Leveraging Model-Driven Monitoring for Event-Driven Business Process Control

Falko Koetter

University of Stuttgart IAT
Allmandring 35
Stuttgart, Germany
falko.koetter@iao.fraunhofer.de

Monika Kochanowski, Maximilien Kintz

Fraunhofer IAO
Nobelstraße 12
Stuttgart, Germany
firstname.lastname@iao.fraunhofer.de

Abstract: Event-driven business process management can help businesses to monitor and guarantee correct and efficient execution of their business processes. Two examples of this are using events to monitor processes and control execution. However, many companies lack mature, executable process models and rely on legacy software systems to execute processes. Creating a customized event infrastructure in such an environment is a cumbersome task. In previous work we detailed aPro, an architecture for model-driven design and implementation of business process monitoring. This paper describes how to use real-time monitoring data to control business process execution in real-time, extending the model-driven approach.
Keywords: Business Process Monitoring, Complex Event Processing, Compliance, Event-driven Business Process

1 Introduction

Today many companies have a need to improve the quality of their business processes. This need arises from volatile market conditions, increasing cost pressure, higher rate of innovations as well as regulatory compliance requirements. Efforts cover all parts of the business process lifecycle [Wes12], including process modeling and management (e.g. CMMI [GG03]), workflow execution, monitoring, and adaptive case management.

However, many companies have a low level of business process management maturity [RdBH04, PCBV11], lacking the capability to execute workflows and optimize their processes. Existing approaches for business process monitoring and execution control often build on top of executable process models running in a process engine[WSL09, BTPT06, BG05]. However, in our work with the industry, we found this to be the exception rather than the norm. Economic and organizational constraints prevent a switch of technology. Existing process models are often not linked to process execution in any way. They only serve documentation purposes. This can lead to discrepancies between process model and actual process, either because the model is out-of date or because the model represents the ideal rather than the actual state of affairs. Additionally, IT resources and budgets are often constrained. IT departments are busy managing daily operations and implementing

A. Baumgraß, N. Herzberg, G. Kappel, J. Mendling, A. Meyer, S. Rinderle-Ma (eds.): Proceedings on Inter-Organizational Process Modeling and Event Processing in Business Process Management, Vienna, Austria, 12-06-2014, published at http://ceur-ws.org

new products and requirements. Structural improvements without immediate gains are often deferred.

In previous work we developed the aPro architecture to develop and introduce business process monitoring in these heterogeneous, distributed environments [KK12, KK13]. In a model-driven approach, a monitoring infrastructure is created in three steps. In the first step a business process model is augmented by a platform-independent goal model, describing where to measure data as well as the Key Performance Indicators (KPIs) and goals of the process. This goal model is then used to automatically configure and deploy the aPro architecture in a web container[KWR12]. This includes monitoring data measurement by providing web services and interchange formats[KK12], processing via Complex Event Processing (CEP) rules[KK13] and display by configuring a dashboard[Kin12]. In the third step, existing systems are integrated using platform-specific monitoring stubs, making only minimal changes to the executing system necessary.

We applied this monitoring infrastructure in multiple use cases, including monitoring of process metrics, business metrics as well as compliance monitoring. In comparison to business monitoring, monitoring of compliance to rules and regulations presents a unique set of challenges. While the violation of business goals may be handled at the discretion of a company, violations in compliance should not occur at all and need to be handled immediately to avoid legal problems. As some real-world processes are executed in short timeframes and in high volumes, manual counteractions to goal violations may not be practical. Even if personnel are notified immediately, they may not act fast enough. Thus, an automated execution control is needed to initiate countermeasures to goal violations immediately after they occur.

In this work we will show how real-time monitoring data generated by aPro can be used to control business process execution in heterogeneous environments, providing the following contributions: We show how existing monitoring events and rules can be used for Business process control. WE extend our model-driven approach to encompass the modeling and creation of business control elements. We show how model-driven business process execution control in heterogeneous, distributed systems can be achieved. We implemented these concepts in a prototype.

