SOCA (2013) 7:59–73
DOI 10.1007/s11761-012-0121-3

ORIGINAL RESEARCH PAPER

Domain-specific language for event-based compliance monitoring in process-driven SOAs

Emmanuel Mulo · Uwe Zdun · Schahram Dustdar

Received: 7 October 2011 / Revised: 11 August 2012 / Accepted: 7 September 2012 / Published online: 26 September 2012

Abstract Organizations today are required to adhere to a number of compliance concerns from laws, regulations and policies. Compliance is achieved through defining and implementing so-called controls in the organizations' business processes. Organizations that build their systems based on the process-driven SOA paradigm realize business processes through orchestration of services to handle the process' business activities. These business activities or groups of business activities in some cases realize the compliance controls. We propose an approach for implementing event-based compliance monitoring infrastructure that observes such business processes to verify that compliance is indeed adhered to. Our approach is essentially a model-driven technique for realizing this infrastructure. We implement a domain-specific language for specification of compliance directives, and we include code generation templates to generate compliance monitoring code, which is leveraged by complex event processing components to monitor for compliance. We evaluate the impact of our approach on the effort and productivity of a developer who is specifying compliance directives.

Keywords Compliance · Domain-specific language · Event-based · Process-driven · Service-oriented architecture · Monitoring

1 Introduction

The service-oriented architecture (SOA) paradigm is today very much utilized in implementations of enterprise information systems in organizations. SOA-based systems are designed to have different functions encapsulated as services. A process-driven SOA [1] additionally introduces a process engine that orchestrates these services to perform the different activities that make up a business process. In large-scale process-driven SOA systems, multiple process instances are executed and coordinated on multiple process engines [1,2]. All process instances are realized through invoking operations from a pool of services that are within and sometimes beyond the boundary of an organization.

Process-driven SOA systems increasingly support organization operations and processes and, therefore, are subject to a number of compliance concerns required of organizations. Compliance concerns include, among other things, acts of law from governments, or regulations drawn up by regulatory authorities, that govern the way organizations should run their affairs. Non-adherence to these compliance concerns may result in consequences such as loss of credibility, financial loss, and litigation of an organization [3]. These (compliance) concerns are implemented in the SOA systems in the form of IT controls; the controls may either be preventive, that is, prevent violations from occurring, or detective, that is, detect the violations after they have occurred. In order to ensure adherence to compliance, monitoring components can provide an audit trail of the correct execution of preventive controls. In cases where preventive controls are hard to implement, for example, failure of a system, service, or human operation, monitoring components can play a more active role by ensuring fast detection of compliance violations [4]; in other words, the monitoring component *itself* is the compliance control. In many organizations, however, implementation of compliance controls does not follow a generic strategy; compliance is reached on a per case basis, with ad hoc, hand-crafted solutions or niche products used for specific compliance scenarios [3,5–7]. Additionally, these

E. Mulo (✉) · S. Dustdar
Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Vienna, Austria
e-mail: e.mulo@infosys.tuwien.ac.at

U. Zdun
Faculty of Computer Science, University of Vienna, Vienna, Austria

compliance controls are scattered throughout an organization system without a clear architectural concept; in some cases, controls are duplicated. This creates difficulties when it comes to maintenance of compliance controls. Moreover, compliance regulations change frequently, meaning the controls have to be constantly updated as well. The goal of this study is to address these issues through a model-driven approach to implement a compliance monitoring component within the scope of process-driven SOA systems.

Our main contributions are a structured approach and tool support to realizing a compliance monitoring component for process-driven SOAs through model-driven development (MDD) techniques. Our approach essentially involves iteratively identifying the controls to be monitored, capturing them (controls) through a domain-specific language (DSL) that we have developed, and finally generating an event-based compliance monitoring component. The resulting component performs runtime monitoring of compliance controls that are realized as process activities or subprocesses. Our tool enables mapping of process workflows into patterns of events that are (re)used within our DSL. We believe this mapping scheme is a step in the direction toward automated management of monitoring components. Considering that business processes and their compliance controls are constantly changing, we feel that our approach enables rapid development and evolution of a compliance monitoring component. In previous work [8], we demonstrated the feasibility of event-based compliance monitoring; we developed a prototype of such a compliance monitoring component and analyzed its performance and scalability by running test scenarios realistically mimicking a large-scale process-driven SOA with event monitoring. In this work, we use a number of scenarios from industry case studies to evaluate the impact of our approach on effort and productivity of a developer when coding compliance monitoring specifications.

The rest of this paper is structured as follows. Sect. 2 gives background information concerning compliance and compliance monitoring components. Also included in this section are a number of illustrative business process compliance scenarios. Next, Sect. 3 explains our proposed approach, and in Sect. 4, we evaluate our approach based on the scenarios taken from industry case studies. Section 5 compares our work to the related work, and finally Sects. 6 and 7 present discussions and a conclusion to this work.

2 Business process compliance

In this section, we present background information concerning business compliance and provide a scope for the kind of compliance that we deal with in this article. We also present limitations associated with the approach to implementing compliance in organizations today and how these may affect compliance monitoring. Additionally, we present some background information into complex event processing, a technique upon which the construction/generation of our compliance monitoring component is based. Finally, we present a number of illustrative scenarios for business process compliance.

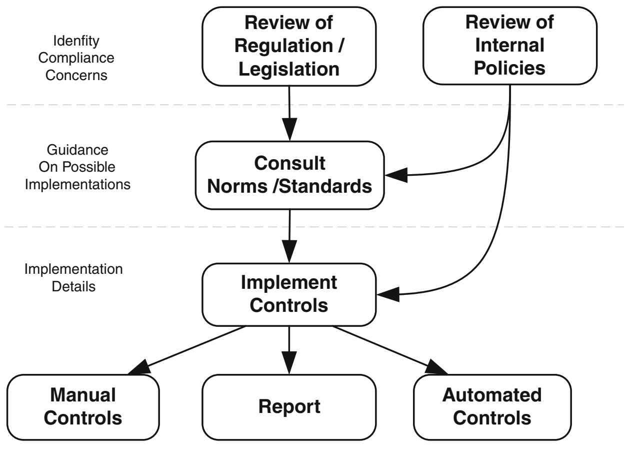

Idenfity Compliance Concerns
Guidance On Possible Implementations
Implementation Details
Review of Regulation / Legislation
Review of Internal Policies
Consult Norms /Standards
Implement Controls
Manual Controls
Report
Automated Controls

Fig. 1 Overview of business compliance assurance

2.1 Compliance in organizations

Typically, when *business compliance* is discussed, one thinks of the goal to ensure that the systems of an organization comply with *regulatory* or *legislative provisions* or similar business provisions originating from parties external to the organization. Common examples include regulations set forth in the Basel II Accord,[1] The Dutch Corporate Governance Code,[2] and the Sarbanes-Oxley Act (SOX),[3] to name a few. These cover issues such as auditor independence, corporate governance, and enhanced financial disclosure. Compliance provisions may also originate internally from organization policies concerning how the internal processes are executed. We use the term *compliance concerns* as an umbrella covering these internal and external provisions. Figure 1 gives a typical high-level overview of steps an organization goes through to realize compliance. In the diagram, rectangles with rounded edges represent activities, and arrows show transitions from one activity to the next. We also use dotted lines to group activities under common themes.

Compliance concerns are related to risks an organization and its stakeholders face in achieving their mission. They provide guidance regarding measures to take to prevent occurrence of these risks, for example, Section 404 of the SOX

[1] http://www.bis.org/publ/bcbs107.htm.

[2] http://www.commissiecorporategovernance.nl/Corporate_Governance_Code.

[3] http://www.gpo.gov/fdsys/pkg/CRPT-107hrpt610/pdf/CRPT-107hrpt610.pdf.

