An Anti-Pattern-based Runtime Business Process Compliance Monitoring Framework

Ahmed Barnawi
King Abdulaziz University
Jeddah, Saudi Arabia

Ahmed Awad
Cairo University
Cairo, Egypt

Amal Elgammal
Cairo University
Cairo, Egypt

Radwa Elshawi
Princess Nourah Bint Abdulrahman University
Riyadh, Saudi Arabia

Abduallah Almalaise
King Abdulaziz University
Jeddah, Saudi Arabia

Sherif Sakr
King Saud bin Abdulaziz University for Health Sciences
Riyadh, Saudi Arabia
University of New South Wales, Sydney, Australia

Abstract—**Today′s dynamically changing business and compliance environment demand enterprises to continuously ensure their compliance with various laws, regulations and standards. Several business studies have concluded that compliance management is one of the main challenges companies face nowadays. Runtime compliance monitoring is of utmost importance for compliance assurance as during the prior design-time compliance checking phase, only a subset of the imposed compliance requirements can be statically checked due to the absence of required variable instantiation and contextual information. Furthermore, the fact that a BP model has been statically checked for compliance during design-time does not guarantee that the corresponding running BP instances are usually compliant due to human and machine errors. In this paper, we present a *generic proactive runtime* BP compliance monitoring framework, *BP-MaaS*. The framework incorporates a wide range of expressive high-level graphical *compliance patterns* for the abstract specification of runtime constraints. Compliance monitoring is achieved using *anti-patterns*, a novel mechanism which is agnostic towards any underlying monitoring execution technology. As a proof-of-concept, complex event processing (CEP) technology is adopted as one of the possible realizations of the monitoring engine of the framework. An integrated tool-suite has been developed as an instantiation artifact of BP-MaaS, and the validation of the approach is undertaken in several directions, which includes internal validity and case study conducts considering two real-life case studies from the banking domain.**

Keywords—Business Process Management; Business Process Monitoring; Business Process Compliance

I. Introduction

The global regulatory environment has grown in complexity and scope since the financial crisis in 2008. This is causing significant problems for organizations in almost all industrial sectors, as the complexities of hard and soft regulations are little understood or appreciated [6]. For example, banking regulations such as anti-Money Laundering Directives are generally complex and far-reaching, with a raft of major banks found to be not in compliance in 2012. Standard Chartered Bank, London, for example, was fined a total of $459 million in 2012[1]. Worse still, HSBC Holdings Plc. had paid a record of $1.92 billion. These incidents were preceded by scandals and business failures such as Enron, and WorldCom back in 2001. Subsequently, much attention has been paid to compliance management from both the academic and the industrial communities.

Research on compliance management has focused on the checking and enforcement of compliance in one of the BP lifecycle phases; i.e. design-time verification (e.g., [4, 22]), runtime monitoring (e.g., [34, 57]) and offline monitoring (e.g., [1]). While each of these phases has its strengths and limitations, we consider *proactive* runtime compliance monitoring as a vital component. With *proactive*, we mean violations are detected as soon as they occur, and appropriate recovery action(s) are taken to mitigate/minimize their impacts. Only a subset of compliance rules can be checked during design-time due to the lack of necessary contextual information; e.g., time span constraints between tasks can not be checked at design-time. Besides, due to human and machine errors, a violation might still occur in a running instance resulting from a statically compliant BP model. Offline monitoring, on the other hand, is more beneficial for statistical analysis and diagnostic purposes, as it is usually performed on a large number of completed executions, following a retrospective (after-the-fact) approach.

Driven by this emergent *business need*, runtime compliance monitoring has recently been an active research topic (e.g., [38, 40, 64]); however despite the efforts, important aspects of compliance monitoring have been overlooked. For instance, the technology used to enable monitoring, the nature of the process execution environment, the type of events generated from these environments and their granularity have not been well discussed in literature. Moreover, such approaches are introduced as a component of a larger system that is usually tightly interwoven with pre-existing components.