2 Related Work

In this section we will present related work regarding event-driven process management and execution control. We have outlined related work in the area of process monitoring in previous work[KWR12, KK] and will focus this section on process control. Related work in the field of process adaptation and control focuses on executable process models running within a process engine. While in such homogenous environments these approaches provide a high degree of automation, they provide little assistance for implementing and configuring different process monitoring and execution environments.

[JMM12] propose an event-based approach, gathering events from a process engine, processing them in a CEP engine and providing results to a dashboard and the CEP

engine, thus creating a feedback loop used for process control. It was implemented using standardized formats and off-the-shelf-components. However, considerable effort was spent integrating components, converting data and defining rules and configuration. In comparison, a model-driven approach as presented in this work automates many of these tasks and is not limited to a process engine.

In [TZ11] running BPEL process instances are adapted using adaptation patterns. This approach is not limited to a specific process engine, as only monitoring and adaptation operations need to be implemented. It is used to change process models on the fly, for example by inserting a process fragment, to migrate processes during runtime, with monitoring information serving to adapt each specific process instance. Adaptations are not based on process goals or KPIs as in this work. While this approach provides more flexibility compared to previous approaches allowing only substitution of activities (e.g. [MRD08]), it is still limited to executable BPEL process models. In comparison, the model-driven approach in this work supports any execution environment that can be integrated via stubs.

[UGSC11] presents multiple approaches to control process execution in distributed systems. An *integration rule* allows measuring at the start and end of activities. *Assurance points* define a checkpoint within the process and allow monitoring and condition checking. *Exception rules* are used to implement countermeasures for exceptions. Finally, *invariant rules* evaluate an invariant throughout process execution and allow countermeasures as soon as it is violated. In comparison to this work, a way to define the different kinds of rules in a unified way and connect them to a graphical process model is missing. Process execution by decentral agents is currently supported, but process engines are not. In comparison, this work supports both, individually or in combination.

[MZD09] describes an approach to detect compliance violations during execution of business processes in a service-oriented architecture. Web service calls are handled as events, which are correlated to event trails and business activities, similarly to correlating events from measuring points in this approach. From these trails CEP rules are created to detect compliance violations. Compliance controls are assumed to be in place and can be invoked if a violation is detected. In comparison, this approach generates event rules directly from business (compliance) goals and provides assistance in implementing compliance controls. However, in a homogenous environment [MZD09] provides a higher degree of automation in the gathering of monitoring events. Similarly, the MASTER project[LPF+08] defines a compliance architecture reading events from services in a SOA, monitoring and assessing them and using the result for compliance enforcement, gathering monitoring data and propagating control actions similarly to this work. For this purpose, the MASTER project defines a high-level language for monitoring data and interfaces.

3 Concept

3.1 Conceptual Overview

This section gives a short overview of the aPro architecture for process optimization and related concepts [KK12, KWR12, KK13, KK].

Figure 1 shows the aPro architecture with the necessary components for monitoring (elements in grey show extensions for control). A BPMN [Obj09] process model is created in the modeling step and stored in a process repository. Depending on the implementation of the business process, the model is either directly executed by a process engine or the process model represents the process execution by one or multiple application systems.

Regardless how the process is executed, monitoring data needs to be gathered by so-called monitoring stubs. These monitoring stubs are used to integrate different kind of application systems, for example as a part of program code or a script. Whenever monitoring data is measured by a monitoring stub, it is sent to a monitoring web service, which in turn creates an event for monitoring data processing. These events are processed by a CEP engine, which aggregates measurements, calculates process goals and KPIs. Monitoring data is presented on a dashboard [Kin12].