Act requires public companies to annually assess and report on the design and effectiveness of internal control over financial reporting. However, the specifics of *how* to implement risk-reduction measures are not normally addressed in these guidelines. These specifics have to be handled on a per organization basis.

Risk-reduction measures are normally implemented as so-called *controls* [9]. A control is any measure taken to assure a compliance concern is met. Controls may be broadly classified into preventive controls and detective controls. Preventive controls aim to avoid risks, whereas detective controls warn of the occurrence of risks. Both classes of controls can be realized as manual controls, automated controls or hybrid controls (a combination of the previous two) [10]. For instance, a door alarm system (manual), a software system realizing segregation of duty requirements (automated), and management reports on system errors/faults (hybrid) are all implementations of controls. During risk management exercises, organizations make decisions regarding required controls and their implementation.

Organizations need not invent all controls from scratch. A number of established norms or standards define and describe standard controls that can be adapted and implemented. The Control Objectives for Information and Related Technologies (COBIT)[4] framework, for example, describes control objectives that guide an organization in making choices about which controls to implement. Such norms and standards are fairly generic and abstract and must be mapped to a concrete systems implementation.

In our work, we focus on monitoring such compliance controls (especially the automated and hybrid controls) in the context of process-driven SOAs, which we discuss in the next section.

2.2 Compliance and business processes

Business processes comprise a collection of related, structured business activities within or across organizations, which produce a specific service or product for a particular customer [11]. Within a process, there typically exists a combination of manual and automated steps (activities/tasks). Business activities may be either atomic or non-atomic [12]. Atomic activities are also known as tasks, whereas non-atomic activities are termed subprocesses. Subprocesses constitute a number of business activities executed in a specific control flow. Upon completion of this flow, the subprocess is said to have completed execution. Whereas a business process is primarily aimed at satisfying a particular client need, one of the major considerations taken into account when designing processes is compliance concerns.

[4] www.isaca.org/cobit/.

During the risk assessment exercises, an organization identifies the compliance concerns that it is required to fulfill, and then makes decisions regarding which controls to implement in order to address those concerns. Controls may constitute or be applied to subprocesses or activities within a business process, that is, parts of the process may be designed and implemented such that they realize a control that fulfills a particular regulation. Consider, for example, the case of the well-known segregation of duty (SoD) control. In this control, it is mandatory for certain activities to be executed by two distinct persons in order to avoid fraud. The control may be implemented in such a way that when the same person attempts to execute conflicting activities, the system does not allow this action to happen or allows it but reports this as a violation of compliance.

Using the existing norms and standards that provide guidance on which controls to use, an organization can make choices on how to concretely implement the prescribed compliance controls for their business processes. The implementation of controls usually does not follow a generic strategy and hence business compliance is reached on a per-case basis, that is, organizations use ad hoc, hand-crafted solutions for specific compliance concerns [3,5–7]. This usually means that a separate project is started and develops an individual, custom solution for the compliance concern to be addressed. Such solutions usually do not follow a clear architectural concept and result in hard-coded controls spread over the systems, possibly with dependencies to other controls. Consequently, it becomes quite a task to ensure compliance and keep up with constant changes in regulations and laws. In this article, we present our approach as a generic strategy to realize a runtime compliance monitoring component in the context of process-oriented SOAs. Our approach essentially follows a model-driven development paradigm for realizing a runtime compliance monitoring component.

2.3 Compliance monitoring through complex event processing

A number of generic monitoring solutions propose an external component to which events are sent. In some cases, the component records events in audit logs (files) and later analyzes them to detect anomalies in system behavior [2,13–15]; these are characterized as offline monitoring solutions. Other solutions propose monitoring events in near real-time using complex event processing (CEP) techniques [16–18]. This approach is characterized as online monitoring and implies that violations in the expected behavior of the system are ideally detected *sooner* after they have occurred.

CEP is a set of tools and techniques for analyzing and controlling a complex series of interrelated events [16]. Events are observed as they occur and are correlated and aggregated in order to discover and respond to certain event patterns [18]. These techniques have a number of application areas, including policy enforcement and regulatory compliance [16,18]. Events represent the occurrence of an activity within a system. They may originate from a number of sources, for example, RFID tags, network traffic data, and enterprise application components, and they may contain information, for example, the event source, or time of occurrence. This information enables us to analyze the events and determine how they relate to each other.

Domain specialists are interested in events of significance in their domain. In some cases, it is not possible to directly observe such special events because they occur as a combination of a number of other events. However, through event pattern languages (EPLs) [16] also known as stream query languages [19], one can configure an event processing engine to aggregate what are termed low-level events into complex (high-level) events. Low-level events are not an abstraction of other events and do not have semantic significance on their own within a specific domain, whereas complex events are an aggregated abstraction of a number of other low-level and/or complex events [16,20]. EPL's are the primary tool through which CEP-based components can be configured to filter and correlate low-level events to yield other higher-level, more semantically significant events [21], that is, special events.

In previous work, we leveraged CEP techniques for the realization of a compliance monitoring component for process-driven SOAs [8]. We discovered some challenges when using CEP for compliance monitoring, especially regarding maintainability of the compliance controls. We build on top of this work with our model-driven approach to implementing compliance monitoring components.

2.4 Illustrative scenarios of business process compliance

In order to illustrate the relationship between compliance and business processes, we present here a number of illustrative scenarios from industry case studies of business process compliance. We use the BPMN standard notation [12] for the diagrams illustrating these scenarios.

2.4.1 Scenario 1: Loan application (LA) scenario

In this scenario, a customer goes to a bank to apply for a loan. In Fig. 2, we present an excerpt of the process model indicating steps typically followed. In this excerpt, a person with the role Credit Broker performs a number of tasks upon receiving a loan request. Upon completion, the loan application may be delegated to a Supervisor or to a Post Processing Clerk, depending on the size of the loan requested by the applicant.

In this scenario, we consider three compliance regulations that fall under the category of SoD requirements. This