In this paper, we described the design and the implementation details of an end-to-end *runtime* business process compliance monitoring as a service framework, *BP-MaaS*, where we aim at providing an independent and comprehensive platform as a compliance monitoring service [2]. In particular, we summarize the main contributions and strengths of our

[1]http://www.accuity.com/industry-updates/free-resources/trends-in-aml-compliance-infographic/

framework as follows:

- The framework supports a rich and wide set of compliance patterns that represent the abstract specification of monitoring requirements and cover the four structural facets of BPs; i.e. control-flow, data, employed resources and timing constraints.
- The monitoring engine of our framework is based on the concept of *anti-patterns*, a novel compliance evaluation mechanism which is agnostic towards the used technology for implementing the compliance monitoring engine.
- In order to ease the task of process designers, our framework is equipped with a user-friendly modeling environment that relies on using graphical notations for modeling the compliance rules.
- The framework is equipped with a Compliance Management Knowledge base (CMKB) that incorporates and integrates a set of ontologies to capture and homogenize the different perspectives of the compliance and business sphere.
- We present the details of a proof-of-concept implementation as an instantiation artifact for the monitoring engine of our framework, which adopts complex event processing (CEP) technology. In addition, our modeling environment has been equipped with a plugin that automatically maps the modeled compliance rules into the concrete utilized queries/scripts for executing the runtime monitoring process.
- We demonstrate the applicability and utility of the proposed framework by employing two real-world case studies that deal with business processes of companies operating in the banking domain. The findings have shown that the proposed framework supports a major subset of real-life compliance requirements addressing the heavily regulated banking domain.

In the following, Section II provides the necessary background for the paper. Section III introduces two case studies which are used for serving the purpose of our examples in the rest of the paper. In addition, these case studies are used for evaluating the applicability of the proposed approach that validates its usefulness and utility, and highlight its limitations. Section IV gives an overview of the proposed framework. Section V details the anti-patterns that back the monitoring engine of our framework. The details of the PoC implementation of our framework is presented in Section VI. Case studies-based evaluation of the proposed framework is presented in Section . Related work is highlighted in Section VIII, with a comparative analysis that aligns and appraises our approach against prominent related work efforts. Finally, conclusions and future work directions are drawn in Section IX.

II. Background

This section introduces the main concepts and techniques that form the groundwork for our approach. In addition, we introduce a case study that forms the basis of our discussion and examples in the rest of the paper.

A. *Reference Life Cycle Models*

As the objective of this work is to enable runtime monitoring of process compliance, we depend on events generated

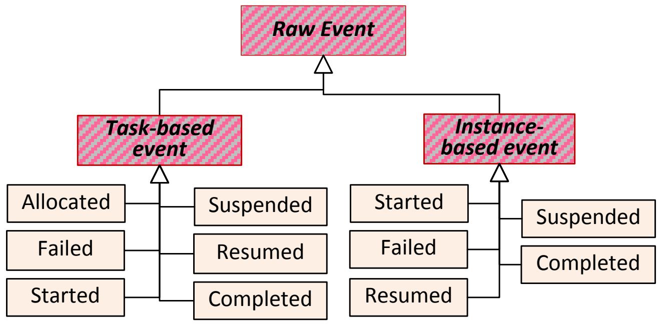


Fig. 1: Classification of raw events

by the execution environment. Events represent the evolution of the running process instances. We assume that these events reflect a change in state of either the whole process instance or one of its task instances. The set of states and transitions among them is assumed to be predefined by a process/task lifecycle. This is important as the events generated by the execution environment will form the input of raw events to the monitoring component upon which violation will be detected. We build on top of the work of Russell et al. [50] and Lerner et al. [30] where we combine both works and project the relevant parts for compliance monitoring.

Figure 1 summarizes the different types of events that are expected to be generated by the execution environment. Definition 2.1 formalizes what a *raw* event means in the context of this paper.

Definition 2.1 (Raw Event): Let PM be the set of all process models, PI be the set of all process instances and TI be the set of all task instances, and R be the set of all resources. A raw event is a tuple (*type, task_instance, process_instance, timestamp, data, resource*), where:

- type $\in \{started, failed, allocated, resumed, suspended, completed\}$ to indicate the actual type of the event,
- $task\_instance \in TI \cup \{\perp\}$ is a reference to the task instance for which the event occurred. When this property is not applicable, e.g. this is a process level event, the property has the value $\perp$,
- $process\_instance \in PI$ is a reference to the process instance within which the event occurred,
- $timestamp \in \mathbb{N}$ indicates the time stamp where the event occurred.
- $data$ is the data payload of the event. This basically holds states of data elements processed within a case but can be extended by execution environment related data. When this property is not applicable or irrelevant in context, the property has the value $\perp$.
- $resource \in R \cup \{\perp\}$ is a reference to the human resource performing a task. When this property is not applicable, e.g., this is an automated step, this property has the value $\perp$.

