Chapter 5
Model-aware Monitoring of SOAs for Compliance

Ta'id Holmes, Emmanuel Mulo, Uwe Zdun, and Schahram Dustdar

Abstract Business processes today are supported by process-driven service oriented architectures. Due to the increasing importance of compliance of an organization with regulatory requirements and internal policies, there is a need for appropriate techniques to monitor organizational information systems as they execute business processes. Event-based monitoring of processes is one of the ways to provide runtime process-state information. This type of monitoring, however, has limitations mostly related to the type and amount of information available in events and process engines. We propose a novel approach – model-aware monitoring of business processes – to address these limitations. Emitted events contain unique identifiers of models that can be retrieved dynamically during runtime from a model-aware repository and service environment (MORSE). The size of the events is kept small and patterns of events that signify interesting occurrences are identified through complex event processing and are signaled to interesting components such as a business intelligence. To illustrate our approach we present an industry case study where we have applied this generic infrastructure for the compliance monitoring of business processes.

5.1 Introduction

Business compliance, i.e., the conformance of an organization's business activities and practices with existing laws (cf. [16, 19, 34, 42]), regulations (cf. [4, 26, 27]) and its own internal policies, is a major concern of today's business community. However, these compliance concerns frequently change, making it hard to systematically and quickly accommodate new compliance requirements. The COMPAS project [15] aims to design and implement novel models, languages, and an architec-

Ta'id Holmes · Emmanuel Mulo · Uwe Zdun · Schahram Dustdar
Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Vienna, Austria, e-mail: `\{tholmes,e.mulo,zdun,dustdar\}@infosys.tuwien.ac.at`

tural framework to ensure dynamic and on-going compliance of software services to business regulations and stated user service-requirements. In this chapter we present part of the results from this project related to runtime monitoring of compliance in process-oriented systems.

Business processes are today supported by process-driven service oriented architectures (SOA). A business process comprises a collection of related, structured activities within or across organizations, that produce a specific service or product for a particular customer. Process-driven SOAs aim to increase productivity, efficiency, and flexibility of an organization, by aligning high-level business processes with applications supported by information technology. Such architectures constitute a process (or workflow) engine that orchestrates services to realize activities in a business process [24]. In an enterprise scale process-driven SOA, moreover, there exist multiple business processes and process instances that are interacting with different external entities at runtime (e.g., services, databases).

While business processes are primarily aimed at creating an output for a specified consumer, one of the other objectives that they realize is providing an auditable asset with which an organization can demonstrate its fulfillment of compliance regulations [23]. Monitoring process-driven SOAs (process instances) at runtime enables a diagnosis of process states, and therefore, provides the necessary information regarding the fulfillment of compliance requirements in the business processes. Please note that in the context of this work such compliance requirements are specified in terms of models (cf. [6]). Also the processes are generated from models. Thus, for the development of process-driven SOAs with compliance concerns model-driven engineering is used.

A typical monitoring solution consists of an external component to which events are sent – these events are recorded in audit logs (files) that can later be analyzed to identify anomalies in system behavior [29, 30, 33, 35]. Monitoring solutions that leverage complex event processing (CEP) techniques, perform an online analysis of streams of events as they happen [21, 31, 38]. These CEP-based solutions are able to deduce high-level events, i.e., events that represent certain semantics in a special domain, through analyzing patterns and properties of the events emitted by monitored components.

In the context of monitoring business processes with such event-based monitoring solutions, some limitations need to be addressed:

- When designing such solutions, it is hard to foresee all kinds of monitoring information needed during process execution. An event captures the state of the process execution at a *point* in time, whereas for monitoring purposes we are interested in the *entire* process, i.e., a wider perspective. Moreover, due to changes of requirements, the monitoring components need to consider additional information that is not transmitted with the events.

- Event-based monitoring solutions usually receive very large amounts of events. Due to the amounts of system resources consumed with this type of monitoring,

it is usually only feasible to process a limited number of events together. Moreover, it is often not feasible to embed large amounts of data in an event message, such as a model together with all its related models.

We propose a novel approach, model-aware, event-based monitoring of business processes at runtime. The monitoring is *model-aware* in the sense, that it can access and reflect on process models at runtime. High-level events (that correspond to business events), containing references to the process models, are recognized from low-level process events using complex event processing techniques. The model references enable runtime retrieval and reflection on the original process models. As a consequence, the size of the events is kept small, and (new) models and model elements can be considered during monitoring. We apply our approach in the context of monitoring for compliance of business processes. We demonstrate this through a mobile number portability case study.

The rest of this paper is structured as follows: Section 5.2 presents a motivating scenario to highlight the issues that arise during monitoring for compliance and how our approach aims to address them. Section 5.3 gives an overview of our approach and presents the details on the architecture we use to realize this approach. Section 5.4 presents an industry case study to evaluate our approach, Section 5.5, discussions, Section 5.6, related work, and finally, conclusions in Section 11.5.

5.2 Motivating Scenario

In this section, we present a scenario to illustrate issues that arise while monitoring for compliance in an organization's business processes. The scenario is based on *mobile number portability* (MNP) in mobile telecommunication companies.

Mobile telecommunication companies, usually referred to as Cell Phone Operators (CPOs), provide voice and data services to their subscribers. A CPO may operate in a single country or have branches in several countries. Inhouse services are offered by the CPO to manage all the subscribers and their stipulated contracts. Moreover each subscriber is usually able to view private information regarding their contract, or public information regarding services offered by the CPO. Not all CPOs have their own telecom infrastructure – some of them rent the network services from big telecom companies and provide a service to subscribers. Such a CPO is referred to as a Virtual Operator (VOs).

In an MNP scenario, a subscriber has the right to keep their mobile telephone number when switching between CPOs [9]. According to various National and European Union (EU) regulations (cf. [18]), MNP is one of the mandatory services a CPO must provide to its subscribers. The MNP procedure is regulated with the aim of allowing the subscribers to freely select the best CPO according to their requirements, without having to change their contact number. Therefore, one of the primary compliance concerns for a CPO as they are implementing MNP procedures is to satisfy national and EU rules and regulations.

