Towards Instant Monitoring of Business Process Compliance

Ahmed Awad Emilian Pascalau
Mathias Weske
Hasso Plattner Institute, University of Potsdam, Germany
{ahmed.awad, emilian.pascalau, mathias.weske}@hpi.uni-potsdam.de

Abstract:
Business process models are becoming the backbone of any enterprise system, the blue prints that drive business. However, there are many situations in today's business when such processes have to be validated against compliance requirements. Albeit compliance and business experts are able to define and relate compliance requirements, constraints to tasks within business processes at design time (usually using an informal notation), compliance enforcement has to be done at enactment time. To do so, processes have to be monitored at execution time. We are introducing a compliance monitoring framework, that tackles the problem of business process compliance by addressing the common concepts from the two mentioned perspective. Moreover, we address the resource and data perspective as well as control flow perspective for monitoring. Enforcing compliance on business processes is strongly related to complex event processing as events are the way through which the state of a process enactment is perceived.

Keywords:
Business activity monitoring, Complex Event-driven process controlling, Instant monitoring, ECA Rules

1 Introduction

Business activity monitoring (BAM) identifies the need for *instant* monitoring and reporting about significant events during a process execution. Usually, BAM solutions provide a real time information about key performance indicators (KPI) for process executions, e.g., duration of an activity, bottlenecks, waiting time within a performer's task list. An activity monitoring software listens to events emitted by the process execution engine to update a dashboard with the new information. Thus, the ability to instantly report about what occurs within a process instance is bounded by the granularity of events emitted by the process execution engine. Moreover, activity monitoring is limited to informing about what happens within the process execution without the ability to enforce specific process instances to suspend if something goes wrong.

Business process models are the blue prints that drive the business. Implementing the processes in a company's IT infrastructure implies the existence of several entities: humans, enterprise systems, process engines, inference engines and such like.

In today's business, there are many situations that require instant monitoring, yet, with different responses rather than reporting about the occurrence of an event. For instance, the maintenance of resource allocation constraints, e.g., separation of duty, requires the monitoring of task delegations among human performers to prevent fraud, especially if not all allocation possibilities are know before hand. In case of violation, a monitoring agent may block the execution of the process instance according to a prescribed policy. Another example for the need to instant monitoring is the follow up of activity execution deadline. In such case, the monitoring agent may send reminding messages to the performers about the approaching of a deadline. In these previous two examples, finer-grain events, other than activity start and activity termination events, must be emitted by the process execution in order to enable such monitoring situations.

Usually workflow engines use a process to guide any involved entities towards process fulfillment. However, there are also other situations when processes are mainly enacted by humans throughout an entire IT infrastructure. In this second case the guidance based on processes is much more difficult as, e.g., order of activities has to be imposed somehow in addition to compliance checking. Nevertheless, all these have to be done at runtime, when processes are instantiated and enacted in a concrete running context that comprises all the participants, required services and IT systems, as well as any required data objects.

In this paper, we aim at establishing an event-based framework for monitoring process compliance. Within this framework we link between constraints, compliance requirements, that are put on the business process at design time and the type of events to be captured at process execution, in order to assess the compliance of the execution with these constraints. The monitoring framework itself sits between the business process models, any compliance requirements that might have been associated with the process models and the IT infrastructure that enacts the processes. In such a way both the blue print processes and the actual running instances are connected to each other, and they are checked in the same time for any compliance requirements.

In Section 2, we discuss in more details the need for instant monitoring of compliance through a use case. The compliance monitoring framework is discussed in Section 3. Related work is discussed in Section 4 before we conclude the paper in Section 5.

2 Use Case

Organizations establish policies to enforce compliance with different types of regulations. The objectives of these regulations center around preventing fraud within financial transactions. To be compliant with these regulations, a first step is to explicitly model processes to be compliant. However, process execution must be monitored to take appropriate actions against compliance violations.

The separation of duty (SOD) principle is one of the most fundamental constraints to prevent fraud. Simply, SOD necessitates that the requester of a financially related service must not be the same person who approves or grants that request. For instance, a purchase request must be approved by a different person other than the one who issued it.