The set RE defines the different raw events that can be received on the stream. Instead of representing the occurrence of start event of task A within a certain process instance i as $e = (started, A, i, t, d, r)$ where t is the event time stamp, d is the data payload and r is the human resource, we write it as $e = started(A, i, t, d, r)$. Also, we might have a shorthand as $e = started(A, i)$ when the other pieces of information are not relevant.

B. Compliance Patterns

In general, pattern-based modeling of compliance rules is well accepted in the community and several studies have provided a comprehensive set of patterns that cover the different aspects as control flow, data flow, resource allocation and timing as summarized by Ly et al. [32]. In this work, we build on top of those patterns. In particular, Figure 2 summarizes the set of compliance patterns which are supported by our framework. We use *pattern* and *rule* interchangeably where a rule is an instantiation of a pattern.

In practice, any pattern can optionally be limited to a scope in which the rule is required to hold. The scope represents a time window that is bounded by case or task instance-related events. In principle, the default scope is the whole process instance execution. Also, the pattern can be refined by a condition where the rule is required to hold only when this condition is true. The condition may refer to process execution data that are reflected in the event data payload. With each pattern, two actions are defined. The *Violation Action* describes the action taken by the monitoring component when the violation occurs whereas the *Prediction Action* describes the action taken when there is a possibility of violation. The nature of the action depends on how the monitoring component is integrated with the execution environment. For instance, the simplest action that can be taken is to alert administrators.

Definition 2.2 (Atomic Compliance Rule): Let PM be the set of all process models to be monitored. A compliance rule is a tuple $(pattern, model, antecedent, consequent, condition, scope\ start, scope\ end, multiplicity, WA, timespan, alerttimespan, isBefore, violationaction, predictiveaction)$ where:

- $pattern \in \{Exists, Absence, Sequence, Next, Precedes, One\ to\ one\ precedes, Response, One\ to\ one\ response, SoD, BoD, Performed\ by\ role, Performed\ by\ resource\}$ defines the pattern from which the rule is instantiated,
- $model \in PM$ is a reference to the process model against which the rule has to be monitored,
- $antecedent \in \{ex, not(ex) | ex \in RE\}$ where $not(ex)$ means that event ex has not been observed,
- $consequent \in \{ex, not(ex) | ex \in RE\}$ where $not(ex)$ means that event ex has not been observed,
- condition is the data condition that is to be examined at the occurrence of the rule's $antecedent$
- $scope\ start \in RE$ defines the delimiting start event of the rule's scope,
- $scope\ end \in RE$ defines the delimiting end event of the rule's scope,
- $multiplicity$ is a constraint on the number of occurrences of the rule's $antecedent$,
- $WA \subset RE$ is the set of events that must not occur between the $antecedent$ and $consequent$,
- $time\ span$ is the time window in/out of which the $consequent$ event must be observed,
- $alert\ time\ span$ is the time window after which there is a possibility of violation if the $consequent$ was not observed,
- $isBefore \in \{0, 1\}$ is a Boolean value indicating wether the $consequent$ event must be observed *before* or *after* the end of the $time\ span$
- $violation\ action \in \{alert, suspend\}$ defines the action to be taken upon the occurrence of a violation,
- $predictive\ action \in \{alert, suspend\}$ defines the action to be taken when there is a possibility of a violation.

When any property does not apply to a rule pattern, it is represented as $\perp$. We define CR as the set of all compliance rules registered with the monitoring component.

As per Definition 2.2, there might be a *time span* window that puts further constraints on the observation of the $consequent$ event with respect to the $antecedent$ event. This is also further controlled by the *isBfore* property. So, if *isBore = 1*, the pattern requires that the $consequent$ event to be observed *before* time span elapses otherwise there is a violation. Whereas, if *isBefore = 0*, then $consequent$ has to be observed *after* the time span elapses.

Composite patterns are used to logically connect other patterns by Boolean operators AND, OR, NOT, etc. This is used to define complex rules that can not be expressed merely by atomic patterns, which is especially helpful when sub-ideal level of compliance is also needed [32].