There are essentially two steps in the MNP process. The subscriber wishing to port his number contacts the so-called recipient network, i.e., the CPO who shall be their new provider. The recipient network then executes the porting. This step of executing the porting involves a number of sub-activities, including contacting the donor network (i.e., the subscriber's current CPO), performing the porting, possibly charging the subscriber, and making a payment to the donor network. A number of issues are regulated in this process. Table 5.1 shows examples of MNP regulations in Austria [9].

Table 5.1: MNP regulations in austria [9]

Compliance/Regulatory Issue	Example of Implementation
Porting Charges	Donor network allowed to charge maximum €19 porting fees. Recipient network allowed to charge subscriber €4 – €15.
Speed of Porting	Porting should take maximum 3 days.
Porting Initiator	Porting initiated by recipient network.

In order to monitor compliance in this scenario, an event-based monitoring solution would transmit a number of events that together reflect occurrence of activities like `Request Number Porting` and `Execute Number Porting`. These events may, additionally, contain information to identify details like the subscriber number, the subscriber's geographical location, and the donor network. In this scenario we can illustrate some of the limitations we attempt to address in this work.

- We are not able to embed all information in individual events, for example, which compliance regulation may have been violated. In our scenario, a compliance violation may occur due to the `Execute Number Porting` activity, whereby the porting of the number took longer than the three (3) days permitted by regulations. This activity is detected by correlating a number of low-level events through CEP techniques. Even then, the complex event processing engine only correlates the events. We would need another source for more detailed information concerning the compliance regulations.

- Such business processes are considered long-running and may take many hours or days for completion. For event-based monitoring solutions, maintaining a history of the entire process execution state for *every* process instance could consume a large amount of resources. Therefore, the monitoring of events would be performed within limited time windows to save on resources. In this case, we would need an external source to deduce information concerning the entire process.

- Some activities within the business processes, for example `Charge Customer`, may occur in other business processes and contexts. When we present moni-

toring information, however, we would like to know in exactly which of the
processes and context this violation occurred.

With our proposed solution, we address some of the limitations stated here. More
details of this scenario and how we apply our approach are given in the case study
section (Section 5.4). First, we give details of the approach in Section 5.3.

5.3 Model-Aware Event-Based Monitoring

In this section, we present our approach for model-aware, runtime monitoring of
business processes in a process-driven SOA. It is based on monitoring and process-
ing of events, coupled with runtime access to business process models and annotat-
ing models. We first present an overview of the approach and then elaborate on this
overview in subsequent sections.

5.3.1 Approach Overview

Before we present some parts of the architecture in more detail, we first provide
an overview of our approach. For model-aware monitoring, we propose our Model-
Aware Repository and Service Environment (MORSE) [25] and an event monitoring
and processing infrastructure. Our approach comprises the following steps:

- For the design and development of processes we apply *model-driven develop-
  ment* (MDD). We use a view-based modeling framework [43] to design process
  models and generate WS-BPEL [37] (BPEL) code. We propose to store these
  process models in a model repository, and require that each model and model
  element is uniquely identifiable.
- During code generation we embed *traceability information* into the BPEL pro-
  cesses for relating the code with original models that we make identifiable by
  unique identifiers, i.e., UUIDs [28].
- The BPEL process, instrumented with traceability information, contains a
  BPEL extension for transmitting *low-level events*, e.g., for process invocation,
  containing traceability information.
- The low-level events are processed by a complex event processing engine. *Busi-
  ness events* are recognized and raised.
- An interested component consumes the business events and provides for *adap-
  tation*, *compensation*, or *synchronization*.

We illustrate our approach in Figure 5.1. Business processes, represented as pro-
cess models, are at the center of our approach. We propose to store these process
models (1) in a model repository that can be queried (7a and 8a). Each process
model and element in the repository is identifiable by a Universally Unique Identi-
fier (UUID) [28]. The model repository manages and versionizes models and model

instances. Additional information about related models or models in other versions can be discovered by querying the repository.
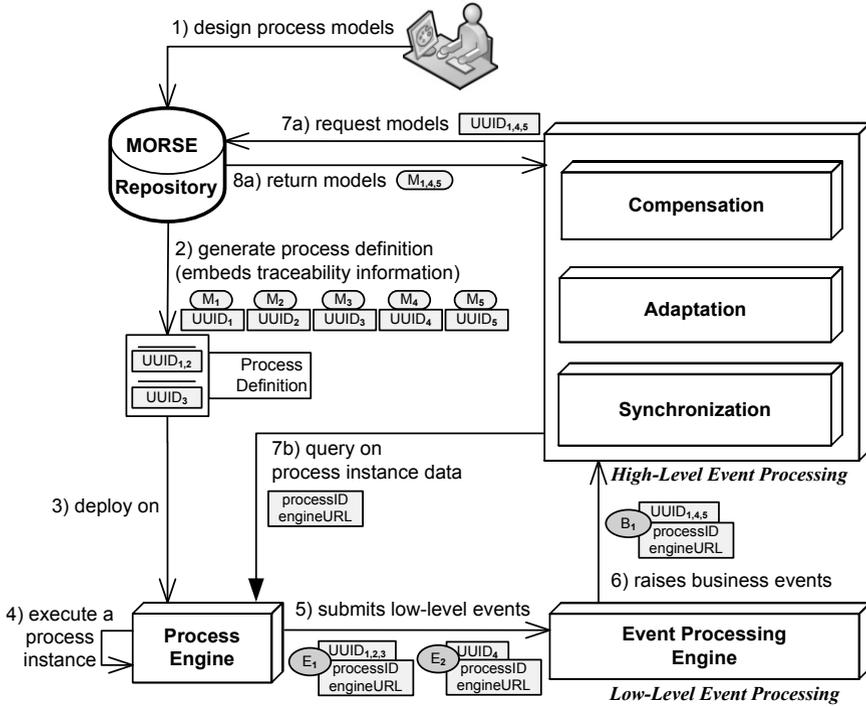
Fig. 5.1: Overview of the approach