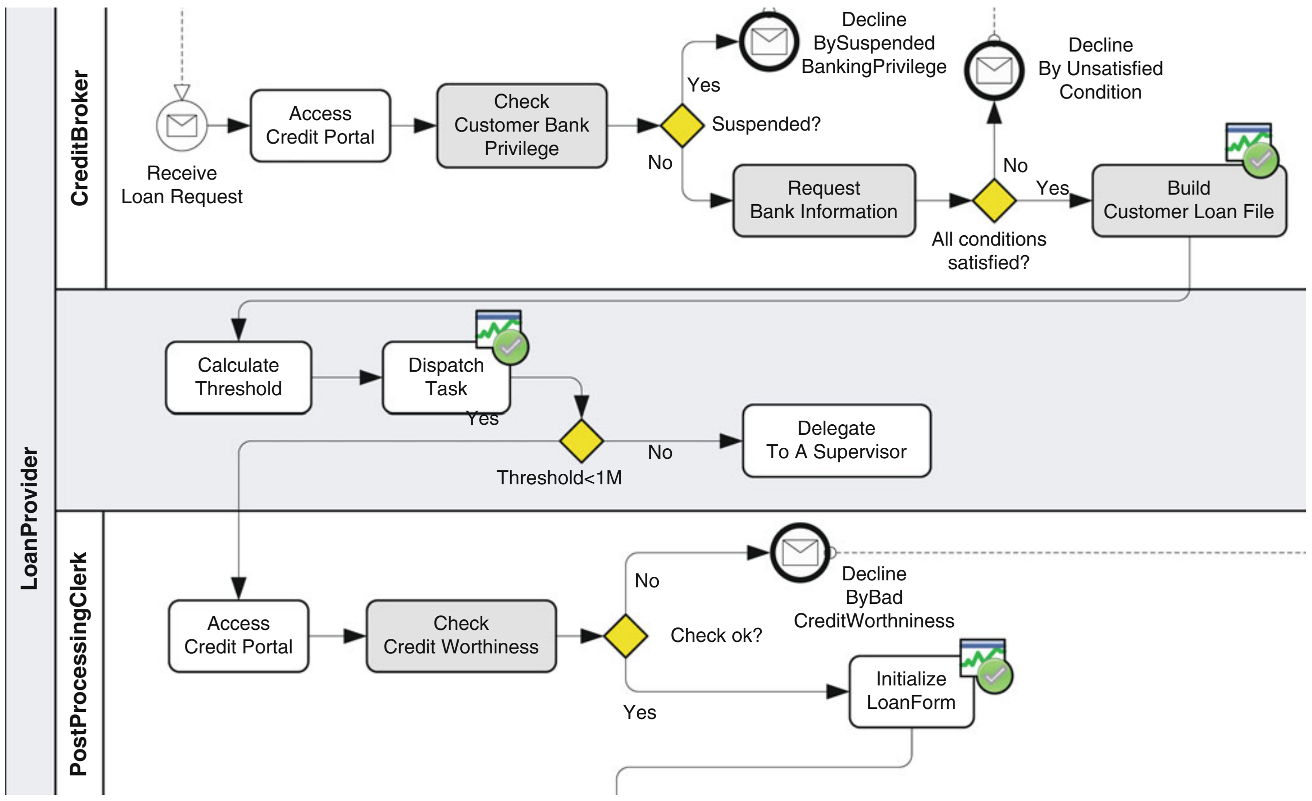


Fig. 2 Excerpt of loan application process

requirement essentially states that activities that pose a risk of fraud or error should not be carried out by the same person. In the loan application process, SoD accounts for two of the compliance rules designed into the process. In Fig. 2, we see that a loan application is initially processed by a Credit Broker, who performs preliminary checks on the customer's application files, banking privileges, and then creates a loan file. However, the next tasks of checking the customer's credit worthiness and approving the loan have to be performed by a Post Processing Clerk. Additionally, if the loan exceeds a threshold, in this case defined as EUR 1 million, the loan application has to be verified by a Supervisor. One final compliance rule implemented in the business process (although not shown in the diagram) states that all final approvals of loans have to be carried out by a person with the role Manager.

2.4.2 Scenario 2: Travel booking (TB) scenario

In this scenario, we consider the business process from a travel management agency (TMA) that handles travel arrangements for corporations. Employees of corporations that are TMA clients are able to arrange their travel through services provided by TMA. TMA has signed contracts with these corporations and has to monitor its applications to ensure that employees using these services adhere to policies agreed upon. We consider two specific policies; first, each corporation decides on preferred suppliers for specific services in order to ease payments. Therefore, employees arranging their travel are encouraged to use these preferred suppliers unless it is absolutely necessary to choose a different one. The other guideline for employees when arranging travel is to arrange their travel whenever possible two weeks in advance of the expected date of travel. Figure 3 shows the business process model for this scenario.

2.4.3 Scenario 3: Claims handling (CH) scenario

In the claims handling scenario, an insurance company has to monitor the process of fulfilling or denying insurance claims from its customers. The insurance company would especially be interested in keeping track of denied claims as these are of interest to regulatory bodies. The process excerpt for the claims handling business process is shown in Fig. 4.

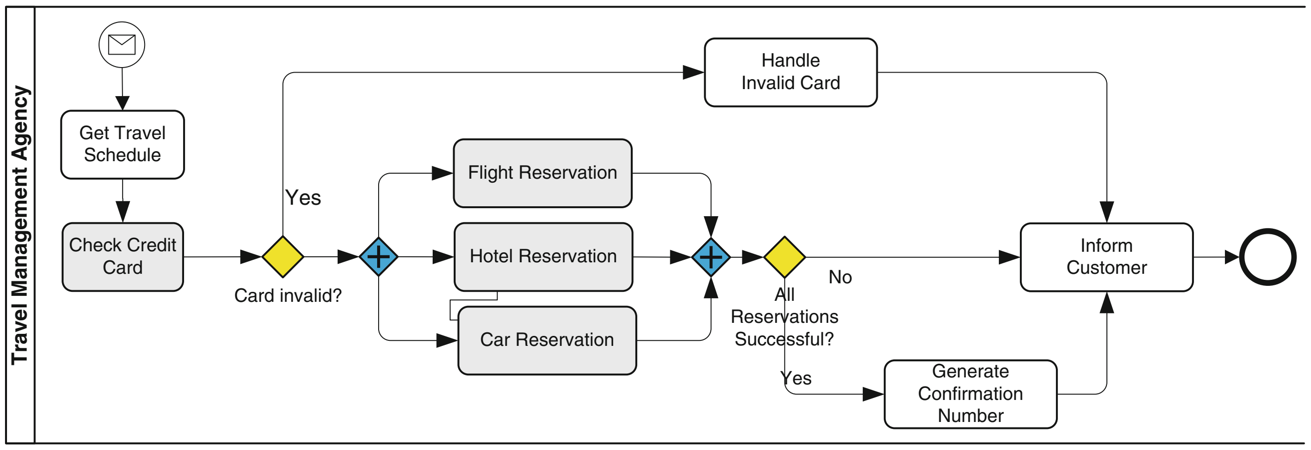


Fig. 3 Travel booking scenario

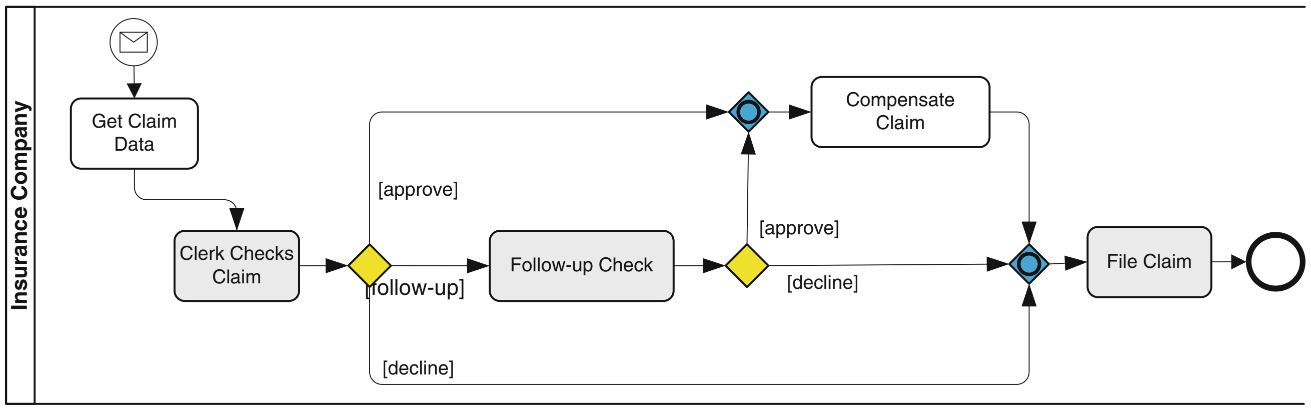


Fig. 4 Insurance claims handling

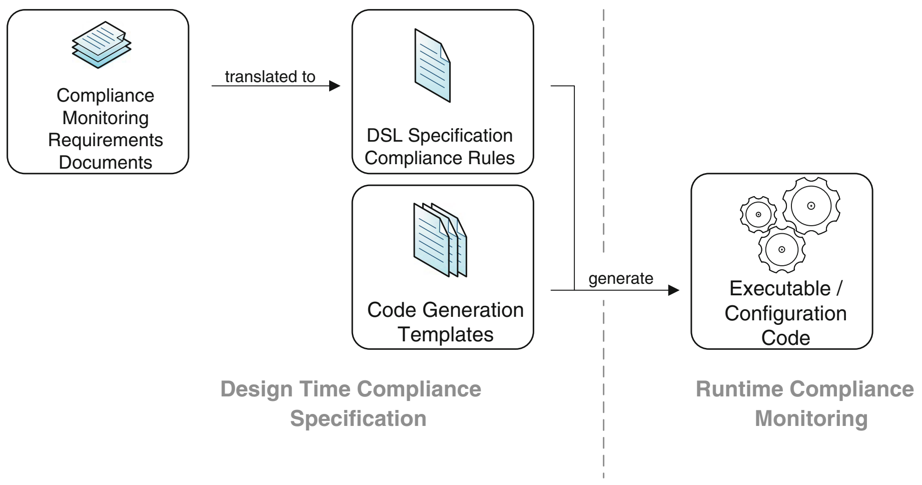

Compliance Monitoring Requirements Documents
translated to
DSL Specification Compliance Rules
Code Generation Templates
generate
Executable / Configuration Code
Design Time Compliance Specification
Runtime Compliance Monitoring

Fig. 5 Conceptual overview of approach

In order to ensure the requirements in the scenarios are adhered to, we implement business process monitors to observe the scenarios during process execution. We use these illustrative scenarios to present our approach in the next section and in the evaluation of our approach in a later section.