C. Complex Event Processing (CEP)

In traditional systems, data are static in the system while the queries used are changing. For example in traditional database systems, the data are stored in tables and users can write different queries that access those tables to process data and get the results. However, when using complex event processing (CEP) technology, the roles of data and queries are reversed; where the queries will be static and data or events will be dynamic based on the input event streams from different sources. These queries will be checked against incoming streams of events to verify that queries are answered correctly.

In the context of business process compliance, monitoring queries (rules) are commonly represented in the form of event-condition-action (ECA) rules [23]. Thus, the first trigger to evaluate a query is to find the matching input event on the stream. In general, there are two types of events, *raw* (*low-level*) events and *business-level* (*composite*) events [31]. Composite events might result from the evaluation of one or more query whereas raw events are generated from the different execution environment. In the context of process compliance monitoring, *raw* events are generated based on a change of the state of running process instances (cases) and their activities. In practice, an event stream contains a set of associated events, that are chronologically ordered. These events are generated from different sources. This stream of events could be homogeneous in which all events must be from the same type, or heterogeneous in which all events may be of different types [23].

III. Case Studies

In this section, we introduce two case studies addressing the banking domain. The *first* case study has been conducted in the Governance, Risk and Compliance Technology Centre

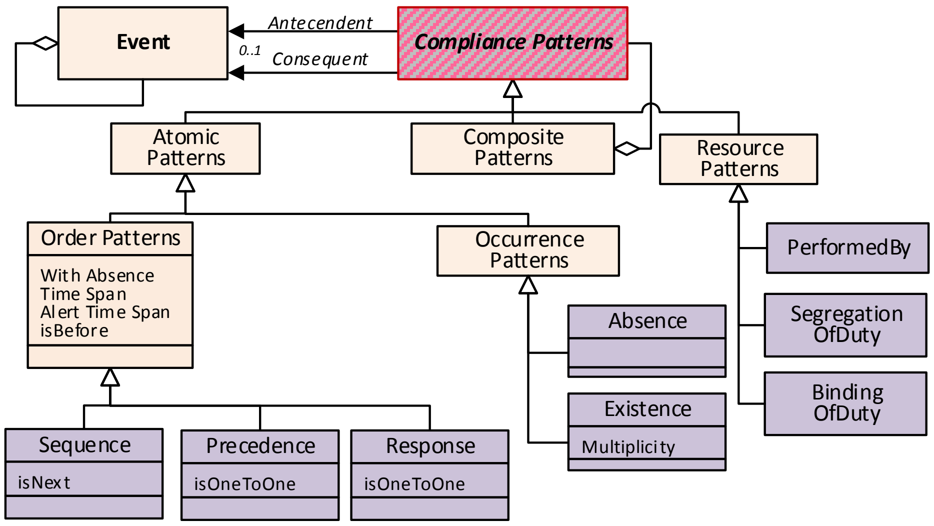


Fig. 2: Compliance Patterns

(GRCTC)[2] focusing on Anti-money Laundering (AML). The *second* case has been conducted within the scope of the EU-funded research project COMPAS[3], targeting the loan processing business scenario. This case study was performed by Thales Services, France[4], as one of COMPAS industrial partners; a company heavily operating in the banking domain. Taking into account the demands for strong regulation compliance schemes, such as Sarbanes-Oxley (SOX), BASEL-III, ISO 27000 and sometimes contradictory needs of the different stakeholders, such business environments raise several interesting compliance requirements. Anti-money laundering is a pressing concern to any organization operating in the financial industry, as it is tightly adjunct to terrorism and proliferation financing. Despite the fact that it is not possible to precisely quantify the amount of money laundered every year, in [48], it has been shown that billions of US dollars certainly are. On-going work in GRCTC in collaboration with respectable Irish financial organizations focuses on developing an end-to-end AML business process encoded in the BPMN v2.0 standard, which is established based on best practices and the Financial Action Task Force (FATF) 40 recommendations [55]. The U.S. Patriot act of 2001 [44] was considered as the main source of compliance requirements targeting anti-money laundering, which constitutes a large number of compliance requirements structured into twelve sections. The interpretations of embedded requirements and encoding them in the Semantics of Business Vocabulary and Business Rule (SBVR) standard [45] as a structured natural language is ongoing work in GRCTC [20]. Since the interpretation of compliance sources mandatory requires legal expertise [58, 60], GRCTC has hired three legal experts, who work very closely with compliance experts to interpret the AML directives and encode them in SBVR. We use the second case study (Loan Approval scenario) as a running scenario to exemplify the next discussion. The conducts and the overall findings of both case studies are discussed later in Section VII.