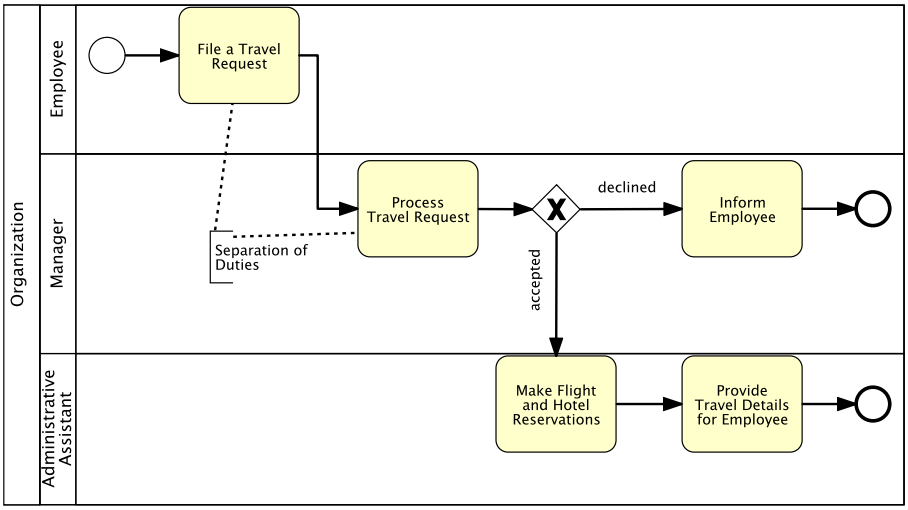

Figure 1: A process to handle travel requests

Although it is possible to explicitly model that [WS07, MPS08] and verify it at process design [WMM09], it is not sufficient to guarantee a compliant execution. The need for instant monitoring might be required for many reasons. One reason is that the process execution environment allows delegation between human performers of tasks. Yet, the process modeling language cannot impose constraints on this delegation. In this case, instant monitoring is needed to fill the gap between the process design and its behavior at runtime. Another reason is the nature of the execution environment itself. There might be no direct support through an execution engine. However, individual steps are supported by IT-systems in the enterprise.

Figure 1 shows a process model to handle a travel request using BPMN [OMG09] notation. The process starts with a travel request filed by an employee. Next, the request is processed by a manager. In case the request is declined, the employee is informed about this declination and the process terminates. On the other hand, in case the manager approves, an administrative assistant handles the booking of the flight and hotel and forwards this information to the employee. To prevent fraud, due to fake travel requests, a separation of duty constraint was added between activity “File Travel Request” and “Process Travel Request”.

Although the SOD constraint was explicitly mentioned on the process design, it is possible to cheat that constraint if appropriate runtime monitoring was not established. One possibility of cheating is that a middle manager who is a member of the employees role as well as the manager role can approve travel requests for himself. Another possibility of cheating comes from the dynamic nature of human performers. For instance, within an execution environment that supports delegation, it is possible that a manager delegates the

approval of the request to the employee. Although the first allocation of tasks was compliant, violation occurred because the employee was empowered to approve the request.

This example shows the need to enforce constraints and monitor them online. Also, monitoring of such constraints needs a level of visibility on process execution beyond the declaration of activities start and termination.

3 Compliance Monitoring Framework

The IT infrastructure of an organization does not comprise only BPM engines but is far more complex and a series of system are involved: enterprise systems, BPM systems, production systems, complex event processing (CEP) [Luc02] etc. The topic of monitoring has been addressed from different angles in enterprise system, BPM systems, CEP. We aim at developing a framework for compliance through monitoring which is more comprehensive and follows a broader understanding by incorporating a set of characteristics that arise as a mixture of this wide pallet of interacting fields and perspectives on monitoring.

Compliance requirements are communicated between a compliance officer and a business expert on a high level and in business terms. In most of the cases, those experts are able to use an informal notation to relate compliance requirements, constraints, to tasks within business processes at design time. However, in order to ensure compliant execution of business processes, these constraints have to be translated into rules that are *monitored* at run time.

Thus, there are two levels of abstraction, the business level and the technical level. These two levels are independent from the technology supporting each of them. For instance, at the business level, it does not matter which modeling language is used to create process models or to express compliance constraints. Similarly, on the technical level, it is independent of a specific execution engine.