3 Developing compliance monitoring components

3.1 Overview of compliance monitoring approach

In process-driven SOAs, services are orchestrated in order to realize a business process, each service executing a particular function. From the perspective of business operations, however, the business process is realized by composing a number of business activities. Our approach is a systematic method of realizing a monitoring infrastructure for observing compliance in a process-driven SOA. Compliance of a business process is determined by monitoring controls applied to the process' business activities. In our approach, we refer to organizations that already have in place controls and/or a method of defining and documenting these controls for compliance. We illustrate an overview of our approach in Fig. 5.

Essentially, we propose an MDD approach to developing a business process compliance monitoring component. Our aim is to realize an event-based monitoring component from these requirements. We provide a DSL with which a developer can specify compliance monitoring directives.[5] The developer draws these directives from the documentation of controls and discussions with compliance domain experts. The developer is able to specify controls, activities to which they are applied, and data from a particular activity required to verify its control. Typically, each compliance monitoring directive would comprise an event or pattern of events representing business activities, from which compliance control data are extracted.

In addition to the DSL, we define code generation templates, which generate the compliance monitoring component based on directives specified with the DSL. We demonstrate our approach through an MDD framework. Within this framework, we implement a DSL for specifying the compliance directives, as well as the necessary code generation templates for generating the compliance monitoring components.

3.2 Process monitoring domain

One of the first steps in developing a DSL is to analyze the domain for which the DSL is being developed [22]. The domain model in Fig. 6 captures the main concepts of the business process monitoring domain. The model is illustrated with UML class diagram notation.

During execution of a business process, the activities executed as well as the order in which they occur depend on actual data in the system.

A process monitor is typically configured to observe occurrence of certain conditions within a specific subset of the entire process execution. We capture this idea of a subset of the process in the form of a `Process Instance Fragment` class. Comprising each `Process Instance Fragment` is a number of `Activity Instance`'s. Each `Activity Instance` represents the occurrence of either an `Atomic Activity Instance`, that is, a single activity, or a group of activities (`Activity Group Instance`). Moreover, each `Activity Instance` is associated with specific data elements, for example, activity name, whose actual values may differ depending on the process instance at execution time.

`Activity Instance`'s are also related to an `Activity Execution Order` that specifies the order in which a group of activities occurs. For our domain model,

[5] We use the terms directives and rules interchangeably.

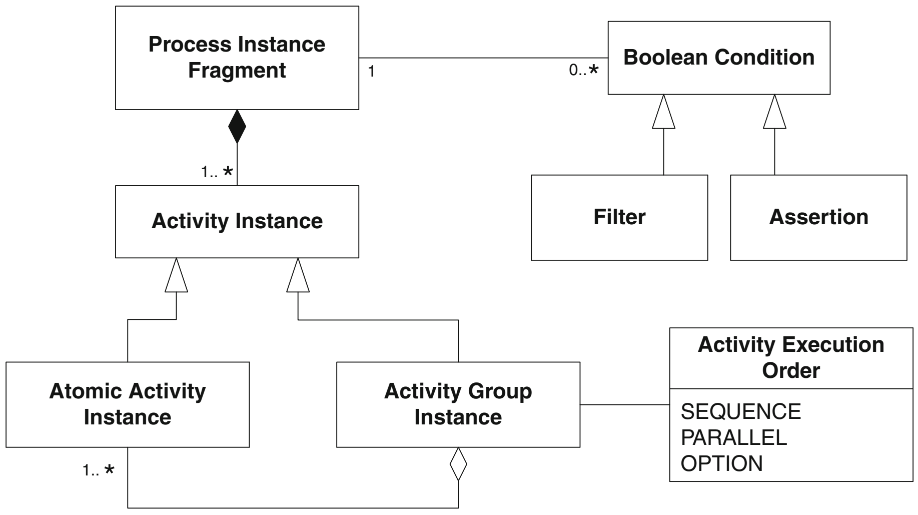


Fig. 6 Process monitoring domain model

we currently define three possible `Activity Execution Order` options, that is, a sequential order (`SEQUENCE`)—activities occur one after the other in a fixed order—a parallel execution order (`PARALLEL`)—activities occur simultaneously, that is, all activities occur but do not have a fixed order—an option execution order (`OPTION`)—activities occur in a mutually exclusive manner, that is, strictly one of the activities shall occur. We have chosen these three options as they can express the most common workflow patterns that occur in business processes. In Table 1, we show the five elementary control flow concepts from the Workflow Management Coalition [23,24] that are found in most business processes. We illustrate how these patterns are represented using the activity execution order options.

Besides `Activity Instance`'s, a `Process Instance Fragment` is associated with a number of `Boolean Condition`'s, which specify constraints on data elements (from `Activity Instance`'s) that are considered for monitoring. They (`Boolean Condition`'s) are grouped into two kinds, `Filter`'s and `Assertion`'s. `Filter`'s specify data categories or conditions that should be ignored or considered in a monitored process fragment. `Assertion`'s on the other hand represent conditions that are expected to be true in the monitored (`Activity Instance`) data.

3.3 Compliance monitoring domain-specific language (DSL)

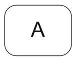
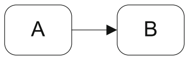
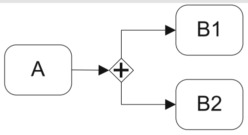
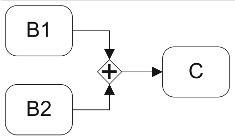
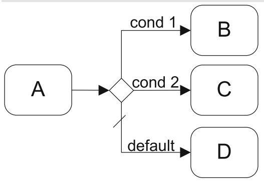
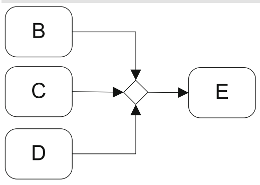
Based on this domain model for the process monitoring domain, we derive the concrete syntax of our DSL for specifying compliance monitoring directives for business processes. We implemented a prototype model-driven process monitoring framework to demonstrate our DSL. The prototype is built on top of an MDD framework (Frag [25]). Note that the concrete syntax takes most of its form from the domain model in the previous section.

To explain our DSL sytax, we use the sample shown in Fig. 7; this is a specification of the SoD process monitor described in Sect. 2.4. As can be seen in the diagram, the DSL typically has two distinguishable sections. The first section is a definition section (can be thought of as declarations), in which activities to be monitored are defined. Here, one can define both `AtomicActivityInstance`'s and `ActivityGroupInstance`'s. `AtomicActivity Instance`'s define single activities, whereas `Activity GroupInstance`'s define a group of activities that may not necessarily be adjacent to each other in the business process definition. The `ActivityGroupInstance` also specifies order of occurrence of activities, which may be sequential, optional, or parallel. In Fig. 7, for instance, the `SEQUENCE` order means the four activities listed are expected to execute in a sequential order.