[2]GRCTC: http://www.grctc.com/
[3]COMPAS: http://www.compas-ict.eu
[4]Thales Group: https://www.thalesgroup.com/

A simplified model of the loan origination and approval process is depicted in Figure 3 using BPMN (Business Process Model and Notation). The process flow can be described as follows: Once a customer loan request is received, the credit broker checks customer′s banking privileges status. If privileges are not suspended, the credit broker accesses the customer information and checks if all loan conditions are satisfied. Next, a loan threshold is calculated, and if the threshold amount is less than 1M Euros, the post-processing clerk checks the credit worthiness of the customer by outsourcing to a credit bureau service. Next, the post-processing clerk initializes the loan form and approves the loan. If the threshold amount is greater than 1M Euros, the supervisor is responsible for performing the same activities instead of the post-processing clerk. Next, the manager evaluates the loan risk, after which she normally signs the loan form and sends the form to the customer to sign. A legal waiting time of 7 days is provided to the customer to send back the signed contract. If a timeout occurs, which means that seven days have passed and the Customer has not sent the signed contract, the relevant loan approval application is closed by the system and the process terminates.

Table I shows an excerpt of the compliance requirements imposed on this loan approval scenario. This table is populated after applying the refined methodology described in [58, 60]. The first and second columns of the table allocate a unique reference and an organization-specific interpretation of the requirement, respectively. The third column represents the pattern-based representation of interpreted compliance requirements.

IV. BP-MaaS: Framework Overview

Figure 4 provides an overview of the generic business process compliance management BP-MaaS framework. In principle, the framework has two major components: Compliance management component (right hand side of Figure 4) and Business process management component (left hand side of Figure 4). The framework is generic, however, in the