At the business level, the rules are related to processes by means of using common vocabularies to describe activities, in process models and compliance rules. To carry this correspondence to the technical level, the transformation of process models to executable ones as well as business level rules to technical ones should be based on the same activity lifecycle. This lifecycle is supposed to be followed by the *process execution environment*. With each transition in that lifecycle of an activity instance, events are emitted to an event cloud. Based on the same lifecycle, technical level compliance rules listen to the emitted events, filter them, create business-level events that are listen to also by compliance rules. At the time a business event is captured and found out to contain a violation, the *runtime monitor* responds to the violation, e.g., by blocking the execution of the process instance. This architecture of the discussed framework is depicted in Figure 2.

An important aspect of our framework is to address the common concepts on the two levels and the mapping between them. We discuss the concepts needed to support modeling compliance at design time in Section 3.1. Concepts to support compliance monitoring are discussed in Section 3.2 along with their relation to design time concepts. Finally, we show how the scenario from Section 2 can be addressed within our framework.

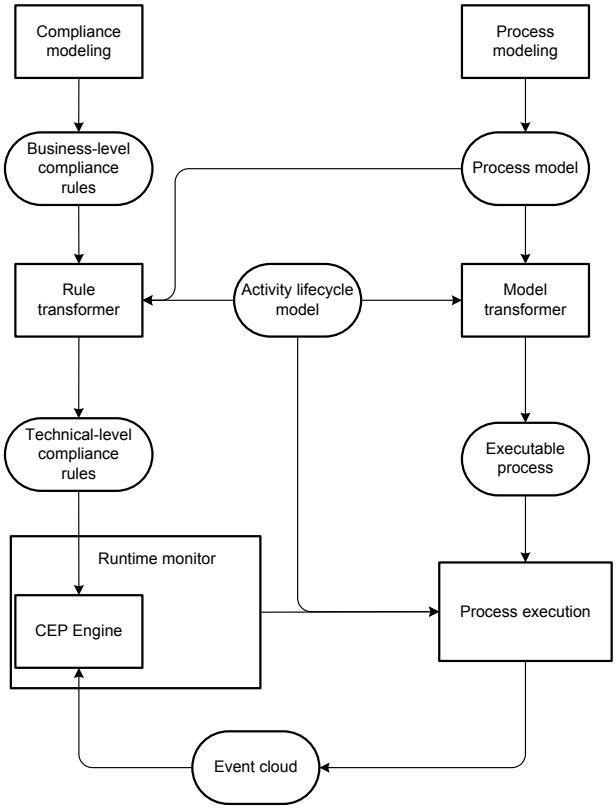

Figure 2: Compliance monitoring architecture

3.1 Design Time

Figure 3 depicts the metamodel for the design-time perspective. This perspective comprises information that refers strictly to the business level and should not be clattered with information that concerns execution. Any information that might refer/influence/concern execution has to be extracted/abducted from the design time perspective. While in the enterprise systems this layer *represents system requirements and other artifacts are produced prior to system deployment* [Rob06] here it comprises the process model, artifacts such as `Resources` and any `Constraints` that might by attached to it.

A `Process` contains `FlowNodes`. `FlowNodes` can be either `Tasks`, `BusinessEvents` or `Gateways`. In accordance with [RvdAtHE05], a task corresponds to a single unit of work. Atomic, Block (SubProcess), multiple-instance and multiple-instance block are foreseen as subtypes of a Task.

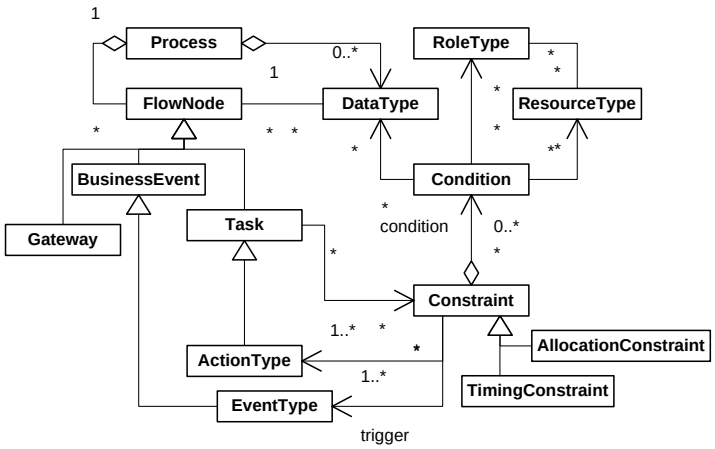

Figure 3: Design Time metamodel

A Process, as depicted in the metamodel, might also contain Data. As a general rule, since we are at design time, we are talking about types. To emphasize this different concept, we have attached the word `Type`.