The second distinguishable section contains process monitoring directives. Here, we specify the process subset that we are interested in monitoring. A process monitoring directive begins with the `ProcessInstanceFragment` keyword. Each directive then has up to three subsections to be defined, activities, filters, and assertions. Under activities, we use the activity definitions from the declaration section. We may also define activities directly under this subsection; however, this limits the possibility to reuse such definitions in multiple compliance rules.

Under the filters subsection, we specify a number of conditions for limiting the type (and consequently amount) of activity instances considered for a particular monitoring directive. For instance, data may only be of interest below a certain threshold value, and so, compliance monitoring components would only observe these data values and ignore the rest. In Fig. 7, the process monitoring directive considers only those `CheckCreditWorthiness` activities with

Table 1 Mapping business process patterns to activity execution order

BPMN	Activity Execution Order
Single Activity – these have a one-to-one mapping with events	
	`SEQUENCE{A}`
Sequence Flow – depicts a very simple event pattern with one business activity following another	
	`SEQUENCE{A,B}`
Parallel Split – indicates that a business activity splits from a single thread of control to multiple threads of control that execute simultaneously. In such a flow, the originating event A occurs and is followed by events B1 and B2. The two following events must occur but do not have a specific order	
	`SEQUENCE{A, PARALLEL {B1,B2}}`
Synchronization – indicates a point in the control flow where two parallel threads of control merge into a single thread. In this case two events B1 and B2 must occur first, although in no particular order, and then followed by the event C	
	`SEQUENCE{PARALLEL{B1,B2}, C}`
Exclusive Choice – represents a point at the control flow where, depending on a certain condition, only one of the available paths in the control flow is chosen. In our mapping, an event A could be followed by at most one of the choices B, C or D. The other events must not occur	
	`SEQUENCE{A,OPTION{B,C,D}}`
Simple Merge – at most one of the events B, C or D occurs (and should not be followed by the other two) and is followed by event E	
	`SEQUENCE{OPTION{B,C,D},E}`

a `loanAmount` less than EUR 1 million and ignores the rest.

The last subsection, assertions, is used to specify expected values in monitored data. Should these expectations not be fulfilled in actual runtime data, a compliance control has been violated. For instance, in our sample when the `loanAmount` value is less than EUR 1 million, it is expected (assertions) that a person with the role `Creditor` executes the first three activities and then another person with the role `PostProcessingClerk` executes the last activity. If persons with different roles than what is expected execute any of these activities, this is reported as a compliance violation.

Fig. 7 DSL specification of SoD compliance monitor

```
# Define/Declare activity instances
AtomicActivityInstance create VerifyBankingPrivileges
AtomicActivityInstance create AcquireBankInformation
AtomicActivityInstance create BuildCustomerLoanFile
AtomicActivityInstance create CheckCreditWorthiness
AtomicActivityInstance create EvalutateLoanRisk

# Loan request initiate activity group
ActivityGroupInstance create LoanRequestInitiate\
            -activityOrder SEQUENCE\
            -activityList [list build\
                        VerifyBankingPrivileges\
                        AcquireBankInformation\
                        BuildCustomerLoanFile\
                        CheckCreditWorthiness]

#  Segregation of duty monitor for normal loans
ProcessInstanceFragment create NormalLoanProcessing        \
            -activityInstances LoanRequestInitiate
            -filters [list build\
                  [BooleanCondition create -expression\
                        "CheckCreditWorthiness.loanAmount < 1000000"]]\
            -assertions [list build\
                  [BooleanCondition create -expression "role = Creditor"\
                        -activities [list build\
                                    VerifyBankingPrivileges\
                                    AcquireBankInformation\
                                    BuildCustomerLoanFile]]\
                  [BooleanCondition create -expression "role = PostProcessingClerk"\
                        -activities CheckCreditWorthiness]]
```

The DSL thus enables a developer to specify compliance monitoring directives for (subsets of) a process instance to which controls have been applied.

3.4 Compliance code transformation and generation

In addition to the DSL, the prototype comprises transformation templates for converting the DSL directives into code for a compliance monitoring component. In this particular implementation, our target platform was an event-based monitoring component—the Esper event processing engine [26]. Therefore, our code transformation-and-generation templates transform DSL compliance monitoring statements into corresponding event processing language (EPL) (cf. Sect. 2.3) queries for the Esper engine.

Such code generation templates are specific to the type of event processing engine or technology that one may be using. They need to be implemented initially, and once done, the technology-specific code can be quickly and repeatedly generated. In Fig. 8, we illustrate a sample code generation template as well as the resulting code that is generated when this template is applied to the SoD compliance rule from Fig. 7.

The `PATTERN` section of the template is where the activity execution order is specified. We map the activity execution order to event patterns using a mapping scheme that we present in Table 2.

The resulting compliance monitoring code provides configuration information to the event processing engine during its initialization; as business processes are executed, the event processing engine observes business activity events to determine compliance of the executing processes.

Note that the EPL is also a DSL specialized for the purpose of defining queries over event-streams and is, therefore, capable of specifying compliance monitoring rules as well. One may wonder why we need the compliance monitoring DSL at all. The compliance monitoring DSL is used to express compliance control monitoring directives in terms that are in the process-driven SOA domain. This design serves three purposes; first, the developer works with process-driven SOA domain concepts and, therefore, is not continuously translating SOA domain concepts into EPL statements. Secondly, the code specified under the DSL can be reused for different technology implementations. Incase there was a different CEP engine to be used, the same compliance code could be used with only the code generation templates requiring changing. And lastly, the developer is better able to communicate and share implementation source code with the (business process or compliance control) domain experts, bringing them closer to the implementation. Of course, this implies that a system developer using the DSL is familiar with the basic concepts of process-driven SOAs and perhaps has some experience implementing them.

Fig. 8 EPL code generation template

```
# Main Template for EP Rule

<~ self applyForeach processFragment [$processFragments get fragments] {
 SELECT <~ $processFragment generateDataElements [self] ~>
 FROM pattern [
                      <~ $processFragment generateProcessActivities [self] ~>
 ].win:length(<~ $processFragment activityCount ~>
WHERE <~ $processFragment generateAssertions [self] ~>
}~>
```

↓

```
NormalLoanProcessing =
SELECT ver.role, che.role, che.loanAmount
FROM pattern [ every ver=VerifyBankingPrivileges ->
                      acq=AcquireBankInformation(processId=ver.processId) ->
                      bui=BuildCustomerLoanFile(processId=ver.processId) ->
                      che=CheckCreditWorthiness(processId=ver.processId,
                                                         loanAmount < 1000000) ].win:length(4)
WHERE (ver.role != 'CreditBroker' or
                      acq.role != 'CreditBroker' or
                      bui.role != 'CreditBroker') or
                      (che.role != 'PostProcessingClerk')
```

Table 2 Transforming activity execution order to event patterns

Activity execution order	Event detection patterns
Single activity	
`SEQUENCE{A}`	(every A)
Sequence flow	
`SEQUENCE{A,B}`	(every A) → B
Parallel split	
`SEQUENCE{A, PARALLEL {B1,B2}}`	(every A) → (B1 and B2)
Synchronization	
`SEQUENCE{PARALLEL{B1,B2}, C}`	(every (B1 and B2)) → C
Exclusive choice	
`SEQUENCE{A,OPTION{B,C,D}}`	(every A) → (B or C or D)
Simple merge	
`SEQUENCE{OPTION{B,C,D},E}`	(every (B or C or D)) → E

In the next section, we present an evaluation of our approach.

4 Evaluation of compliance monitoring approach

One of the challenges with respect to monitoring of compliance is the ad hoc implementation of compliance controls, in most cases using niche products to implement a particular compliance requirement. CEP can today be considered as one such niche product in compliance monitoring for business processes (a technique known as business activity monitoring (BAM) [16]). In this section, we evaluate our approach based on the illustrative scenarios presented in Sect. 2.4. We present a quantitative comparison of the DSL presented in Sect. 3 alongside an EPL for realizing CEP-based monitoring; this comparison serves to highlight the differences between our DSL and the EPL and to initiate a discussion concerning the pros and cons of our proposed approach.