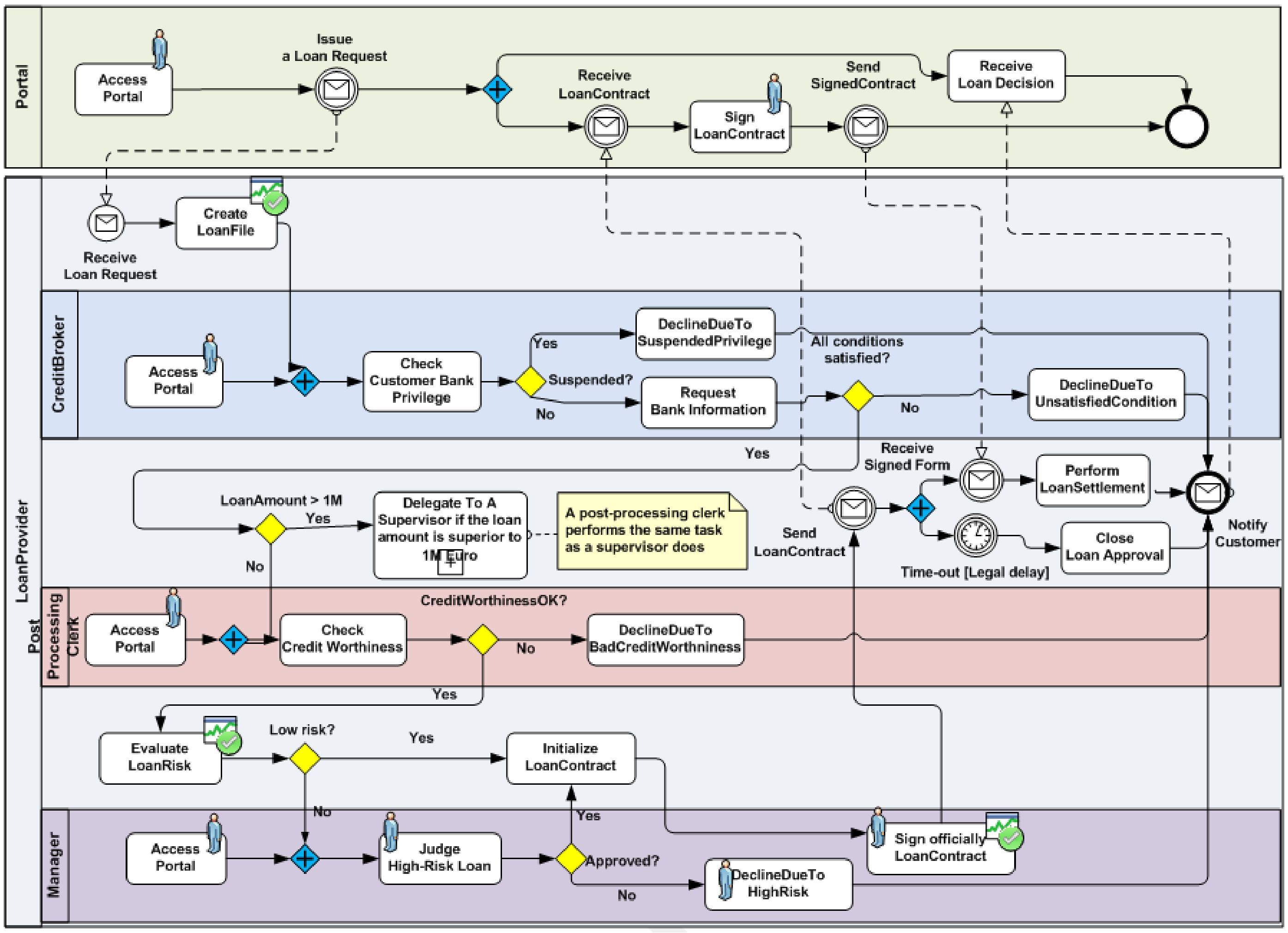


Fig. 3: A simplified BPMN model of the loan origination and approval process

next discussion we are considering Business Process Modelling and Notation standard [25] as a possible instantiation of the proposed framework. As shown in the figure, the BP-MaaS exhibits two basic abstract roles: *business expert* and *compliance expert*. The *business expert* is responsible for the definition, design, development and management of service-enabled business processes, while taking compliance requirements into consideration. The *compliance expert* is responsible for the refinement, interpretation and specification of compliance requirements emerging from various internal and external compliance sources in close collaboration with the business expert.

On the top of the monitoring components is the Compliance Management Knowledge base (CMKB) that incorporates and integrates a set of ontologies to capture the different perspectives of the compliance and business spheres [21]. In practice, compliance requirements usually originate from various sources, including laws and regulations, standards, public and internal policies, partner agreements, etc., and organizations are continuously required to comply with increasing number of diverse and evolving laws and regulations. Furthermore, compliance and business concepts may be treated differently by different stakeholders with different backgrounds. This ambiguity results in inconsistency, which makes it infeasible to share and re-use business and compliance specifics. All these problems make it infeasible for automated compliance monitoring and analysis. In principle, the need to manage regulatory and compliance data, especially in heavily-regulated domains, exceeds the abilities of current information systems. The majority of compliance management approaches in the literature (cf. Section VIII) assumes this ontological alignment, due to the complexity of this problem. Therefore, our framework is equipped with a uniform conceptualization of the process and compliance space which provides various advantages including:

- Enabling the sharing and re-use of compliance and business knowledge
- Eliminating any ambiguity that may result in unforeseen inconsistencies
- Significantly facilitating the communication between stakeholders with different backgrounds, e.g., compliance and business experts
- Ensuring the ontological alignment between business and compliance concepts