We are addressing the problem of process compliance. Tasks in the processes have to be performed, by entities that are capable of doing work. The `Resource` concept stands for an entity that is capable of actually performing the work behind tasks modeled in a process. Resources might impersonate either humans or non-humans. With the Role concept, organizational perspective of processes is addressed. Namiri and Stojanovic [NS07] distinguish three roles involved in business process compliance: *business process expert*, *compliance expert* and *external auditors*. We are addressing a different perspective, and by `Role` we mean organizational perspective, e.g., *administrative assistant*, *manager*, *employee*. These roles are involved in the execution of the process. Roles subsumed by our `Role` concept are not designing the compliance constraints as in [NS07] but constraints are defined to validate their work.

To be able to validate processes against compliance requirements, constraints have to be attached to elements of processes. A set of `Constraints` could be attached to a Task. A constraint is triggered by one or more `Events`. We understand here also complex events. Constraints might have several `Conditions`. If a constraint holds then at least one `Action` is performed. Conditions refer to `Data`, `Resources` or `Roles`. Actions that be performed as result for verifying a constraint can be either system actions or business action, e.g., a task from the process could be invoked. Similarly to this situation, events that trigger a constraint could be either low level events or business events, e.g., a message event modeled in the process.

Special types (that follow a concrete pattern) of constraints can be defined. Right now, two types of such constraints are foreseen: `TimingConstraints` and `AllocationConstraints`. Specific constraints can be defined based on patterns

[NS07] to ease and to guide the modeler in defining constraints. Based on the context, at design time the event's type might be unknown. In such situations, at deployment it will be made concrete by mapping to a concrete event through the usage of pattern constraints, e.g., `AllocationConstraint`. Decker et al. define in [DGB07] means to model complex events in the process model. Thus for situations where a complex business event is required; this can be a way to define it and later use in the compliance constraint.

We do not enforce a specific language to define constraints; the modeler could provide the constraint using e.g. Semantics of Business Vocabulary and Business Rules (SBVR) [OMG]. SBVR is an Object Management Group (OMG) specification for business modeling. In SBVR meaning is kept separte from expression, thus allowing to express the same thing in different ways. Textual form allows higher flexibility in defining vocabularies and expressing rules [GMV07]. SBVR Structured English is part of the SBVR specification and it is a structured English vocabulary for describing vocabularies and verbalizing rules. SBVR has been already used in the context of processes [GMV07].

The fact that SBVR is an OMG standard for business and that it has been already used in the context of processes where the reasons for which we choose to exemplify here with it our approach. See Example 1 for specifying a separation of duty constraint in SBVR. At run time, by means of an adapter, constraints will be transformed into specific runable rules. We present here only a rule using SBVR specific styling and formatting in accordance with the OMG standard. For an in depth discussion on how to define the vocabulary and process specific elements one could refer to [GMV07].

The followings are the main concepts from SBVR structured English and their representation:

- term: stands for a noun concept
- Name: stands for an individual of a particular noun concept
- *verb*: verb
- keyword: reserved words

Example 1 (Separation of Duty Constraint at Design Time).

The performer of the File a Travel Request task must not be the performer of the Process Travel Request task.

3.2 Run Time

The run time model represents the system execution [Rob06]. The metamodel is presented in Figure 4.