In a generation step (2), traceability information is embedded into the process definition. That is, the process and the process elements are linked in the code via UUIDs to the models. After this model-to-code transformation an executable form of a business process, such as BPEL [37], is provided to a process engine (3). Each process execution essentially orchestrates different business activities to realize the entire process. In order to monitor a process execution (4), we monitor the progress of each process instance through events that are emitted by the process engine (5). Within these events, we embed the UUIDs of the instance's process model. Events emitted by the process engine are considered low-level events. In order to detect events at a business level, complex event processing techniques are applied (6). Business events containing, among other things, the relevant UUIDs are passed on to a component, say business intelligence (BI) component, that can perform subsequent retrieval and reflection on the process models (7a and 8a). For accessing instance data, we assume the process engine also exposes an interface for querying (7b).

In the following sections we present the details of our model-aware, event-based monitoring approach.

5.3.2 Model-Aware Repository and Service Environment

In our approach we aim at addressing the problem of monitoring and analyzing business processes to identify compliance violations through MORSE[1]. For the processes and the compliance concerns we use dedicated models. These models are used in the model-driven development (MDD) process and related to during runtime. Thus, information on a process such as stored in control-, orchestration-, or information-view models [43] and annotating compliance models is stored in and managed by MORSE that allows for the *storage* and *retrieval* of MORSE objects such as models, model elements, model instances, and other MDD artifacts. It offers read and write access to all artifacts at runtime and design time. Moreover, it stores relationships among the MDD artifacts, e.g., model-relationships such as instance, inheritance, and annotation relations. Moreover, the MORSE repository provides *versioning* capabilities not only to MDD artifacts, but also to their relationships. This way, models can be manipulated at runtime of the client system with minimal problems regarding maintenance and consistency. New versions and old versions of the models can be maintained in parallel, so that old model versions can be used until all their model instances are either deleted or migrated to the new model version.

Figure 5.2 shows the internal architecture of MORSE. The model repository exposes all its functionality as Web services which ease the integration of MORSE into service-oriented environments. The services can be consumed by various clients, e.g., design tools, administrative clients, monitoring services, or services that provide for adaptation. Thus, MORSE supports the development of models during design-time and allows for the retrieval of models at runtime.

5.3.2.1 Model-Traceability for Process-Driven SOAs

With MORSE we follow a model-driven approach. That is, we apply model-to-code transformation for the generation of process code, deployment artifacts, and monitoring directives. For this the process models and annotating models such as a compliance metadata model are processed by a transformation template. During this step we embed traceability information into the generated code so that the original model(s) can be related to during runtime. As MDD artifacts in MORSE repositories are identifiable by UUIDs, the traceability information uses these UUIDs as well. Thus, the generator automatically weaves references into the generated source code or configuration instructions, so that the corresponding models can be identified and accessed from the running system.

For the traceability of models in process-driven SOAs we propose an extension for BPEL. Figure 5.3 shows an excerpt of a BPEL process[2] with a BPEL extension

---

[1] `http://www.infosys.tuwien.ac.at/prototype/morse`

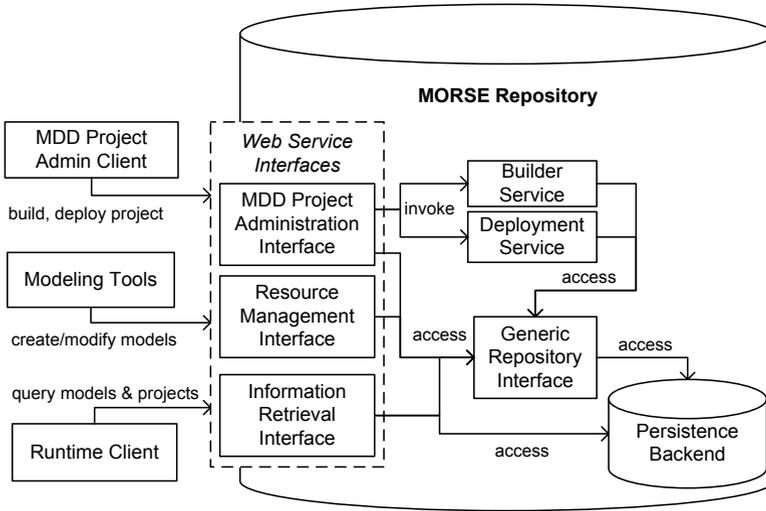[2] For simplicity reasons most XML namespaces have been omitted.

Fig. 5.2: MORSE architecture

for mapping code elements of the BPEL process to MORSE object identifiers. The `traceability` element, that indicates the UUID of the `build` as an attribute, is a sequence of `rows` that maps BPEL elements to the `uuids` of corresponding MORSE objects. The XML Path Query Language (XPath) [5] is chosen as the default query language for selecting the XML elements of the BPEL code. For extensibility, an optional `queryLanguage` attribute, that has the same semantics as in BPEL (cf. Section 8.2 of [37]), can specify an alternative query language or XPath version.

Note that this traceability information can annotate any XML based target code and can often be supplied as an inline extension[3]. It can also be supplied exogenously within a separate file. As a consequence, our approach is not limited to BPEL but can applied to other process languages as well.