To use this architecture for a specific process, configuration files for each component are needed. Using a model-driven approach, these do not need to be created manually, but can be created from a platform-independent model. A so-called goal model describing the monitoring is modeled alongside the process model in a graphical notation. In Figure 2 the modeling elements are shown. A measuring point may be attached to a BPMN element and indicates a measurement to be taken whenever this element is reached during execution. Parameters within the measuring point describe the data to be measured. Based on measured parameters, KPIs and goals can be calculated. A KPI is an important value of a process. A goal is a Boolean condition imposed on a parameter or KPI. If it evaluates to false, the goal is violated. A timing goal imposes a maximum time interval between two measurements of the same process instance. If the second measurement (and its associated activity) is not performed within a specified time after the first, the goal is violated.

After modeling, the process and goal model are stored in a so-called ProGoalML file, an interchange format used as an input to generate component specific configuration files. From this, monitoring stubs, data formats, event processing rules and dashboard layout are created automatically. For the end user, the monitoring infrastructure can be deployed directly from the modeling tool. However, depending on the systems running the process, monitoring stubs need to be deployed manually, though (semi-)automatic assistance is possible. In earlier prototypes, monitoring stubs can be created as a web forms, e-mail forms, JAVA classes, and batch scripts.

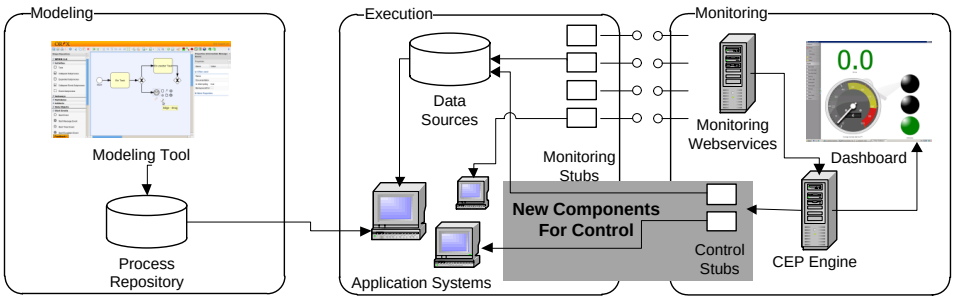

Figure 1: aPro architecture extended for process execution control, new elements marked grey

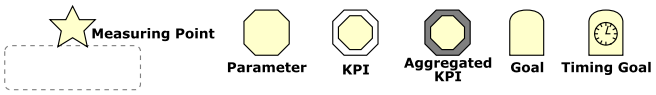

Figure 2: ProGoalML modeling elements

3.2 Running Example

Figure 3 top shows a simplified real-life insurance claim management process consisting of four steps. In the step *Receive claim* an insurance claim is received from a customer after an event of damage. This claim is usually sent in paper form and contains a description of the claim as well as a damage assessment from a local claim assessor which may or may not be associated with the insurance company. During the receive step, the data is parsed in a structured form and any containing images are extracted. This data is used in the next step *Process claim* to examine the claim based on multiple factors, for example if the damage is covered by the contract, if the damage description is plausible, if the claimed amount is realistic, etc. In the third step *Decide claim* a decision about the claim is made. A claim can be accepted, accepted with deductions or rejected. In the fourth step *Send claim* a letter with the result is sent to the customer. This process is executed in a heterogeneous environment involving a claim management system used by insurance personnel, a separate image extraction system, a service provider for text extraction and an expert system for claim assessment. As these systems handle a large volume of about 300 claims a day, of which approximately a third fulfills one or more special cases, a high degree of automation is desired.

To monitor the system with aPro, each activity has an associated measuring point. The step *Receive claim* is measured after the claim is received and the image extraction system is called. It measures the ID of the claim, the timestamp the claim is received and the amount of images that have been extracted from the document. An aggregated KPI *avg_img_count* is defined measuring the average

count of extracted images over the last. The goal *avg_img_above_zero* is fulfilled if this average is above 0. If the goal is not fulfilled, this indicates a problem with the image extraction system. An alert is generated and an administrator needs to investigate.