Though, when it comes to processes the situation changes. According to the notion of a process instance, at run time, instances of a process are known as cases of a single process model. Invocation of tasks are termed as `WorkItems`, we stick to the terminology

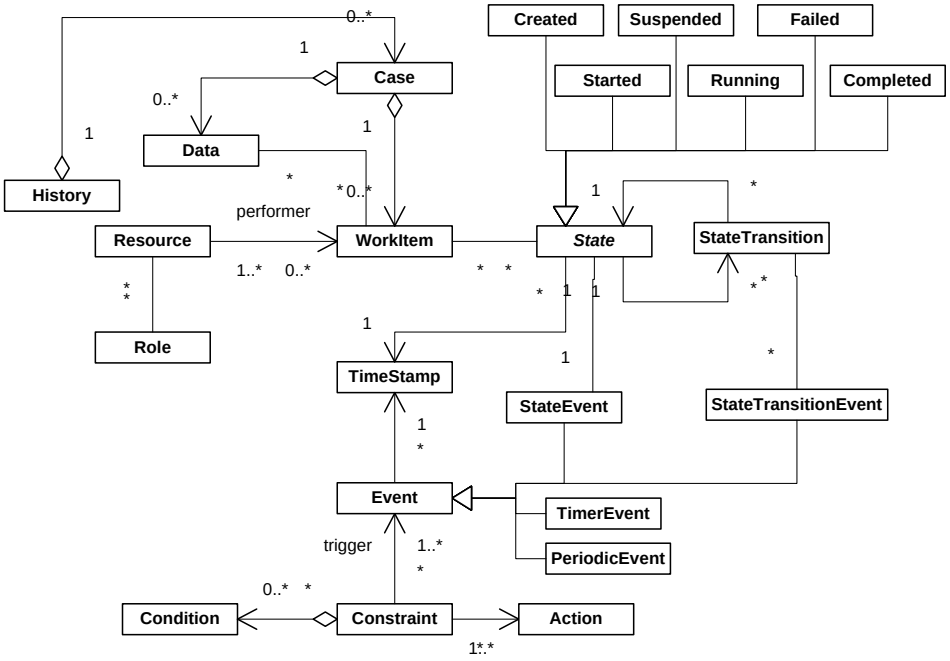

Figure 4: Run Time metamodel

introduced in [RvdAtHE05]. Usually, there is one work item for each task in a given process. In case of multiple instance tasks there could be created several work items. An `WorkItem` is performed by one or more `Resources`. Although other approaches, e.g., [RvdAtHE05], address the concept of `State` for an `WorkItem`, nothing is said about the way in which the state is made public to the execution engine. In conformance with [LF98] we argue that the state of a work item is private to the work item; it can not be directly accessed. The proper way to know about the changes is by means of events. In our metamodel in Figure 4, there is a one to one association between a `State` and a `StateEvent`. We take into consideration also the intermediary phase, between states. We call this a `StateTransition`. Every state transition is also announced by means of a `StateTransitionEvent`. This fine grained granularity provides greater monitoring capabilities and a wide range of compliance checking and recovery from situations where violations have been encountered. States are refined according to [RvdAtHE05] into `Created`, `Started`, `Suspended`, `Running`, `Failed`, `Completed`. With such a view a system implementing the framework will be almost completely event based [PG09].

Table 1 depicts the almost straight forward mapping between the design time concepts and the run time ones. `ResourceType` is at run time a `Resource`. Similarly `RoleType` is `Role`, `DataType` is `Data`, `EventType` is `Event` and `ActionType` is `Action`.

Complex event processing (CEP) [Luc02] is a technology *to extract information from distributed message based systems* [LF98]. Event hierarchies as they are defined in [LF98]

Design-time	Run-time
ResourceType	Resource
RoleType	Role
Process	Case
Task	WorkItem
DataType	Data
EventType	Event
ActionType	Action
Constraint	Constraint

Table 1: Mapping design-time concepts to run-time concepts

provide views on the activities of a system at different levels. Low level events or system events reside at the lowest level. Events from the higher levels are *virtual events* [LF98] and are mainly created by aggregating events from lower levels. [LF98] defines two ways for creating high level events: *filters* and *maps*. Both approaches take use of event patterns. Maps are also called *aggregators*. Their input is represented by posets of events (partially ordered set of events) to create higher level events.

From an architectural point of view, an `EventManager` is the component in charge of, among other things, catching (listening) of events, querying for events from a cloud of events that it manages, or from a History and matching sequences of events against defined event patterns and through an aggregator; thus to create higher level events. The creation of high level events done by means of Event-Condition-Action (ECA) rules. Walzer et al. [WBG08] discuss this issue as an extension of the RETE [For82] algorithm. In this context, rules have as actions creation of new events. A more refined discussion about the execution process of an EventManager is provided in [PG09].

