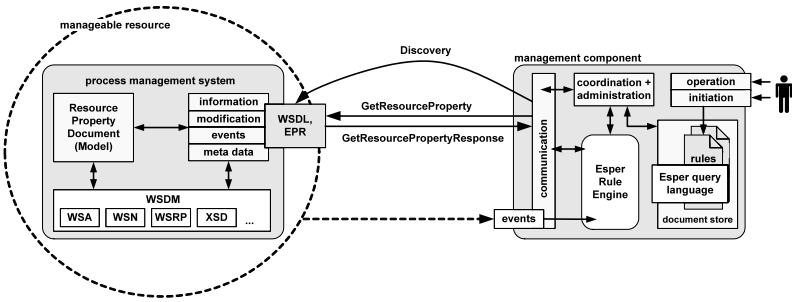


Figure 6: Prototype implementation using WSDM and Esper

The left side of Figure 6 shows an overview of the implementation with WSDM. In order to interact with the manageable resource, the management consumer only requires the *Endpoint Reference (EPR)* of the process management system, i.e. the information from the WSDL file. After that, read-only or write requests as well as subscriptions for events can be carried out by requesting a property of the resource properties document. The requested resource respectively replies with a *GetResourcePropertyResponse* resp. *UpdateResourcePropertyResponse* message in case of a successful operation or with an exception in case of an error. In case the consumer wants to subscribe for an event, besides the topic also the type of the event can be specified, i.e. if the event should block the execution of the process or not.

On side of the management component (cp. right side of Figure 6), the management document is interpreted and stored for the time of its validity. The *coordination and administration module* is now responsible for the discovery of the specified management endpoint, to subscribe for the required events and to manage time constraints and deadlines. The received event stream is processed and tested against the user-defined event patterns. In the prototype implementation, the complex event processing is done by the existing Esper rule engine (cp. Section 2). As Esper expects expressions in the SQL-based Esper query language, the abstract *event pattern* part of the management rules (cp. line 5 of Listing 1) is represented by the respective terms of this language. A simplified example for a complete management rule is presented in Listing 5. In this case, the process variable named *deadline* within all instances of the process model with id='1' is updated if the execution of an activity with id=4 takes longer than 60 seconds.

```
1  <Rule name="ExtendDeadline">
2    <Trigger>
3       SELECT * FROM PATTERN
4       [EVERY (e1=ActivityStarted(activityId="4",modelId="1")
5               -> TIMER:INTERVAL(60 SEC)
6               -> NOT e2=ActivityFinished(activityId="4",modelId="1"))]
7       WHERE e1.instanceId=e2.instanceId
8    </Trigger>
9    <Action>
10     <Service epr="http://example.com/bpms.wsdl" operation="UpdateResourceProperty">
11       <Param type="PropertyName"><Value>DataField</Value></Param>
12       <Param type="ProcessInstanceID"><Value>${e1.instanceId}</Value></Param>
13       <Param type="DataFieldName"><Value>deadline</Value></Param>
14       <Param type="DataValue"><Value>60</Value></Param>
15     </Service>
16   </Action>
17   <Trigger>
```

Listing 5: Example management rule with Esper event pattern and WSDM service invocation

4 Evaluation

So far, the prototype implementation has been applied to two existing process management systems: first to the DEMAC [ZKL09] process engine which uses XPDL processes and supports the runtime migration of process instances and, second, to the Sliver [HH+06] process engine which uses a subset of WS-BPEL processes. The following example scenario is used to show the main observations and results also in comparison to two previous approaches.

4.1 Example Scenario

Figure 7 shows an example from the *eErasmus eHigher Education (eEH)* project [JL06], which is an international exchange program of higher education institutes among EU countries. In order to facilitate a uniform exchange of students joining this program, a standardized process is proposed for participating universities. The simplified functional process used here involves subcontracting the host university for *approving the credentials* necessary for taking courses there, allowing to take *courses and exams* until a specified deadline and *preparing the credentials* achieved at the host university in order to acknowledge them at the home university.

The distributed execution involves several management requirements which are *expected in advance*, i.e. before execution of the process starts: (R_1) The host university is paid a certain amount of money for each student and for the associated administration effort. Therefore, the duration of each activity executed by the host university has to be logged. (R_2) In order to handle potential errors in time, the home university wants to be sure that the foreign university has received the sub-process and is able to execute it, and, (R_3) if duration of an activity expected as critical (here *preparation of credentials*) exceeds the average time for executing a task, (R_4) the activity should be skipped in order to at least allow the control flow of the process to return to the calling system. (R_5) As it sometimes happens that the deadline for taking courses is adapted by the host university, e.g. because the student gets ill, the home university wants to know about such events in order to avoid automatic removal from the home register of students. In addition, there are a number of *unexpected occurrences* during the runtime of this rather long-running (i.e. several months) process: First, a financial aid program asks about the status of the student's overall study (R_6). Second, the student has married and his/her name has to be adapted (R_7).