5.3.3 Event Monitoring and Processing

In MORSE each model and model-element is identifiable by a UUID. These are embedded as traceability-links in the process definition as discussed in the previous section. During process execution, events are emitted by the process engine, for example, an activity starts, a database is accessed, etc. Whereas the events emitted by the process engine are considered low-level events, we are interested in business related events. Business events cannot be observed directly – rather, they are derived by observing and aggregating patterns of low-level events, through certain processing rules. Therefore, in addition to monitoring, events are processed to iden-

---

[3] Supposed that such extensibility is provided with an `any` element in the XML schema.

```
<process name="NumberPortabilityProcess">
  <extensions>
    <extension mustUnderstand="yes"
        namespace="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"/>
  </extensions>
  <import importType="http://www.w3.org/2001/XMLSchema"
      namespace="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"
      location="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"/>
  <morse:traceability build="ec46dcdc-3f81-4dec-b437-8da5269ad334">
    <row query="/process[1]"
        queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
      <uuid>65130c63-dda7-4193-9bac-fe7eda9f38b9</uuid>
      <uuid>2e3261fc-452c-4b55-8c77-fc4aaac29ddf</uuid>
      <uuid>9a7f6681-2616-4797-9510-0c5cd126b24c</uuid>
      <uuid>11194fb5-3b9f-4507-a3a0-6b3f7e8e146a</uuid>
    </row>
    <row query="/process[1]/sequence[1]/receive[1]">
      <uuid>0e4bc6b8-8f28-4e0d-a8f7-ebf99bf95b62</uuid>
    </row>
    <row query="/process[1]/sequence[1]/invoke[2]">
      <uuid>8b62c1ef-2c12-41f3-a660-dae82c6168dc</uuid>
    </row>
  </morse:traceability>
  <sequence>
    <!-- ... //-->
  </sequence>
</process>
```

Fig. 5.3: BPEL process with an extension for MORSE traceability

tify high-level events that have significance from the business perspective, e.g., a low-level event indicating database access might not be so interesting, however, a combination of events that indicate completion of a specific business activity is of more interest to business actors.

As a business process is executed, events are emitted that represent the state in the progress of the business process. In a large scale SOA, multiple instances of different business processes execute concurrently, resulting in the emission of an interwoven sequence of events. The event processing engine has a sequential view of these events as they arrive at its interface. In order to process these events, the event processing engine is configured with queries based on a number of factors like the type of event, the data contained within the event, or the expected patterns of events. Typically queries for configuring the engine are defined in an Event Processing Language (EPL). These EPL statements are similar to Structured Query Languages (SQL) for database querying. The statements instruct the engine on which events, event data, or patterns of events to search for in an event stream. For example, the query shown in the inset of Figure 5.4 enables the engine to identify a business activity D from a stream of events.

The queries enable the correlation of events and identification of a particular group of business events within a specific process instance. In Figure 5.4, we see the pattern of events 8-9-10 matches activity D of a process instance. Further filtering of
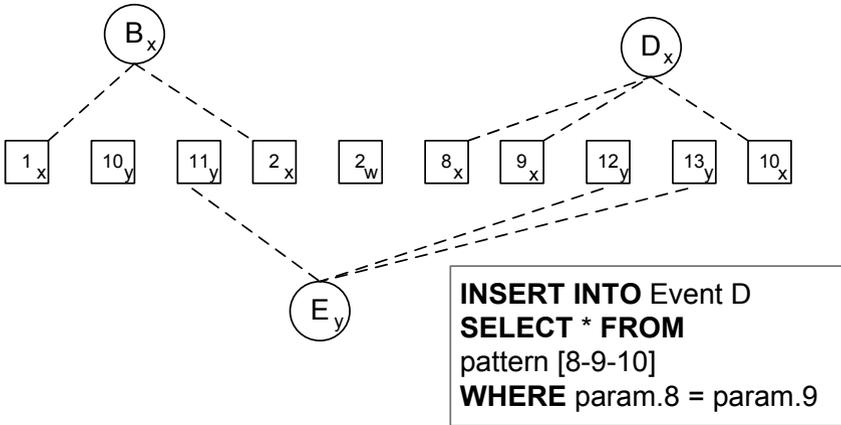
Fig. 5.4: EPL query

events is performed through comparison of their parameters. In the example, we use the subscript x as a parameter to indicate that these events belong to the same process instance. The WHERE clause performs a more fine-grained selection of events based on related parameters, for example, the process instance. Within the same stream, there are other events that belong to different process instances. Therefore, using such filtering mechanisms we are able to separate events into different business activities in their distinct process instances. Finally, these business-events are passed on to a BI component to make the necessary compliance checks and decisions on actions to take.

Combining the low-level and high-level events with runtime access to a model repository, we provide information to enable compliance detection tasks. Business events represent the execution progress of a process instance and, in addition, provide a UUID to its process model. The UUIDs enable querying of the model repository, for retrieval of additional information that is not directly accessible to the event processing engine, but is required for compliance detection.

5.4 Case Study

In this section we present details of the case study introduced in Section 5.2 that deals with monitoring for compliance in the mobile number portability (MNP) process. We demonstrate how we use our approach to achieve a model-aware, event-based monitoring solution, that checks for compliance at runtime. In the first step, we need to annotate the process models with compliance data. We do not present details concerning the MDD environment, however, it is important to note that when the executable processes are generated from the process models, UUIDs are incor-

porated into them to allow for unique identification of the models. Following this we define relevant EPLs that enable filtering of the low-level events to high-level, business events. We consider one of the compliance requirements that relates to the quality of service (QoS) regarding the MNP process, i.e., the portability needs to be performed within three (3) days (cf. Table 5.1).

5.4.1 Annotating Business Process Models with Compliance Concerns

Figure 5.5 illustrates an excerpt of a compliance model with concepts from the compliance domain. In this domain compliance experts derive `Compliance Requirements` from `Compliance Sources`. Such sources refer to national, European, or international regulations and laws or internal policies. For realizing the compliance to requirements, `Controls` are employed together with `Compliance Rules`. The latter formalize the requirements for a control in a way suitable for the BI to check the compliance of a system. Using `Controls`, `Compliance Targets` such as `Business Processes`, activities (`BP Activities`), and external `Web Services` can be annotated.