4.1 Evaluation method

We applied the goal–question–metric technique [27] to define appropriate metrics for comparison of our DSLs. We selected two main issues to compare in the DSLs; *effort*—how much of the developer's resources (e.g., time) are required to maintain compliance specifications—and *productivity*—how productive is a developer who is using the DSLs for compliance specifications.

To measure these qualities, we chose two size metrics to compare programs specified by both DSLs. The size metrics we utilize are number of variables (NV) and number of operations (NO). *Variables* refer to identifiers that indicate data to be monitored for a particular compliance rule; *Operations* are considered in the traditional sense, that is, an action or procedure involving operands/variables. Size metrics are considered as relatively good predictors of maintenance effort even though they are not the sole predictor [28]. Considered from this angle, NV and NO both provide indicators of effort and productivity of programmers using our compliance specification DSLs. Once we decided on the metrics, we applied them to the compliance rules from our illustrative scenarios. We applied the metrics to both the EPL and DSL rules.

Table 3 Comparison of compliance program sizes for EPL versus DSL

	EPL		DSL	
	NV *(NVu)*	NO	NV *(NVu)*	NO
LA Rule 1	23 *(16)*	18	10 *(4)*	3
LA Rule 2	17 *(11)*	9	6 *(6)*	2
LA Rule 3	19 *(13)*	10	6 *(5)*	1
LA Reused	–	–	6 *(6)*	0
TB Rule 1	21 *(14)*	16	10 *(10)*	3
TB Reused	–	–	5 *(5)*	0
CH Rule 1	13 *(12)*	10	7 *(6)*	1
Totals	93 (66)	63	50 (42)	10

4.2 Evaluation results

In Table 3, we present the results from applying the NV and NO metrics to our set of compliance rules based on the scenarios from Sect. 2.4. Note that in the NV–column, we have a value *NVu* in brackets. The NVu count represents the number of *unique* variables in a rule. We perform this count as well to provide further analysis and/or comparisons.

From the results presented in the table, we observe a number of things; when we compare the NV for the EPL and the DSL, the EPL in most cases has double the value of NV as compared to the DSL. The EPL is a verbose language and does not provide possibilities for reusing code. The DSL on the other hand enables extraction of some common business process patterns and possibility to reuse them in multiple rules. We include a count of NV from reused rules in the bottom row of the table (EPL does not have reuse features). This reuse drastically reduces the NV that are needed by the DSL for each rule.

The difference in the NO count between EPL and DSL is even greater, with the EPL count almost six times that of the DSL on average. The EPL is a more generic language and, therefore, uses a greater number of operators / operations to realize certain monitoring requirements. We can see examples of this in Fig. 8, where the EPL incorporates Boolean operations (`or`), equality operators (`=`, `!=`), and timing operators (`->`). Since our DSL is more specialized, a lot of operations are implicitly captured in a declarative textual language form that is closely related to the domain. Moreover, lots of default values are assumed during code generation, which is not the case with the EPL. As an example (see Fig. 7), instead of using multiple followed by (`->`) operators, we express this with the declarative command `SEQUENCE`, to indicate that all activities listed occur in a sequence, following the order in which they are listed.

Overall, we would like to argue that the characteristics of our DSL syntax are in tune with the mental working style or approach of a programmer, that is, through referencing short-term memory for tasks at hand and long-term memory to make broader connections [29]. With fewer variables and operations to consider, the programmer is better able to quickly understand and maintain the source code. The reuse feature also fits within this model, described in Henderson-Sellers [30] as chunking and tracing, that is, the programmer typically chunks together related pieces of information and mentally refers to it from one point of view. This is very much what the DSL provides with the reuse feature.

4.3 Limitations and threats to validity

There are a number of issues that we consider to be possible limitations and/or threats to the validity of the results that we obtained during our experiments. We consider these from two angles, that is, threats related to the data that we used in the experiment, and threats related to the method we followed to carry out the experiment.

When we refer to data, we consider the compliance rules as well as their implementation. We have a number of compliance rules in our experiment; however, it is always a question whether there is enough of them to make more general conclusions. The number of rules, therefore, poses a threat to the generalizability of our proposed approach. In addition, one could argue that the implementation of the compliance rules in the two different languages is also a threat, because there is the possibility of bias, in favor of the DSL, in the implementations. We minimized the effect of this threat by reviewing the compliance rules in both DSLs to eliminate any unnecessary code.

Regarding the method we followed in carrying out the experiment, the first threat we can identify is that of the choice of the metric. While Li and Henry clearly state that size metrics are predictors of maintenance effort, they also point out that these metrics are not the sole predictors [28], that is, they should be considered in conjunction with other metrics. Oman and Hagemeister [31] actually propose combining a set of metrics (considered to have influence on maintenance effort) into a single index of maintainability. In a more ideal situation, we could consider multiple metrics for evaluation. We were, however, limited by the choice of a metric that could provide an analysis of both DSLs side by side. Moreover, most metrics we came across are designed with procedural and/or object-oriented languages in mind. We did not come across metrics that have been explicitly designed with a focus on declarative languages of the kind that we analyzed in this work.

5 Related work

In a very general sense, all forms of monitoring are related to compliance of some sort. Monitoring is concerned with observing the state of a system to ensure that it fulfills a particular goal and to inform an interested party when things go wrong. With regard to our work, the purpose of monitoring is to ensure that our business processes actually execute compliance controls as expected at runtime. Whereas the compliance controls are built into the system at design and implementation time, monitoring provides the required runtime validation and ensures that compliance of business processes is auditable [3]. In this section, we discuss a number of other related works aimed at ensuring or monitoring for compliance at runtime.

5.1 Compliance monitoring approaches

The works by Mahbub and Spanoudakis [32], Giblin et al. [33,34], Sadiq et al. [6], and Rozinat and van der Aalst [35] are directly related to monitoring and checking for compliance in business processes. Sadiq et al. [6] propose an approach for incorporating compliance at design time. They propose a method for incorporating compliance controls into a business process. Their approach involves defining compliance controls in Formal Constraint Language and then incorporating this into process models through annotations. They aim to support process designers to effectively incorporate compliance controls into business processes. This complements our approach in the sense that processes for which our approach creates runtime monitoring infrastructure are designed with such compliance controls embedded. However, the runtime monitoring still offers validation and ensures that processes are auditable, which are important issues in compliance.

In Mahbub and Spanoudakis [32], the authors propose an approach similar to ours in that they provide a mapping from business-level information to monitoring patterns. In their work, however, they map from BPEL4WS to event calculus. The differences in mapping between the two approaches may have some implications on how they are converted into a running monitor. Expressing the patterns using event calculus informs of the changes that happen in the system through time, whereas expressing the patterns using event algebra as we do simply states what are the operations performed on the events.

The approach in Giblin et al. [33,34] aims for runtime monitoring for compliance. The authors propose REALM, a meta-model for the specification of compliance regulations in a technology-agnostic manner. A REALM model consists of a concept model that captures concepts and relationships of the domain in which regulations are being applied, a compliance rule set that represents regulatory requirements in a real-time temporal object logic, and meta-data providing information about the regulations, for example, source and enactment dates. They propose a framework that uses the REALM specification to generate technology-specific correlation rules for runtime monitoring. Rather than attempt to capture compliance rules, our approach assumes that compliance controls are already defined within a business process, for example, using methods like in Sadiq et al. [6]. However, our approach also provides the mapping from the modeled business process to a set of queries that are used for correlation and monitoring of events.