The second step *Process claim* is measured at the claim management system. Here, the *assessor_address* is measured, indicating who has assessed the claim. Additionally, a lookup of this assessor is performed within an internal database. If the assessor is found, his or her ID in the database is given as *assessor_db_id*. In case the assessor is unknown, this value is null and the goal *known_claim_assessor* is violated. The insurance company plans to keep its database of local assessors up-to-date, as they need to commission assessors themselves. This goal serves to identify new opportunities to add to the database.

The third step *Decide claim* is measured at the expert system after a decision about claim acceptance has been made. Here the *savings* generated in claim processing are measured. The value *details_missing* indicates if all details were available. While a claim can be valid with missing details, additional details may enable additional checks. For example the zip code of the claimant may not be extracted properly from the print document, limiting local comparisons of claimed damages. As the amount of claims cannot be handled manually, only select claims may be examined in detail. The goal *decision_viable* is used to select these claims. It fails if both details are missing and no savings have been generated.

The last step *Send claim* is measured at the claim management system after the claim answer letter is sent and represents the end of the process. The *timestamp* of the process end is measured.

The aggregated KPI *open_claims* calculates the current number of open claims by counting the measurements at *Receive claim* and subtracting the measurements at *Send claim*. The amount of open claims is an indicator of the general load of the system, as it rises with incoming claims and process duration.

The timing goal *notification_within_14_days* is related to a compliance requirement. The German Insurance Association (GDV) has issued a code of conduct regarding data privacy, which is to be followed by the claim management process [Ger12]. Requirement 5-8 of the code stipulates that a customer who has provided personal data has to be asked to agree with the use of this data for marketing purposes. This needs to occur with the next message to the customer or within a short time span (in this example 14 days). If the claim is answered within 14 days, the agreement can be asked within the letter. However, if claim processing takes longer, the goal is violated. This may happen under high load or in case of complex claims which are checked manually. After the goal is violated a separate letter asking for agreement needs to be sent to the customer manually.

While process monitoring helps to identify problems within the process by sending alerts, counteractions need to be performed manually. To lessen manual work and increase reliability, the monitoring events shall be used for process execution control. The following controls shall be implemented:

- Whenever the goal *avg_img_above_zero* is violated, the image extraction service may have encountered a problem. In this case the service shall be restarted automatically by running a batch script.
- If an unknown claim assessor is detected (goal *known_claim_assessor* is violated), the address of the claim assessor shall be sent to a different service, which automatically mails a questionnaire to the claim assessor, asking him to provide information about his or her services.
- If the goal *decision_viable* is violated for a claim, the claim shall be assigned to an expert within the claim management system.
- Depending on the amount of *open_claims* the length of time until automatic finalization of the claim shall be regulated so experts are not overwhelmed.
- If the goal *notification_within_14_days* is violated, a letter asking for agreement shall be created and sent automatically by a separate service.

In the following sections we will define the requirements to implement these control measures as well as the necessary modifications in aPro.

3.3 Requirements

In this section we will describe the requirements for process execution control both from a business and compliance perspective.

R1 *Goal violations create an event, which may be processed within the process.* From a compliance perspective, process control aims to counteract detected compliance violations. In aPro, compliance requirements are modeled as goals. Thus, goal violations need to trigger counteractions.

R2 *Whenever a KPI is calculated, an event is created, which may be processed within the process.* While the same applies to the business perspective, in a business use case in addition to the goals KPIs may be relevant for execution control. One example are the *open_claims* in the running example. While target values may be checked with a goal, a balance depending on the concrete value needs to be kept.

R3 *Events are created for both aggregated and nonaggregated goals and KPIs.* Goals and KPIs may be calculated for a single process instance or aggregated by time and length, which both may lead to control actions.

R4 *Events and control actions are modeled alongside the process and goal model.* As aPro uses a model-driven approach, process execution control needs to be part of the model.