In our work we use name-based matching for such annotations. That is, model elements from different models that contain the same name are matched. Thus, a process P with the name MNP in a model M_1 can be annotated exogenously by a compliance model M_2. In this model a control C references a named element with the same name MNP. Please note, that our approach is not limited to name-based matching. Any form of annotation or direct relation from compliance controls to compliance targets is possible in order to specify compliance concerns for business processes.
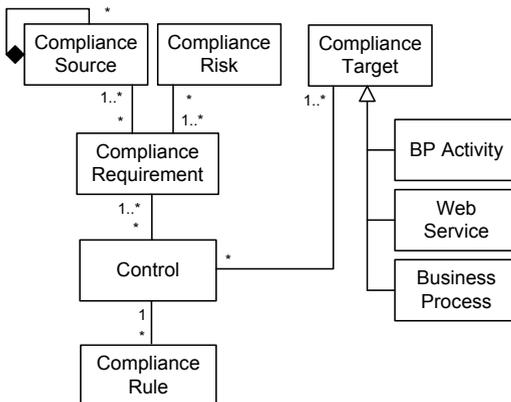
Fig. 5.5: Excerpt of a compliance model

For the generation of process definitions, instances of the process and compliance models are taken as inputs. The MORSE builder service realizes this generation step of the model-driven development process by weaving traceability information into the code. This traceability information is stored as a traceability matrix that relates code elements to the UUIDs of the model(-element)s.

The traceability matrix was realized as a BPEL extension (see also Section 5.3.2.1)[4]. We modified the Apache ODE [41] BPEL process engine so that events from the engine contain the UUIDs as specified in the BPEL extension. That is, if a BPEL event for an activity is raised by the engine it would contain the correlating UUIDs for the activity as specified in the traceability matrix.

5.4.2 Event Processing

In order to detect business events of interest, a CEP engine is configured automatically with EPL rules or statements that specify event types, event data, and patterns of events to detect at runtime. The CEP engine observes a stream of low-level events emitted by the process execution engine and deduces that a particular business event has occurred.

In our MNP process, a number of low-level events are emitted by the Apache ODE process engine in response to the execution of the different process steps. The events contain, among other things, the process engine's internal identifier of the process instance (`processID`), a URL of the process engine, and UUIDs of the models and model-elements that relate to the process and process elements. Using the low-level events, we monitor that the `Execute Number Porting` activity occurred before we can alert of the need to check for compliance. We have the simple EPL statements illustrated in Figure 5.6, that show how such an event combination can be monitored with CEP. The EPL statement emits a business event corresponding to the `Execute Number Porting` activity. This event includes the process identifier and a UUID indicating the target model to be checked in the repository.

```
INSERT INTO ExecuteNumberPortEvent(pid, sUUID, eUUID)
SELECT StartEvent.pid,
       StartEvent.UUID,
       EndEvent.UUID
FROM ActivityExecStartEvent as StartEvent,
     ActivityExecEndEvent as EndEvent
WHERE StartEvent.pid = EndEvent.pid AND
      StartEvent(id = 'executeNumberPort')
```

Fig. 5.6: Number portability EPL

---

[4] Following a model-driven approach we profit from the abstraction and platform independent models. That is, although we realized support for the generation of BPEL code, our approach is not limited to this technology but support for other process languages can be developed.

Since the UUIDs of the business process models and model elements are embedded in the events, we send them to the BI component that uses them to query the MORSE repository in order to look up more information concerning the process. We present the types of tasks that would be expected from such a BI component in the next section.

5.4.3 Compliance Checking

The compliance checks are performed by a BI component that receives business events from the CEP engine. The BI component uses the event type and UUID data in order to determine what compliance concerns have to be checked, and in case of compliance violations consequently what actions to invoke. As the `Execute Number Porting` activity executes, the process execution engine emits low-level events to the CEP engine, e.g., when a process activity is started. However, when the combination of two process engine events `ActivityExecStartEvent` and `ActivityExecEndEvent`, have occurred, the CEP engine recognizes this as a business event `Execute Number Portability`, and alerts the BI component the occurrence of this business event.

For realization of the compliance checking, we assume that the process engine exposes an API for querying process instance data, e.g., variables of a process instance. The BI component consumes the business event, looks up the compliance metadata model from the repository and requests process-instance data, i.e., the time, the activities have been performed, from the process engine. With this information, i.e., by reflecting on the compliance model and process instance data, a violation of the compliance requirement as stated in Table 5.1 can be determined. Finally, if a violation was detected, a compensation action can be initiated. For example, the Cell Phone Operator may account free credits to the customer as a compensation to the tardy number porting.

5.5 Discussion

In this section we motivate some advantages and limitations of our model-aware, event-based monitoring approach.

First of all, with the MORSE approach we expand the usage of models and propagate them from the design time to the runtime. That is, models are not only used for describing business processes and specifying compliance concerns during design time but they are related to during runtime, using automatically embedded traceability information, and used for the compliance checks. Coupled with an event-based (CEP) approach, a business intelligence profits from accessing and reflecting on the models. That is, having defined abstraction levels, the conceptual models are suitable for the required reflections.

Upon the detection of a compliance violation, the user typically wants to understand which process has caused the violation and why. With MORSE the user can access

the process model that has caused the violation. However, the process model in general is not the only relevant information or not the root cause of the violation. Other models such as a compliance model that annotates the process model might carry the answer to the violation; hence, they are accessed, too. Examples are the model specifying the compliance rule that has been violated or the models of the processes that have invoked the process leading to the violation. Finally, once the root cause has been identified, it is likely that the user will fix the affected models. Then, the corrected models should from then on be used for new instances of the business processes. Here, a transparent versioning support as realized by MORSE is crucial, as it is typically not possible to automatically migrate running process instances from one model to another. Old model versions are supported as long as instances of them are running, unless the execution of such instances is halted.

Event-based (CEP) monitoring solutions usually receive large volumes of events. Typically, these solutions do not persist long-running information about process executions. This limits the size of the window of events, over which these solutions are able to store and process data regarding process executions – processing a large window size would require lots of computing resources. In our approach, we embed traceability information, i.e. identifiers to models and model-elements, in the process events. In this way, the CEP engine is still able to identify interesting business events, and in a further step, relevant information concerning the entire process model and process instance data can be retrieved by a business intelligence (BI) component. For instance, the BI can lookup process and annotating compliance models such as compliance requirements and rules at MORSE as well as process-instance data from the process engine. Additionally, we believe that size of events is kept to a minimum when we use traceability information that allows later querying.