Rozinat and van der Aalst [35] present an approach for conformance checking of business processes. While the previously discussed approaches target design time and runtime compliance checking, this approach is more retrospective with respect to execution time, that is, checking for conformance after business processes have already been executed. The authors mine event logs and process models to check conformance from two angles, the fitness, that is, how close do the actual execution logs match with the process model, and the appropriateness, that is, how well does the model describe the actual process execution recovered in the execution logs. This conformance check (specifically the fitness conformance) is similar to how we approach compliance checking; however, we do not consider an *entire* process in our checks. Instead, we consider a subset/excerpt of activities that are specifically in a process for the purpose of checking compliance. The authors, however, do raise an interesting qualitative issue concerning the appropriateness of models. In our mappings from business activities to event detection patterns, we provide all possible combinations. In cases where there would be multiple parallel activities, the possible combinations from the mappings would be exponential, and yet not all possible combinations shall exist in the actual execution of the process.

5.2 Generic business process monitoring approaches

Regarding more generic approaches for monitoring business processes and SOA, that is, not specifically monitoring as applied to compliance checking, we consider some approaches that incorporate monitoring logic into business processes ([13,15,36–38]), other leverage CEP ([14,39]), and finally business protocol monitoring ([40,41]). Baresi et al. [36] present an approach to monitor service compositions, that is, business processes. They embed annotations into BPEL code, which annotations are later transformed by a preprocessor into BPEL statements. In addition, they implement a monitoring Web service. This Web service is a gateway to an external server component that monitors a process execution. While this approach has the benefit of sticking to the standard services paradigm, the nature of Web services, as the authors also recognize, is to be stateless. This implies that it is not possible for the monitoring Web service to maintain state while monitoring an entire process; yet, state is required for monitoring a service composition over a period of time. We encountered similar issues in our implementation of the monitoring infrastructure and instead opted to use a messaging queue as a gateway rather than a Web service.

In both approaches by Erradi et al. [38] and Baresi et al. [37], the monitoring logic is also embedded into the process control flow in a similar manner to what is done in Baresi et al. [36]. However, these two approaches are extensions to the WS-Policy specification. Erradi et al. [38] develop Manageable and Adaptive Service Compositions (MASC), a policy-based middleware that executes WS-Policy4MASC assertions. WS-Policy4MASC is the authors' extension to the WS-Policy Framework—the extension enables incorporating new monitoring and control policy assertions. Baresi et al. [37] propose Web Service Constraint Language (WS-CoL), a domain-independent language for expressing monitoring policies for WS-BPEL processes. WS-CoL is compliant with the WS-Policy specification. Their approach proposes weaving monitoring directives into a WS-BPEL specification such that calls to a monitoring manager are attached to parts of the specification. This weaving is done at deployment time to keep a separation between the WS-BPEL code and the WS-CoL monitoring constraints. Whenever these constraints are encountered (at runtime), the monitoring manager performs constraint checking and then calls the relevant service. In both these approaches, the monitoring logic is tightly coupled with the actual running system.

Muehlen and Rosemann [13] present a process monitoring approach that aims to not only monitor the process execution but also the economic impact of processes on a business. This approach achieves this by taking three views of process monitoring, the process view, resource view, and objects view. They present an architecture for a process monitoring and control system. The essential component for the monitoring is an evaluation method library that contains algorithms for performing the calculations based on monitored data. The library is extensible with the possibility for one to plug in customized evaluation algorithms.

The work from Grigori et al. [15] proposes an integrated business process intelligence toolsuite for managing process execution quality. One of the components of the tool is the business process cockpit (BPC) [42] that offers an interface for business users to monitor different perspectives (e.g., services, resources) of a business processes as the process executes. The BPC monitoring achieves this by periodically reading data from audit logs from the integrated toolsuite. A user needs to configure *alerts* that determine information of interest for monitoring. The BPC component alerts the user or can execute other actions. The alerts are similar to our the query rules that we use to configure our event processing engine to monitor for events. However, event processing provides continuous monitoring rather than the periodic style of the BPC. In addition, it is not clear whether it is possible for the alerts to define queries similar to ours in the sense of mapping patterns of events to a series of service invocations, for example.

5.3 CEP-based monitoring approaches

Although not explicitly stated, the work from McGregor et al. [14] provides a form of CEP monitor for QoS compliance. The monitoring solution provides a logging service interface, to which data from the service being monitored are passed. This data are forwarded to an internal data constructor agent that computes summary data based on previously logged state information for the same service. This summary data may then be checked against an original definition of the Web service capabilities to identify any discrepancies. They do not state in their approach whether it is possible to configure the agents that perform processing on the state information concerning the Web services. In our case, queries are used for such configuration and can be easily changed and updated depending on what kind of monitoring is required. Vaculín and Sycara [39] also propose CEP for semantic monitoring of Web services. They achieve this by extending an event algebra to enable specification of composite events. These concepts are similar to low-level service invocation events and high-level business events. However, they focus on monitoring for semantic Web services. They do not filter events based on syntax and parameters; rather they define an event ontology, and whenever a primitive event is fired, it is actually an instance of the ontology class representing the event type on which semantic filtering is performed.

6 Discussion

We propose a systematic approach to map the compliance controls (activities or subprocesses) that are defined in business processes into monitoring queries that can check for this compliance at runtime. We now discuss some of the advantages and limitations of this approach.

When applying our approach, we think there is a clear separation of issues concerning the system functionality from those concerning compliance assurance. As a result, maintenance overheads that are incurred without this clear separation are reduced. This separation characteristic is also present in related works (Sect. 5). We assume that business specialists have already incorporated their choices on compliance controls into the business process designs, and we make a mapping of these choices to the technical implementation of monitoring logic. Other works like Giblin et al. [33,34] tackle expressing these compliance controls (as compliance rule sets [34]) and then mapping them to the technical implementation.

Organizations have to continuously adapt their processes and systems to match ever-changing compliance requirements. Our approach provides a clear change strategy: Whenever a compliant business process is changed, the change impact affects the activity execution order, which in turn affect a set of CEP rules. Hence, the explicit trace links in our approach foster understandability, changeability, and maintainability of our event-based compliance solutions.

The identification of common activity execution order patterns might foster reuse of compliance rules. Whenever another business process can be mapped to the same sequence of technical events, we can identify the same business events. Hence, even if the business process activities are not the same but can be mapped to the same event trail, reuse of existing compliance rules is possible.

We propose to emit service invocation events to monitor business processes. These events have a standard format and only change the values of parameters depending on the service invoked. In an organization with many services (as is the case in large-scale SOAs), we are able to reuse the event emitting code across services. The approach proposed by Giblin et al. [34] expresses compliance regulations in a technology-agnostic manner and finally generates the technology-specific correlation rules. This improves reuse in situations where the runtime monitoring technology might change.

Although our focus is on monitoring process-driven SOAs, we feel that our approach is usable even in a situation where there exists ad hoc implementations of compliance regulations. This is due to the fact that the monitoring is event-based, and so, regardless of the technology in use, if it is possible to emit events, one only has to ensure that the format of the event is appropriate. This is easily done using wrappers or transforming adaptors. Therefore, it is already possible to use this approach for monitoring during compliance even during a transition from ad hoc to more systematic methods of implementing compliance controls.

The compliance detection rules and queries used in CEP tools are, in most cases, written in simple query languages. They are thus relatively easy to understand by technical personnel. Erradi et al. [38] and Baresi et al. [37] use WS-Policy-based languages that are expressed in XML—making these languages readable and perhaps even providing opportunities for automated processing and transformations. The other monitoring approaches use more complicated expression languages. Giblin et al.[33,34] use a compliance rule set based on temporal object logic, while Vaculín and Sycara [39] uses an event algebra to aggregate events.