R5 *All parts of the aPro architecture relevant for process execution control are configured automatically.* Events and control actions need to be taken into account when configuring the aPro architecture. This includes monitoring event creation and functionality for triggering control actions.

R6 *All modeling extensions need to work with executable process models.* As described in Section 1, aPro supports heterogeneous execution environments. This

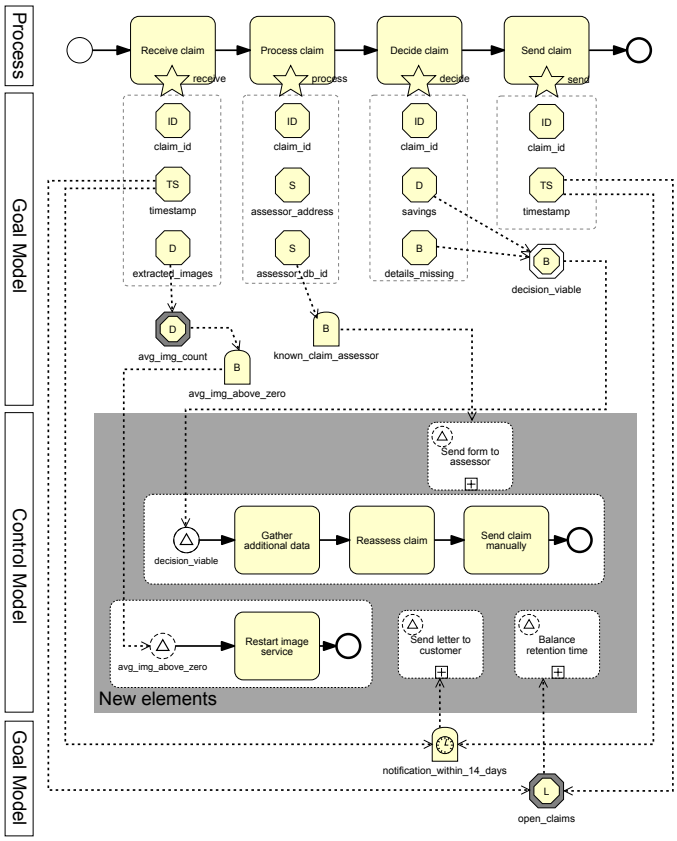

Figure 3: Claim management process (top) with goal model for monitoring the process (middle and very bottom, consisting of measuring points, KPIs, aggregated KPIs, and goals) and control model (marked grey, new elements introduced in this work consisting of events and event subprocesses for controlling the process)

means an event may trigger different kinds of control actions, depending on the system(s) executing the process. In case of a process engine, this may be achieved by simply catching a BPMN event [Sil09].

R7 *aPro assists in the creation and integration of control actions.* Without a process engine, control actions include performing an additional activity, calling a web service, executing a script or changing a configuration parameter. As these control actions need to be defined in a platform-specific way, they cannot be created automatically. Similar to the monitoring stubs for gathering measurements, control actions need to be integrated with the executing system. Thus, instead of full automation, semi-automatic assistance for control point creation needs to be provided.

3.4 Modeling extensions

To fulfill requirement **R4**, we need to extend the ProGoalML notation to include the definition of events and control actions. BPMN already contains modeling elements for event handling. Events can be thrown and caught to control process flow, escalate process instances and handle exceptions [Sil09]. To keep compatibility with executable process models (requirement **R6**), the existing event modeling capabilities should be used. To implement a control action, the action itself as well as the triggering event needs to be modeled. The event is thrown if a KPI is measured or a goal is violated. It is then caught to trigger the control action.