We do not intend to define ontologies for business and domain specific events. The reader could refer to [Pas08] for work towards the direction of ontologies and design patterns. We use the general concept of `Event` to address the notion of event. Though, we refine the high level notion of event with a set of sub classes of events that have direct implication in monitoring and compliance enforcement, e.g., `StateEvent`, `StateTransitionEvent`, `TimerEvent`, `PeriodicTimeEvent`.

The ability to impose compliance or the ability to recover from compliance breaches is achieved with the concept of `Action`. An action can be either a high level business task, which has been modeled in the process already, or a low level action such as blocking of a work item. Being able to invoke business tasks, we achieve what von Ammon et al. [vAEE+09] call: dynamic changes of processes. On the other hand [NS07] provides a list of low level actions that can be used in compliance enforcement: `Ignore`, `Block`, `Notify`, `Retry`, `Rollback`, `Recover`.

The `History` stores information about the `Cases` that have been run. Thus we provide the means through which after execution checks can be performed, or address situations when historical data is required in the execution of a current process.

A Monitor, from an architectural point of view is the supervisor and the component in charge of dealing with monitoring and compliance enforcement. It keeps track of the running processes and stores cases into history. It comprises two major components the InferenceEngine and the EventManager.

3.3 Implementing the SOD scenario

In Section 2, we described a separation of duties scenario that serves as a basis for exemplifying our framework. This section is devoted to exemplifying and explaining how the use case can be implemented using the framework and concepts introduced in this paper.

The scenario was stating that a violation occurs if a middle manager who is both an employee and a manager could approve travel requests for himself. Based on Figure 1, such a person would be able to perform both the File a Travel Request task which is located in the Employee lane as well as the Process Travel Request task from the Manager lane in the same figure. Also in an environment that supports delegation, it would be possible that a manager delegates the approval of a request to an employee.

To implement constraints triggered by events as defined in our framework we use ECA rules of the form:

```
ON E1 && ... && E2 IF C1 && ... && Cn DO [A1, ..., Am]
```

To enforce compliance according to the SOD scenario, we have to verify two things: (1) separation of duties and (2) separation of duties through delegation. For the first case, separation of duties without delegation we have to verify that the performer of the File WorkItem and the performer of the Process WorkItem are different. They should be different if the Data object TravelRequest is the same. These checks have to be performed each time an AllocationConstraintEvent is raised. Separation of duties with delegation is triggered by a DelegationConstraintEvent and requires an additional check on the role of the performer of the Process Travel Request work item.

The AllocationConstraintEvent is a high level business (in the sense that it refers and has meaning in compliance enforcement) event. This event is created by means of aggregation [LF98] from a set of low level events. High level AllocationConstraintEvent is raised when in the events cloud we have a StateEvent that refers to the Completed state of a File a Travel Request WorkItem and have another StateEvent that refers to the Created state of a Process Travel Request WorkItem. If such a pattern is matched then a high level AllocationConstraintEvent is raised. Upon catching of such an event the constraint can be checked.

As JBoss provides both a workflow engine jBPM as well as an inference engine that can deal also with events Drools, we are using these platforms for our prototype implementation. Constraints will be defined using Drools language.

Example 2 provides an example of a rule used to match a pattern for complex event creation. If there are events that hold the pattern then a complex event is created and fired into the cloud. In Example 2, we look for the low level events, `StateEvents` that should refer to the states of the `WorkItems`: `File a Travel request` and `Process Travel Request`. `File a Travel Request` has to be in `Completed` state and `Process Travel Request` has to be in the `Created` state. If these conditions are met then a complex event `AllocationConstraintEvent` is created and raised.

Example 2 (Complex Event Creation).

```
rule "raise AllocationConstraintEvent"
when
    $ev1:StateEvent(state.type=="Completed",
        state.workItem.name=="File a Travel Request",
        $w1:state.workItem)
    $ev2:StateEvent(state.type=="Created",
        state.workItem.name=="Process Travel request",
        $w2:state.workItem)
then
    raiseNewEvent(AllocationConstraintEvent, w1, w2)
end
```

When `AllocationConstraintEvent` is caught, the separation of duties constraint can be checked (see Example 3). This implies checking that `File a Travel Request` and `Process Travel Request` refer to the same `Travel Request` data object, an alternative could be to check that the two work items belong to the same process instance (case). In addition, we check if the performers of the two work items are the same. If so, we have a violation of separation of duties and we block the execution of the `Process Travel Request WorkItem`.