Because compliance rules are looked up by the BI, they may dynamically change during runtime without the need to adapt the CEP. For example, the BI may want to consider the latest requirements when determining the compliance. In other circumstances processes only need to comply to the set of requirements as effective during instantiation. Both scenarios are supported with MORSE, that realizes a transparent versioning of models.

There is a clear separation of the CEP engine, that task is to identify interesting events, and the BI, that reasons about the compliant execution. Thus, the CEP does not need to comprise the logic from the BI but its configuration is kept simple and therefore is manageable. In contrast, the BI unit focuses on determining the compliant execution. It does so by, first, retrieving all relevant information and, second, reason about it.

One of the main limitations we view with our approach at the moment, is that for some scenarios it is not readily applicable to support long-running monitoring at the process instance level – this would require access to process instance variables. We currently assume existence of a query interface on the process engine, which provides access to this information.

Our approach assumes that model-driven engineering is used for the development of the system. That is, the business process definitions (BPEL code) and the traceability matrix (BPEL extension for the MORSE traceability) are generated from models. It is possible to introduce our approach into a non-model-driven project (e.g., as a first step into model-driven development). For that, existing business process definitions would be manually extended and related to compliance models. In this case the traceability matrix would only relate elements from the process definition to compliance models but not process models. This would limit the business intelligence in case the latter models are needed as well.

Finally, our approach introduces some complexity to the system. That is, in addition to the process engine we employ a MORSE repository, a CEP engine, and a business intelligence.

5.6 Related Work

Some of the work related to integrated event-based monitoring solutions are now presented. In addition we relate to work in the field of requirements monitoring. Finally, we mention on various model repositories and compare to MORSE.

5.6.1 Related Work on Event-based Monitoring

Event-based support of process executions has been previously researched [12, 22]. Casati and Discenza [12] proposes extending workflow models with the capability to specify points at which events can be raised during the workflow execution. This is a similar idea to the events raised by the process engine in our approach, except that in our case, the engine would emit a fixed set of events (process started, process end); in their approach, the user has greater control over the types of events emitted and when an event should be emitted. Their approach also proposes an event service that can filter and correlate events to dispatch them to other workflow models. The main difference in our approach is the use of the model repository to provide extra information concerning process models. Hagen and Alonso [22] use events for direct communication between processes. A process instance exchanges an event with another process instance, to make decisions on how to proceed with the execution.

In the MOSES approach and framework [11], process models (e.g., BPEL) are fed into the system – these models are required to fulfill certain criteria. The system builds a behavioral model which is used in optimization calculations. Monitoring at runtime observes the system and the optimization calculations to decide on adaptation. The monitoring, however, is not based on events, and the process models fed into the system are from external entities, hence necessitating model verification. On the other hand the approach provides adaptation based on *per instance* variables. A similar idea to the MOSES approach is proposed by Cappiello et al. [10], where quality attributes are monitored to determine adaptation. The difference is that this approach proposes predictive adaptation.

5.6.2 Related Work on Requirements Monitoring

While particular monitoring infrastructures can be integrated with MORSE and used for the compliance checking, our work particularly focuses on relating to models, the monitored systems have been generated from. Thus, our work makes such models accessible at runtime. Note, that not only, e.g., process models but also compliance concern models are managed by MORSE. This allows for the novel and direct linkage and correlation of model-driven system and requirements models. In this section we refer and relate to work in the areas of runtime requirements-monitoring.

Feather et al. [20] discuss an architecture and a development process for monitoring system requirements at runtime. It builds on work on goal-driven requirements engineering [17] and runtime requirements monitoring [14].

Skene and Emmerich [39] apply MDD technologies for producing runtime requirements monitoring systems. This is, required behavior is modeled and code is generated for, e.g., the eventing infrastructure. Finally, a metadata repository collects system data and runs consistency checks to discover violations. While in our work we also showcase the generation of code for the eventing infrastructure (see Section 5.3.2), our approach assumes an existent monitoring infrastructure. In case of a violation the MORSE approach not only allows us to relate to requirement models but also to the models of the monitored system.

Chowdhary et al. [13] present a MDD framework and methodology for creating Business Performance Management (BPM) solutions. This is, a guideline is described for implementing complex BPM solutions using an MDD approach. Also, with inter alia the specification of BPM requirements and goals the framework provides runtime support for (generating) the eventing infrastructure, data warehouse, and dashboard. The presented approach allows for the monitoring and analysis of business processes in respect of their performance. Thus, similarly to our approach, compliance concerns such as quality of service concerns as found in service level agreements can be specified and monitored by the framework. Besides the monitoring of business processes and service-based systems in general, our approach particularly focuses on also relating to conceptual models of the systems from the runtime, not only their requirements. As a consequence, the system and end-users can directly relate to the MDD artifacts of a system in case of a violation. This allows for the subsequent reflection, adaptation, and evolution of the system. In contrast, the BPM solution supports compensation, i.e., the execution of business actions according to a decision map.

Another model-based design for the runtime monitoring of quality of service aspects is presented by Ahluwalia et al. [1]. Particularly, an interaction domain model and an infrastructure for the monitoring of deadlines are illustrated. In this approach, system functions are abstracted from interacting components. While a model-driven approach is applied for code generation, the presented model of system services is only related to these in a sense that it reflects them. This is, it is not a source model for the model-driven development of the services. In contrast, MORSE manages and

is aware of the real models, systems are generated from. This allows for root cause analysis and evolution of as demonstrated in the presented case study.

5.6.3 Related Work on Model Repositories

Besides the monitoring of runtime requirements in form of compliance concern models, the MORSE approach particularly focuses on the management of models of service-based systems and their accessibility during runtime. Particularly, it targets at integration with services and – as presented in this work – facilitates model-aware monitoring. For this reason, a model repository with versioning capabilities is deployed (see Section 5.3.2). It abstracts from modeling technologies and its UUID-based implementation allows for a straightforward identification of models and model elements.