As monitoring data is gathered and processed in an event-based fashion using CEP, no separate modeling of events is necessary. From a conceptual point of view, control actions are not triggered by events, but by goals and KPIs. Thus, instead of explicitly modeling the throwing event, implicitly an event is assumed to be thrown at each KPI and goal, whenever they are calculated. BPMN 2.0 has multiple event types which are used for different purposes. To decide which event is implicitly thrown by KPIs and goals, we investigated the usage of each event type. Event types differ in the scenarios where they may be thrown and caught. Message events are used to communicate between pools. Error events handle errors in subprocesses and cannot be caught as a start event or at a gateway. Escalation events can only be caught at the border of the subprocess they're thrown in. If maximum flexibility in event handling is to be provided, these event types are unsuitable. BPMN 1.1 introduced the Signal event type to provide more flexible event communication [Sil09]. Signal events may be used to listen to external events within a process model, which is suitable for goal and KPI events generated by the CEP engine. Thus, events are to be modeled as signal events named by the KPI or goal they refer to. Using catching signal events, control actions can be modeled with standard BPMN elements, e.g. as an event subprocess or as part of an event gateway. An event subprocess is a subprocess which is started if its start event is thrown. An interrupting event subprocess stops its parent process, while a non-interrupting event subprocess runs in parallel without stopping its parent process [Sil09].

Figure 3 shows the running example extended by implementing the specified controls. Control actions are modeled as event subprocesses. For example, if the goal *avg_image_ above_zero* is violated, a signal event named *avg_img_above_zero* is thrown. It is caught in an event subprocess, as indicated by an association. In this subprocess, the image service is restarted without interrupting the main process. Similarly, three other control actions are modeled, shown as collapsed event subprocesses. If *decision_viable* is false, the event triggers an interrupting event subprocess in which an expert finishes the process manually.

Associations are used to indicate event flow within the process. During creation of the ProGoalML file these as well as the goal model are removed from the process model to create a plain BPMN model. Additionally to the process and goal model, a control model is added to the ProGoalML file indicating which goals and KPIs

may trigger control actions.

3.5 Model transformation

Using the control model, the aPro architecture has to be configured (requirement **R5**). Figure 1 shows the extended aPro architecture, new control elements marked in grey. Requirements **R1**, **R2** and **R3** are fulfilled by the CEP engine which already generates relevant events for monitoring purposes [KK]. Thus, no additional CEP rules are necessary. Below an automatically generated example CEP rule for the timing goal *notification_within_14_days*. It generates an alert event whenever a measurement of *receive* is not followed by a matching measurement of *send* within 14 days.

```
INSERT INTO ‘ALERTnotification_within_14_days‘
SELECT receive.claim_id AS claim_id FROM PATTERN [EVERY
       receive = INreceive -> (timer:interval(14 days)
       AND NOT send = INsend(claim_id = receive.claim_id))]
```

As control actions are specific to the application systems, an integrating component is necessary. Similarly to the monitoring stubs, which send unified monitoring data from a specific system, we introduce so-called *control stubs*. *Control stubs* are specific to a control action and listen to the relevant event from the CEP engine. If an event is received, the control stub performs an implementation-specific task (e.g. calling a script or sending the event to a process engine). To configure this, a listener is registered for each relevant event at the CEP engine, which is called whenever such an event occurs.

3.6 Control stubs

During deployment of the aPro architecture, the specific control stubs for each control action need to be available as Java classes. These are implemented by extending a control stub abstract class, which in turn is a listener to the CEP engine. Only a method handling the event needs to be implemented. For example regarding the timing goal *notification_within_14_days*, a control stub listening to *ALERTnotification_within_14_days* events is implemented which calls a service generating the letter whenever an alert occurs. The process instance is identified by the *claim_id* within the event.

To map control actions to control stubs, an XML format called a *control mapping* is used. For each control action it contains the implementing control stubs as well as configuration parameters. This allows custom as well as predefined control stubs. Class files need to be provided for each custom control stub. A default control mapping is created automatically, mapping each control action to an empty control stub.