Example 3 (Separation of Duties).

```
rule "check separation of duties"
when
    $ev:AllocationConstraintEvent()
    $d:Data(name="Travel Request")
    $w1:WorkItem(name=="File a Travel Request",
        this==$ev.workItem1,
        $w1Performer:performer,
data==$d)
    $w2:WorkItem(name=="Process Travel Request",
        this==$ev.workItem2,
        this==$w1Performer,
data==$d)
then
    block($w2)
end
```

Separation of duties with delegation, Example 4, checks in addition to the simple separation of duties in Example 3 that the role of the `Process Travel Request` performer is `Manager`.

Example 4 (Separation of Duties with delegation).

```
rule "check separation of duties with delegation"
when
    $ev:DelegationConstraintEvent()
```

```
    $d:Data(name="Travel Request")
    $w1:WorkItem(name=="File a Travel Request",
        this==$ev.workItem1,
        $w1Performer:performer,
data==$d)
    $w2:WorkItem((name=="Process Travel Request",
        this==$ev.workItem2,
        this!=$w1Performer,
data==$d, role!="Manager"))

then
    block($w2)
end
```

4 Related work

Requirements monitoring in enterprise systems [Rob06] addresses the issue of misalignment between systems and their policies. A distinction is made between policies, goals and requirements. Requirements refine goals as stated in [Rob06] based on three properties: *(1) it is described entirely in terms of values monitored by the software; (2) it contains only values that are controlled by the system; (3) the controlled values are not defined in terms of future monitored values*. Unfortunately requirements monitoring in enterprise systems do not address business processes. The framework introduced in the paper in hand starts by having business processes as the backbone of the enterprise infrastructure. In this context, monitoring of the enterprise environment must be performed in relationship with the business processes that drive the business.

In [NS07] Namiri and Stojanovic discuss a pattern based approach to compliance validation. In addition, they provide metamodels to capture internal controls, compliance constraints, and how to relate them to business processes. The approach discusses compliance patterns from a high level point of view. In comparison to our framework, we are concerned with run time monitoring of compliance constraints and how low level events at process execution are aggregated to trigger business events that need response. Moreover, we address the monitoring of resource behavior, e.g., separation of duties.

[MJDP02, GM05] discuss the problem of *monitoring* of contract execution. By monitoring of significant events, the comparison of actual behavior to the expected behavior is possible. In case of violation, several resolution mechanisms are offered depending on the severity of violations. The approach discusses monitoring of significant events without identifying how these significant events can be derived from process execution events.

Giblin et al. [GLM+05] propose REALM as a metamodel to express compliance policies. A policy is a rule set that has a scope of applicability. Rules are expressed by means of real-time temporal object logic. This gives expressiveness not only to capture ordering between events; rather, the actual times of their occurrences. This is necessary to express timing constraints, e.g., an activity must not take more than two days. In addition to enforcing these rules on process definitions, they are monitored at run time. This approach is limited to control flow related events where human interactions with processes are not addressed.

Ontology languages are used by ExPDT [KGM08] to express privacy compliance rules. With ExPDT, security policies concerning both data and access to them by business processes can be expressed. One can express a ExPDT rule indicating conditions on the user, the data and the action taken in the business process. The rules are monitored at runtime to enforce compliance.

5 Conclusion

We have introduced a framework for instant monitoring of business process compliance. While compliance constraints can be defined at design time by compliance and business experts, enforcing the constraints must be performed at enactment time. The framework emphasizes the fact that beside the usual service oriented character of business process execution, humans also play an important role in process enactment. Thus, frameworks that use monitoring approaches to enforce compliance must address also the human resources that interact with enterprise systems.

Enforcing compliance, in scenarios such as separation of duties, requires a dynamic approach. The idea introduced here uses complex event processing and rules to achieve the stated goals. Complex event processing plays an important role in the process of monitoring and compliance enforcement as events are the way to perceive the state of an enterprise system.

Future work comprises evaluation and comparison of our work with the exiting workflow and business process engines with respect to the abstraction layers: design and run time concepts used.