We provide a mapping for control flows to event detection patterns. We feel that such a mapping provides a good basis for automating the process of realizing the compliance monitoring infrastructure. However, such a mapping of BPMN control flows to executable diagrams applies to private (internal) business processes. Our approach is, therefore, limited to realizing the monitoring infrastructure for internal business processes. There might be some challenges applying it in a setting with cross-organizational business processes, stemming from the fact that business process execution (event) data are generally not shared across organizations. The organizations only externalize a predefined interface to partner organizations. Therefore, without access to the event data from within an organization, we are not able to express the event patterns and consequently are unable to generate a monitoring solution.

7 Summary and conclusion

Compliance with regulations, laws, and policies is a requirement for organizations to avoid negative consequences. These organizations thus have to monitor their information systems to ensure that they still adhere to these compliance concerns. Considering that many organizations today implement their systems based on process-driven SOAs, we are proposing an approach for monitoring business processes for compliance in such process-driven SOAs.

We propose a structured approach for realizing compliance monitoring components for business processes. We assume that compliance controls are implemented as business activities or groups of business activities, and in effect, we monitor for the execution of these compliance controls. We evaluate our approach through a number of scenarios in order to determine the impact of our approach and tool support on the productivity and effort a developer puts into specifying compliance monitoring directives.

Acknowledgments This work was supported by funds from the European Commission (contract No. 215175 for the FP7-ICT-2007-1 project COMPAS).

References

1. Zdun U, Hentrich C, Dustdar S (2007) Modeling process-driven and service-oriented architectures using patterns and pattern primitives. ACM Trans Web 1(3):14
2. Kung P, Hagen C, Rodel M, Seifert S (2005) Business process monitoring & measurement in a large bank: challenges and selected approaches. In: Proceedings of the 16th international workshop on database and expert systems applications, pp 955–961
3. Cannon JC, Byers M (2006) Compliance deconstructed. Queue 4(7):30–37
4. Anderson R (2008) Security engineering. Wiley, New York
5. O'Grady S (2004) SOA meets compliance: compliance oriented architecture. http://redmonk.com/public/COA_final.pdf. Accessed April 2010
6. Sadiq SW, Governatori G, Namiri K (2007) Modeling control objectives for business process compliance. In: Alonso G, Dadam P, Rosemann M (eds) BPM. Lecture notes in computer science, vol 4714. Springer, Berlin, pp 149–164
7. Bonazzi R, Hussami L, Pigneur Y (2010) Compliance management is becoming a major issue in is design. In: D'Atri A, Saccà D

(eds) Information systems: people, organizations, institutions, and technologies. Physica-Verlag, Heidelberg, pp 391–398
8. Mulo E, Zdun U, Dustdar S (2009) Monitoring web service event trails for business compliance. In: SOCA, IEEE, pp 1–8
9. Stoneburner G, Goguen A, Feringa A (2002) National institute of standards and technology special publications 800–30: risk management guide for information technology Systems
10. IT Governance Institute (ITGI) (2006) IT control objectives for Sarbanes-Oxley. 2nd edn. Information Systems Audit and Control Association (ISACA) Inc
11. Havey M (2005) Essential business process modeling. O'Reilly Media, Inc., USA
12. Object Management Group/Business Process Management Initiative (2008) Business process modeling notation (bpmn) version 1.0
13. Zur Muehlen M, Rosemann M (2000) Workflow-based process monitoring and controlling-technical and organizational issues. In: Proceedings of the 33rd annual hawaii international conference on system, sciences, vol 2, pp 10
14. McGregor C, Kumaran S (2002) Business process monitoring using web services in B2B e-commerce. In: Proceedings of the international parallel and distributed processing symposium (IPDPS 2002), pp 219–226
15. Grigori D, Casati F, Castellanos M, Dayal U, Sayal M, Shan MC (2004) Business process intelligence. Comput Ind 53(3):321–343
16. Luckham DC (2002) The power of events: an introduction to complex event processing in distributed enterprise systems. Addison-Wesley, Reading
17. Greiner T, Düster W, Pouatcha F, von Ammon R, Brandl HM, Guschakowski D (2006) 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 (PPPJ '06), ACM, New York, pp 237–242
18. Rozsnyai S, Vecera R, Schiefer J, Schatten A (2007) 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 (CEC/EEE 2007), pp 409–420
19. Wei M, Ari I, Li J, Dekhil M (2007) ReCEPtor: sensing complex events in data streams for service-oriented architectures. Technical report HPL-2007-176, HP Labs
20. Brandl HM (2007) Complex event processing in the context of business activity monitoring. University of Applied Sciences Regensburg, Master's thesis
21. Wu E, Diao Y, Rizvi S (2006) High-performance complex event processing over streams. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD '06), ACM, New York, pp 407–418
22. Völter M (2009) Md* best practices. J Object Technol 8(6):79–102
23. Workflow Management Coalition Specification (1999) Workflow management coalition terminology & glossary (Document No. WFMC-TC-1011). Workflow Management Coalition Specification
24. Wohed P, van der Aalst WMP, Dumas M, ter Hofstede AHM, Russell N (2006) On the suitability of bpmn for business process modelling. In: Dustdar S, Fiadeiro JL, Sheth AP (eds) Business process management. Lecture notes in computer science, vol 4102. Springer, Berlin, pp 161–176
25. Zdun U (2010) A DSL toolkit for deferring architectural decisions in DSL-based software design. Inf Softw Technol 52(7):733–748
26. EsperTech (2009) Esper reference documentation version 3.2.0. EsperTech Inc
27. Basili V, Caldiera G, Rombach H (1994) The goal question metric approach. Encycl Softw Eng 1:528–532
28. Li W, Henry SM (1993) Object-oriented metrics that predict maintainability. J Syst Softw 23(2):111–122
29. Hatton L (1998) Does oo sync with how we think? IEEE Softw 15(3):46–54
30. Henderson-Sellers B (1996) Object-oriented metrics: measures of complexity. Prentice Hall object-oriented series, Prentice Hall PTR,
31. Oman P, Hagemeister J (1992) Metrics for assessing a software system's maintainability. In: Proceerdings of the 1992, conference on software maintenance, pp 337–344
32. Mahbub K, Spanoudakis G (2004) A framework for requirements monitoring of service based systems. In: Aiello M, Aoyama M, Curbera F, Papazoglou MP (eds) ACM, ICSOC, pp 84–93
33. Giblin C, Liu AY, Zhou X (2005) Regulations expressed as logical models (REALM). In: A.I.O.S. Press (ed) Proceedings of the 18th annual conference on legal knowledge and information systems (JURIX '05), pp 37–48
34. Giblin C, Müller S, Pfitzmann B (2006) From regulatory policies to event monitoring rules: towards model-driven compliance automation. Technical report RZ 3662, IBM Research
35. Rozinat A, van der Aalst WMP (2008) Conformance checking of processes based on monitoring real behavior. Inf Syst 33(1):64–95
36. Baresi L, Ghezzi C, Guinea S (2004) Smart monitors for composed services. In: Proceedings of the 2nd international conference on Service oriented computing (ICSOC '04), ACM, New York, 193–202
37. Baresi L, Guinea S, Plebani P (2006) Lecture notes in computer science. In: WS-policy for service monitoring. Springer, Berlin, pp. 72–83
38. Erradi A, Maheshwari P, Tosic V (2007) WS-policy based monitoring of composite web services. In: 5th IEEE European conference on web services (ECOWS '07), pp 99–108
39. Vaculin R, Sycara K (2007) Specifying and monitoring composite events for semantic web services. In: 5th IEEE European conference on web services (ECOWS '07), pp 87–96
40. Li Z, Jin Y, Han J (2006) A runtime monitoring and validation framework for web service interactions. In: The Australian software engineering conference (ASWEC '06), pp 70–79
41. Benatallah B, Casati F, Toumani F (2004) Analysis and management of web service protocols. In: Proceedings of the 23rd international conference on conceptual modeling (ER '04), Shanghai, pp 524–541
42. Sayal M, Casati F, Dayal U, Shan MC (2002) Business process cockpit. In: VLDB, Morgan Kaufmann, pp 880–883