4 Prototype and Evaluation

We have extended the existing prototype to model a control model as well as store it in a ProGoalML file as described in 3.4. Figure 3 has been created using the

Oryx-based [1] modeling component of the prototype. During deployment, the control mapping as well as all control stubs are added to a Java web archive (war) file containing the monitoring infrastructure.

The open source CEP engine Esper [2] is used to generate events. We adapted existing aPro functionality for sending e-mail alerts to implement control stubs. The abstract control stub implements the Esper *UpdateListener* interface. During startup of Esper, the *control mapping* XML file is read. Each control stub is instantiated as a Java object with the specified parameters and registered as a listener to the specified event at the CEP engine.

We implemented predefined control stubs allowing for invoking an executable (e.g. a batch script), calling a URL and sending an event to the process engine Activiti [3]. These can be selected and configured in the *control mapping* without writing custom source code. Therefore, requirement **R7** is fulfilled. However, we encountered an issue regarding events from aggregated goals and KPIs in Activiti. As no process instance ID is known, the only option is to broadcast these events to all process instances. This may lead to undesired behavior, as control actions may be executed multiple times. Thus, requirement **R6** is only fulfilled for nonaggregated goals and KPIs.

We evaluated the prototype using the example process implemented with a test driver as well as deployed within Activiti. Using the test driver, control actions were invoked as desired. However, using Activiti we learned that event subprocesses are currently only partially supported. We were able to test the general mechanism of sending signals to Activiti but will need to further test when development of Activiti has progressed.

In future work we plan to evaluate the prototype further using the real-life claim management process. While this process is currently monitored non-intrusively by aPro, implementing controls is a critical alteration and requires further testing.

5 Conclusion and Outlook

In this paper we described a model-driven approach for business process monitoring and execution control in heterogeneous, distributed environments. The existing monitoring infrastructure is used to provide events which trigger control actions. We described how control stubs can be used to define these control actions with minimal effort. We implemented these concepts in a prototype and evaluated it with an example process containing business and compliance controls.

Monitoring and controlling compliance at runtime is only one aspect of guaranteeing compliance of a business process. In future work in the C.om.B. project[4] we plan to unify compliance management in a generic descriptor encompassing process modeling, deployment, runtime and control [KKR+13].

[1] http://bpt.hpi.uni-potsdam.de/Oryx/WebHome
[2] http://esper.codehaus.org/
[3] http://www.activiti.org/userguide/#bpmnSignalEventDefinition
[4] http://www.iaas.uni-stuttgart.de/forschung/projects/COMB/

In future work we plan to extend the prototype and evaluate it in a real-life environment. Planned extensions include full process engine support and easier configuration management. While deploying aPro is simple, advanced options like e-mail alerting, passwords, visualization preferences and control mappings need to be specified with each deployment. We plan to unify these options in a deployment descriptor, further increasing ease-of-use. We also plan to extend the generation of monitoring dashboards to the generation of control panels, so as to provide a GUI for manual controlling tasks, and for overseeing automatic ones.

Acknowledgement

The work published in this article was partially funded by the Co.M.B. project of the Deutsche Forschungsgemeinschaft (DFG) under the promotional reference SP 448/27-1.

References

[BG05] Luciano Baresi and Sam Guinea. Towards dynamic monitoring of ws-bpel processes. In *Service-Oriented Computing - ICSOC 2005*, volume 3826 of *Lecture Notes in Computer Science*, pages 269–282. Springer Berlin / Heidelberg, 2005.

[BTPT06] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *International Conference on Web Services ICWS '06.*, pages 63 –71, 2006.

[Ger12] German Insurance Association (GDV). Verhaltensregeln fuer den Umgang mit personenbezogenen Daten durch die deutsche Versicherungswirtschaft, 2012.

[GG03] D. Goldenson and D.L. Gibson. Demonstrating the impact and benefits of CMMI: an update and preliminary results. Technical report, 2003.