Other model repositories primarily aim at model-based tool integration. Model-Bus [40], e.g., addresses the heterogeneity and distribution of model tools and realizes transparent model update. Designed as an open environment, ModelBus focuses on integrating functionality such as model verification, transformation, or testing into a service bus.

Odyssey-VCS 2 [36] is an EMF based model repository after initially relying on the NetBeans Metadata Repository [32]. Odyssey-VCS 2 [36] and AMOR [2, 7] particularly have a focus on the versioning aspect of model management (see also [3]), e.g., for the conflict resolution in collaborative development (cf. [8]).

These works mainly focus on the design time. MORSE, in contrast, focuses on runtime services and processes and their integration, e.g., through monitoring, with the repository and builds on the simple identification for making models accessible at runtime. Instead of aiming at reconciling a multitude of modeling tools' languages and the integration of arbitrary (legacy) tools, MORSE also concentrates on some selective concepts such as relations between models. The MORSE repository abstracts from technologies, focuses on MDD projects, and targets at integration with services.

5.7 Conclusion

We propose a model-aware approach for runtime monitoring of business compliance in process-driven SOAs. Our approach leverages a model repository and event-based monitoring. Business process models are stored in a model repository that can be queried. The process models are uniquely identifiable, and relations between process models can also be discovered. During execution a process engine emits low-level events, embedded with a reference to the process models. These low-level events are correlated into high-level business events that trigger compliance checking actions. Any additional information required for compliance checks is retrievable through the query interface of our MORSE repository.

With the model-aware monitoring for SOAs we have presented a novel and generic approach of how to relate system models and system requirements models. In the context of compliance monitoring for example and by relating processes and process activities to compliance models a business intelligence has reacher means of analyzing the runtime process execution as dynamic reflection on the models and related models becomes possible.

Our approach supports compliance checking by combining the power of CEP techniques, which work best in a limited processing window, and the model querying capabilities from a model repository to provide a wider context of information from the models. At the moment, for context information regarding a process model, we provide MORSE. However, at the process instance level, we assume the possibility to query an event processing engine. Our future work shall look into how such a querying interface can be provided for an engine, to manage queries at process instance level.

References

1. Ahluwalia, J., Krüger, I.H., Phillips, W., Meisinger, M.: Model-based run-time monitoring of end-to-end deadlines. In: W. Wolf (ed.) EMSOFT, pp. 100–109. ACM (2005)
2. Altmanninger, K., Kappel, G., Kusel, A., Retschitzegger, W., Seidl, M., Schwinger, W., Wimmer, M.: AMOR – towards adaptable model versioning. In: 1st International Workshop on Model Co-Evolution and Consistency Management, in conjunction with MODELS '08 (2008)
3. Altmanninger, K., Seidl, M., Wimmer, M.: A survey on model versioning approaches. IJWIS **5**(3), 271–304 (2009)
4. Bank for International Settlements: Basel II: International Convergence of Capital Measurement and Capital Standards: A Revised Framework - Comprehensive Version. `http://www.bis.org/publ/bcbsca.htm` (2006). [accessed in June 2010]
5. Berglund, A., Boag, S., Chamberlin, D., Fernández, M.F., Kay, M., Robie, J., Siméon, J.: XML path language (XPath) 2.0. W3C recommendation, W3C (2007). [accessed in July 2009]
6. Bézivin, J.: On the unification power of models. Software and System Modeling **4**(2), 171–188 (2005)
7. Brosch, P., Langer, P., Seidl, M., Wimmer, M.: Towards end-user adaptable model versioning: The by-example operation recorder. In: CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models, pp. 55–60. IEEE Computer Society, Washington, DC, USA (2009). DOI http://dx.doi.org/10.1109/CVSM.2009.5071723
8. Brosch, P., Seidl, M., Wieland, K., Wimmer, M., Langer, P.: We can work it out: Collaborative conflict resolution in model versioning. In: ECSCW 2009: Proceedings of the 11th European Conference on Computer Supported Cooperative Work, pp. 207–214. Springer (2009). URL `http://dx.doi.org/10.1007/978-1-84882-854-4_12`
9. Buehler, S., Dewenter, R., Haucap, J.: Mobile number portability in europe. Telecommunications Policy **30**(7), 385 – 399 (2006). DOI DOI:10.1016/j.telpol.2006.04.001. URL `http://www.sciencedirect.com/science/article/B6VCC-4K5JBY0-1/2/e83f338b89f16a55cb0fb8d852473840`. Mobile Futures

10. Cappiello, C., Kritikos, K., Metzger, A., Parking, M., Pernici, B., Plebani, P., Treiber, M.: A quality model for service monitoring and adaptation. In: First Workshop on Monitoring, Adaptation and Beyond in conjunction with ICSOC-ServiceWave Conference, pp. 183–195 (2008)

11. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F., Mirandola, R.: Qos-driven run-time adaptation of service oriented architectures. In: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering on European software engineering conference and foundations of software engineering symposium, pp. 131–140. ACM (2009). DOI http://doi.acm.org/10.1145/1595696.1595718

12. Casati, F., Discenza, A.: Supporting workflow cooperation within and across organizations. In: Proceedings of the 2000 ACM symposium on Applied computing, pp. 196–202. ACM (2000). DOI http://doi.acm.org/10.1145/335603.335742

13. Chowdhary, P., Bhaskaran, K., Caswell, N.S., Chang, H., Chao, T., Chen, S.K., Dikun, M.J., Lei, H., Jeng, J.J., Kapoor, S., Lang, C.A., Mihaila, G.A., Stanoi, I., Zeng, L.: Model driven development for business performance management. IBM Systems Journal **45**(3), 587–606 (2006)