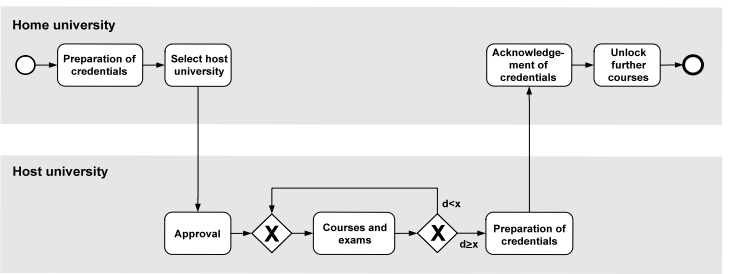
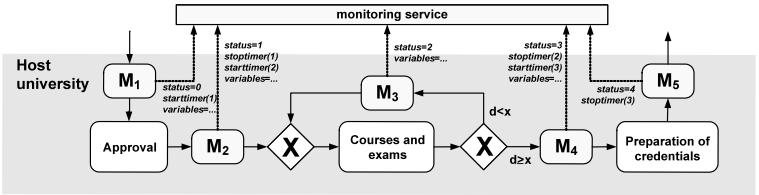


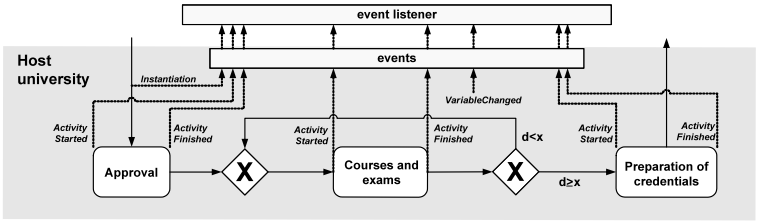
Figure 7: eErasmus example process

4.2 Comparison and Results

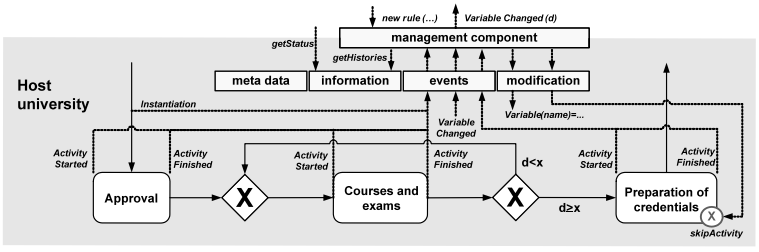
Figure 8 shows the realization of the monitored process instance with weaving of monitoring activities, event-based monitoring, and the ad-hoc management approach proposed here. Results are summarized in Table 1. It shows that monitoring aspects which are known in advance, such as measuring of the duration of predefined activities, the start of instance execution and the observation of variable value modifications can be realized by the design time insertion of respective monitoring activities (timer activities and passing of variables values to the central monitoring service) and by the event-based monitoring and the ad-hoc management approach (by subscription of the respective events). The detection of abnormal activity duration can be realized by the ad-hoc management as a complex rule involving also additional information about previous process instances executed on this system and calculating their average time of execution. This is neither possible by a system which makes use of events only (the events of other process instances have not been captured before) nor by activity weaving (histories of other process instances are not visible in the monitored process instance). Skipping critical activities is also a problem, because event-based monitoring does not offer control functionalities at all, and activity weaving cannot skip crashed activities by weaving an "end activity" because in this case control flow will not reach this activity.



a) Approach using activity weaving



b) Approach using event-based monitoring only



c) Approach using ad-hoc management

Figure 8: Different realizations of the scenario-based management requirements

Table 1: Applicability of management requirements during execution of the example process

Management requirement	Ad-hoc management	Event-based monitoring	Activity weaving
(R_1) Duration of activities	+	+	+
(R_2) Instance started	+	+	+
(R_3) Detect critical activity duration	+	o	o
(R_4) Skip critical activity if necessary	+	-	-
(R_5) Observe variable value	+	+	+
(R_6) Ad-hoc status retrieval	+	o	o
(R_7) Ad-hoc variable value modification	+	-	-

Considering unexpected occurrences, the ad-hoc management shows its biggest advantage: The status retrieval can be made by calling the process's resource property *process status* and interesting *data values* directly. Both activity weaving and event-based monitoring can provide this data only in case a monitoring activity is inserted after each functional activity resp. all available events have been subscribed. Therefore, it is more or less a coincidence if such requests can be fulfilled as they cannot be determined in advance and relevant properties have to be weaved/subscribed before runtime. The ad-hoc variable modification is also not possible because of missing runtime modification operations. However, even by using the ad-hoc management approach, the process manager has to be careful not to violate the integrity of the process. Therefore, in case of the modification of the student's name, the process manager should abstain from calling the modification interface directly, but better update the management document by inserting a new rule – e.g. to wait until the current activity is finished (subscribe *ActivityFinished* as a blocking event), perform the modification, and then resume execution.