[JMM12] Christian Janiesch, Martin Matzner, and Oliver Mueller. Beyond process monitoring: a proof-of-concept of event-driven business activity management. *Business Process Management Journal*, 18(4):625–643, 2012.

[Kin12] Maximilien Kintz. A semantic dashboard description language for a process-oriented dashboard design methodology. In *Proceedings of MODIQUITOUS 2012*, Copenhagen, Denmark, 2012.

[KK] Falko Koetter and Monika Kochanowski. A Model-Driven Approach for Event-Based Business Process Monitoring. *Information Systems and e-Business Management (to appear).*

[KK12] Falko Koetter and Monika Kochanowski. Goal-oriented model-driven business process monitoring using progoalml. In *BIS*, volume 117 of *Lecture Notes in Business Information Processing*, pages 72–83. Springer, 2012.

[KK13] Falko Koetter and Monika Kochanowski. A model-driven approach for event-based business process monitoring. In *Business Process Management Workshops SE - 41*, volume 132 of *Lecture Notes in Business Information Processing*, pages 378–389. Springer, 2013.

[KKR+13] Falko Koetter, Monika Kochanowski, Thomas Renner, Christoph Fehling, and Frank Leymann. Unifying Compliance Management in Adaptive Environments through Variability Descriptors (Short Paper). In *Proceed. of SOCA 2013*, 2013.

[KWR12] Falko Koetter, Anette Weisbecker, and Thomas Renner. Business process optimization in cross-company service networks - architecture and maturity model. In *Proceedings of the 2012 Annual SRII Global Conference*, 2012.

[LPF+08] Volkmar Lotz, Emmanuel Pigout, Peter M Fischer, Donald Kossmann, Fabio Massacci, and Alexander Pretschner. Towards systematic achievement of compliance in service-oriented architectures: The MASTER approach. *Wirtschaftsinformatik*, 50(5):383–391, 2008.

[MRD08] Oliver Moser, Florian Rosenberg, and Schahram Dustdar. VieDAME-flexible and robust BPEL processes through monitoring and adaptation. In *Companion of the 30th international conference on Software engineering*, pages 917–918. ACM, 2008.

[MZD09] E. Mulo, U. Zdun, and S. Dustdar. Monitoring web service event trails for business compliance. In *Service-Oriented Computing and Applications*, pages 1–8, Jan 2009.

[Obj09] Object Management Group (OMG). Business Process Model and Notation (BPMN) Version 2.0, 2009.

[PCBV11] Susanne Patig, Vanessa Casanova-Brito, and Barbara Vogeli. IT Requirements of Business Process Management in Practice âĂŞ An Empirical Study. In *Business Process Management*, volume 6336 of *LNICS*, pages 13–28. Springer, 2011.

[RdBH04] Michael Rosemann, Tonia de Bruin, and Tapio Hueffner. A model for business process management maturity. *ACIS 2004 Proceedings*, page 6, 2004.

[Sil09] Bruce Silver. *BPMN Method and Style: A levels-based methodology for BPM process modeling and improvement using BPMN 2.0.* Cody-Cassidy Press, 2009.

[TZ11] Simon Tragatschnig and Uwe Zdun. Runtime Process Adaptation for BPEL Process Execution Engines. *2012 IEEE 16th International Enterprise Distributed Object Computing Conference Workshops*, 0:155–163, 2011.

[UGSC11] Susan D Urban, Le Gao, Rajiv Shrestha, and Andrew Courter. The dynamics of process modeling: new directions for the use of events and rules in service-oriented computing. In *The evolution of conceptual modeling*, pages 205–224. Springer, 2011.

[Wes12] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition.* Springer, 2012.

[WSL09] B. Wetzstein, S. Strauch, and F. Leymann. Measuring Performance Metrics of WS-BPEL Service Compositions. In *Fifth International Conference on Networking and Services (ICNS 09)*, pages 49 –56, 2009.