14. Cohen, D., Feather, M.S., Narayanaswamy, K., Fickas, S.S.: Automatic monitoring of software requirements. In: ICSE '97: Proceedings of the 19th international conference on Software engineering, pp. 602–603. ACM, New York, NY, USA (1997). DOI http://doi.acm.org/10.1145/253228.253493

15. COMPAS Consortium: Compliance-driven Models, Languages, and Architectures for Services. `http://compas-ict.eu` (2007). [accessed in June 2010]

16. Congress of the United States: Public Company Accounting Reform and Investor Protection Act (Sarbanes-Oxley Act), Pub.L. 107-204, 116 Stat. 745. `http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/content-detail.html` (2002). [accessed in June 2010]

17. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Program. **20**(1-2), 3–50 (1993)

18. European Parliament and Council: Directive 2002/22/EC of the European Parliament and of the Council of 7 March 2002 on universal service and users' rights relating to electronic communications networks and services (Universal Service Directive). `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0022:EN:NOT` (2002). [accessed in June 2010]

19. European Parliament and Council: Directive 2004/39/EC on markets in financial instruments. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:02004L0039-20060428:EN:NOT` (2004). [accessed in June 2010]

20. Feather, M., Fickas, S., van Lamsweerde, A., Ponsard, C.: Reconciling system requirements and runtime behavior. In: Software Specification and Design, 1998. Proceedings. Ninth International Workshop on, pp. 50–59 (1998). DOI 10.1109/IWSSD.1998.667919

21. Greiner, T., Düster, W., Pouatcha, F., von Ammon, R., Brandl, H.M., Guschakowski, D.: Business activity monitoring of norisbank taking the example of the application easycredit and the future adoption of complex event processing (CEP). In: Proceedings of the 4th international symposium on Principles and practice of programming in Java, pp. 237–242. ACM (2006)

22. Hagen, C., Alonso, G.: Beyond the black box: event-based inter-process communication in process support systems. In: 19th IEEE International Conference on Distributed Computing Systems, pp. 450–457 (1999). DOI 10.1109/ICDCS.1999.776547

23. Havey, M.: Essential Business Process Modeling. O'Reilly Media, Inc. (2005)

24. Hentrich, C., Zdun, U.: Patterns for process-oriented integration in service-oriented architectures. In: Proceedings of 11th European Conference on Pattern Languages of Programs (2006)

25. Holmes, T., Zdun, U., Dustdar, S.: MORSE: A Model-Aware Service Environment. In: M. Kirchberg, P.C.K. Hung, B. Carminati, C.H. Chi, R. Kanagasabai, E.D. Valle, K.C. Lan, L.J. Chen (eds.) Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference (APSCC), pp. 470–477. IEEE (2009). DOI 10.1109/APSCC.2009.5394083

26. Information Systems Audit and Control Association: Control Objectives for Information and Related Technology (CobiT). `http://www.isaca.org/cobit` (1996). [accessed in June 2010]

27. International Accounting Standards Committee (IASC) Foundation: International Financial Reporting Standards. `http://www.iasb.org/IFRSs/IFRS.htm`. [accessed in June 2010]

28. International Telecommunication Union: ISO/IEC 9834-8 Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components (2004)

29. Kang, J.G., Han, K.H.: A business activity monitoring system supporting real-time business performance management. In: Third International Conference on Convergence and Hybrid Information Technology, vol. 1, pp. 473–478 (2008)

30. Kung, P., Hagen, C., Rodel, M., Seifert, S.: Business process monitoring & measurement in a large bank: challenges and selected approaches. In: Sixteenth International Workshop on Database and Expert Systems Applications, pp. 955–961 (2005)

31. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2002)

32. Matula, M.: NetBeans metadata repository. `http://mdr.netbeans.org`. [accessed in July 2009]

33. McGregor, C., Kumaran, S.: Business process monitoring using web services in B2B e-commerce. In: Parallel and Distributed Processing Symposium., Proceedings International, pp. 219–226 (2002)

34. Ministre de l'économie, des finances et de l'industrie: loi de sécurité financière. `http://www.senat.fr/leg/pjl02-166.html` (2003). [accessed in June 2010]

35. zur Muehlen, M., Rosemann, M.: Workflow-based process monitoring and controlling-technical and organizational issues. In: Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, p. 10 pp. vol.2 (2000)

36. Murta, L., Corrêa, C., Jo a.G.P., Werner, C.: Towards Odyssey-VCS 2: Improvements over a UML-based version control system. In: CVSM '08: Proceedings of the 2008 international workshop on Comparison and versioning of software models, pp. 25–30. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1370152.1370159

37. Organization for the Advancement of Structured Information Standards: Web service business process execution language version 2.0. OASIS Standard, OASIS Web Services Business Process Execution Language (WSBPEL) TC (2007). [accessed in February 2010]

38. Rozsnyai, S., Vecera, R., Schiefer, J., Schatten, A.: Event cloud - searching for correlated business events. In: The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, pp. 409–420 (2007)

39. Skene, J., Emmerich, W.: Engineering runtime requirements-monitoring systems using mda technologies. In: R.D. Nicola, D. Sangiorgi (eds.) TGC, *Lecture Notes in Computer Science*, vol. 3705, pp. 319–333. Springer (2005)

40. Sriplakich, P., Blanc, X., Gervais, M.P.: Supporting transparent model update in distributed case tool integration. In: H. Haddad (ed.) SAC, pp. 1759–1766. ACM (2006)

41. The Apache Software Foundation: Apache ODE (Orchestration Director Engine). `http://ode.apache.org`. [accessed in June 2010]

42. The Netherlands Corporate Governance Committee: The Dutch corporate governance code. `http://www.commissiecorporategovernance.nl/page/downloads/CODEDEFENGELSCOMPLEETII.pdf` (2003). [accessed in June 2010]

43. Tran, H., Zdun, U., Dustdar, S.: View-based and model-driven approach for reducing the development complexity in process-driven SOA. In: Intl. Working Conf. on Business Process and Services Computing (BPSC'07), *Lecture Notes in Informatics*, vol. 116, pp. 105–124 (2007)