Considering non-functional characteristics, it shows that desired separation between business logic and management logic can be achieved by event-based and ad-hoc management approaches (as the original business process does not have to be changed), but not by activity weaving (cp. Figure 8). Especially in the context of mobile process management, the approach of activity weaving furthermore proves to be very instable (i.e. if the monitoring service is not available, the process execution is delayed or even fails). For the event-based approach, no delays effected by the management are visible at all – however no reactions are possible and thus events can be emitted in parallel to an ongoing process execution without delay. Compared with both event-based and ad-hoc management approaches, activity weaving has, however, the advantage that no system modifications, security mechanisms or agreements are necessary.

5 Conclusion

In today's highly dynamic business networks customized monitoring and controlling options for distributed business processes gain increasing importance. This paper advances existing approaches for the management of such processes by presenting a concept to not only passively observe the behaviour of business processes running on a remote process management system but also to enable quick automatic and spontaneous reactions on the basis of a service-based management interface. Thereby, the presented approach allows for increased flexibility during process execution and for the integration of valuable functionalities of remote process management systems which have not been exploited before. However, process managers have to be aware of their respective new

potential, e.g. by influencing process execution during runtime which may lead to undesired side effects. Furthermore, the presented approach has to be secured so that both provider as well as the consumer of distributed process management are protected in a sufficient way. Therefore, the conceptualization and application of protective measures and customizable security and privacy mechanisms are an important part of future work.

References

[BG05] Luciano Baresi and Sam Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In 3rd Int.Conf. on Service-Oriented Computing, pages 269-282, 2005.

[BGG04] Luciano Baresi, Carlo Ghezzi, Sam Guinea. Smart Monitors for Composed Services. 2nd Int. Conf. on Service-oriented Computing, pages 193–202, 2004.

[Esp10] EsperTech. Esper - Performance. http://docs.codehaus.org/display/ESPER/Esper+performance, 2010.

[HH+06] Gregory Hackmann, Mart Haitjema, Christopher D. Gill, Gruia-Catalin Roman. Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. 4th Int. Conf. on Service-Oriented Computing, pages 503–508, 2006.

[JL06] R. Vermer, Juliet Lodge. Case Study e Erasmus eHigher Education (eEH). Sixth Framework Programme R4eGov, Deliverable WP3 D1-D4, 2006.

[KZTL08] Christian P. Kunze, Sonja Zaplata, Mirwais Turjalei, Winfried Lamersdorf. Enabling Context-based Cooperation: A Generic Context Model and Management System. Business Information Systems (BIS 2008), pages 459-470, 2008.

[Luc02] David Luckham. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Professional, 2002.

[MWL08] Daniel Martin, Daniel Wutke, Frank Leymann. A Novel Approach to Decentralized Workflow Enactment. Enterprise Distributed Object Computing, pages 127–136, 2008.

[OAS06a] OASIS. Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1. Standard Specification, 2006.

[OAS06b] OASIS. Web Services Distributed Management: Management UsingWeb Services (WSDM-MUWS) 1.1. Standard Specification, 2006.

[SPG04] Keith D. Swenson, Sameer Pradhan, Mike D. Gilger. Wf-XML 2.0 XML Based Protocol for Run-Time Integration of Process Engines. WfMC, 2004.

[vA09] Rainer von Ammon. Event-Driven Business Process Management. Encyclopedia of Database Systems, pages 1068–1071, 2009.

[vdA00] Wil van der Aalst. Loosely coupled interorganizational workflows: modeling and analyzing workflows crossing organizational boundaries. Information and Management, 37(2), 2000.

[vL+08] Tammo van Lessen, Frank Leymann, Ralph Mietzner, Jorg Nitzsche, Daniel Schleicher. A Management Framework for WS-BPEL. European Conference on Web Services (ECOWS 2008), pages 187–196, 2008.

[WfM98] WfMC. Workflow Management Coalition Audit Data Specification. WFMC-TC-1015, Workflow Management Coalition, 1998.

[WK+10] Branimir Wetzstein, Dimka Karastoyanova, Oliver Kopp, Frank Leymann, Daniel Zwink. Cross-Organizational Process Monitoring based on Service Choreographies. 25th ACM Symposium on Applied Computing (SAC 2010), pages 2485–2490, 2010.

[ZKL09] Sonja Zaplata, Christian P. Kunze, Winfried Lamersdorf. Context-based Cooperation in Mobile Business Environments: Managing the Distributed Execution of Mobile Processes. Business and Information Systems Engineering, pages 301–314, 2009(4).

[ZK+10] Sonja Zaplata, Kristian Kottke, Matthias Meiners, Winfried Lamersdorf. Towards Runtime Migration of WS-BPEL Processes. 5th InternationalWorkshop on Engineering Service-Oriented Applications (WESOA'09).