References

[DGB07] Gero Decker, Alexander Grosskopf, and Alistair Barros. A Graphical Notation for Modeling Complex Events in Business Processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC2007)*, 2007.

[For82] Charles Forgy. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[GLM+05] Christopher Giblin, Alice Y. Liu, Samuel Müller, Birgit Pfitzmann, and Xin Zhou. Regulations Expressed As Logical Models (REALM). In *Proceeding of the 2005 conference on Legal Knowledge and Information Systems*, pages 37–48, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press.

[GM05] Guido Governatori and Zoran Milosevic. Dealing with contract violations: formalism and domain specific language. In *EDOC*, pages 46–57. IEEE Computer Society, 2005.

[GMV07] Stijn Goedertier, Christophe Mues, and Jan Vanthienen. Specifying Process-Aware Access Control Rules in SBVR. In Adrian Paschke and Yevgen Biletskiy, editors,

Proceedings of the Advances in Rule Interchange and Applications (RuleML2007), volume 4824 of *LNCS*, pages 39–52. Springer-Verlag Berlin Heidelberg, 2007.

[KGM08] Martin Kähmer, Maike Gilliot, and G. Muller. Automating Privacy Compliance with ExPDT. In *CEC/EEE*, pages 87–94. IEEE, 2008.

[LF98] David C. Luckham and Brian Frasca. Complex Event Processing in Distributed Systems. Technical report, Stanford University, 1998.

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

[MJDP02] Zoran Milosevic, Audun Jøsang, Theodosis Dimitrakos, and Mary Anne Patton. Discretionary Enforcement of Electronic Contracts. In *EDOC*, pages 39–50. IEEE Computer Society, 2002.

[MPS08] Jan Mendling, Karsten Ploesser, and Mark Strembeck. Specifying Separation of Duty Constraints in BPEL4People Processes. In *BIS*, volume 7 of *Lecture Notes in Business Information Processing*, pages 273–284. Springer, 2008.

[NS07] Kioumars Namiri and Nenad Stojanovic. Pattern-Based Design and Validation of Business Process Compliance. LNCS, pages 59–76. Springer, 2007.

[OMG] OMG. Semantics of Business Vocabulary and Business Rules (SBVR), v1.0, organization=OMG, year=2008, howpublished=http://www.omg.org/spec/SBVR/1.0/PDF.

[OMG09] OMG. Business Process Model and Notation (BPMN). FTF Beta 1 for Version 2.0. `http://www.omg.org/spec/BPMN/2.0`, 2009.

[Pas08] Adrian Paschke. Design Patterns for Complex Event Processing. In *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS2008)*, ACM International Proceedings. ACM, 2008.

[PG09] Emilian Pascalau and Adrian Giurca. A Lightweight Architecture of an ECA Rule Engine for Web Browsers. In *Proceedings of 5th Knowledge Engineering and Software Engineering, KESE 2009*, volume 486. CEUR-WS, 2009.

[Rob06] William N. Robinson. A requirements monitoring framework for enterprise systems. *Requirements Engineering*, 11:17–41, 2006.

[RvdAtHE05] Nick Russell, Wil M.P. van der Aalst, Arthur H. M. ter Hofstede, and David Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520 of *LNCS*, pages 216–232. Springer-Verlag, Berlin, 2005.

[vAEE+09] Rainer von Ammon, Thomas Ertlmaier, Opher Etzion, Alexander Kofman, and Thomas Paulus. Integrating Complex Events for Collaborating and Dynamically Changing Business Processes. In *Proceedings of the 2nd Workshop on Monitoring, Adaptation and Beyond (MONA+), collocated with ICSOC 2009*, 2009.

[WBG08] Karen Walzer, Tino Breddin, and Matthias Groch. Relative temporal constraints in the Rete algorithm for complex event detection. In *Proceedings of the second international conference on Distributed event-based systems (DEBS'08)*, pages 147–155. ACM New York, NY, USA, 2008.

[WMM09] Christian Wolter, Philip Miseldine, and Christoph Meinel. Verification of Business Process Entailment Constraints Using SPIN. In *ESSoS*, volume 5429 of *LNCS*, pages 1–15. Springer, 2009.

[WS07] Christian Wolter and Andreas Schaad. Modeling of Task-Based Authorization Constraints in BPMN. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2007.