TABLE I: Compliance Requirements

Compliance Requirements	Control	Pattern Representation
1) Access to sensitive data is restricted	Only Credit Broker, Post-processing Clerk and Supervisor roles can access the Credit Bureau service and the Customer Information File service.	R1.1: PerformedByRole(Verify Banking Privilege,Role=Credit Broker) R1.2: PerformedByRole(Acquire Bank Information,Role=Credit Broker) R1.3: PerformedByRole(Check Credit Worthiness,Role$\in$ $\{$Post Processing Clerk ,Supervisor$\}$) R1.4: PerformedByRole(Prepare Package Price,Role $\in$ $\{$Post Processing Clerk , Supervisor$\}$)
2) Only credit worthy customers receive a loan.	The Credit Broker checks the Customer Bank Privilege and rejects the loan request if the Credit Bureau Service indicates that the customer's banking privileges have been suspended.	R2.1: PerformedByRole(Verify Banking Privilege,Role = CreditBroker) R2.2: Response(Verify Banking Privilege, Decline By Suspeneded Banking Privilege, condition=(Suspeneded = Yes))
3) Duties in Loan Origination are segregated.	The Credit Broker checks the banking privileges and the Post-processing Clerk or the Clerk Supervisor checks the credit worthiness.	R3.1: Response(VerifyBankingPrivilege,CheckCreditWorthiness) R3.2: Absence(CheckCreditWorthiness,scop_end = VerifyBankingPrivilege) R3.3:SoD(VerifyBankingPrivilege,CheckCreditWorthiness)
4) High loan request have to be processed by supervisors.	If the loan requests credit is below 1 million EURO, the Post-processing Clerk of Credit Operations checks the credit worthiness, if it is higher than 1 million EURO the Clerk Supervisor checks the credit worthiness of the customer.	R4.1: PerformedByRole(Check Credit Worthiness,Role = Post Processing Clerk, condition = Threshold $< 1M$ Euro) R4.2: PerformedByRole(Check Credit Worthiness,Role = Supervisor, condition = Threshold $\geq 1M$ Euro)
5) Duties in Loan Origination are adequately segregated.	As a final control, the branch office Manager has to check high-risk loan requests whether it is profitable and risks are acceptable and makes the final approval (or denial) of the request. Only the Office Manager is able to perform the final approval.	R5.1: PerformedByRole(Evaluate Loan Risk,Role= Manager) R5.2: Exists(Evaluate Loan Risk) R5.3: PerformedByRole(Sign Loan Contract,Role= Manager) R5.4: PerformedByRole(Decline By High Risk,Role= Manager) R5.5: Response(Evaluate Load Risk, Decline By High Risk,condition=(LowRisk = No)) R5.6:Response(Evaluate Load Risk, Sign Loan Contract,condition=(LowRisk = Yes) R5.7: Exists (Sign Loan Contract) XOR Exists(Decline By High Risk)
6) Customers receive a certain period of time for reflection.	The Credit Broker can start a loan approved by the customer, only if five work days or more have elapsed since the loan approval form was sent.	R6.1: Response(Send Loan Contract,Perform Loan Settlement , time span = after 5 Working days) R6.2: PerformedBy(Perform Loan Settlement, Role= Credit Broker)
7) Customers personal data is handled confidentially.	The customer receives an email notification when his data is collected from the Credit Bureau service.	R7.1:Response(Acquire Bank Information,Send Notification To Customer,WA=All other) R7.2: Response(Check Credit Worithness,Send Notification To Customer,WA=All other)

- Improving the level of automation provided by the framework

To achieve these goals, three main ontologies are employed: (i) *BP ontology*: an ontology that captures the semantics of the adopted BP language, e.g. BPMN, BPEL, (ii) *Domain Ontology*: an ontology that represents the concepts and relationships that exist in the domain of interest, e.g., medical, transport, aerospace, etc. and (iii) *Regulatory Ontology*: an ontology that formalizes the requirements, controls and rules of compliance imperatives. Ontologies in the CMKB may be represented formally in the Ontology Web Language (OWL2.0) [63]. We build upon the BPMNO ontology [18] as the Business Process ontology. BPMNO provides a rich ontological representation of the BPMN v2.0 standard, which we consider as one of the possible instantiation of the framework. If another business process language is adopted, e.g., Business Process Execution Language (BPEL) [43], an ontology should be incorporated to capture its semantics. Moreover, we utilize the Financial Industry Business Ontology (FIBO) [15, 12] as the domain ontology since the case study we use in this paper (and introduced in Section III) concerns with the financial/banking domain. FIBO is an adopted OMG standard, a collaborative initiative led by industry members of the Enterprise Data Management Council (EDMC) in collaboration with the Object Management Group (OMG). Similarly, if another domain is considered, relevant domain ontology should be incorporated.

Business process management component (left hand side of Figure 4) starts by the business expert defining new BPMN model or re-use/update an existing one. In a green scenario, if the BPMN process is built from scratch, concepts and relations from FIBO can be directly used to guide its design and development. However, if BP models already exist (which is the common case), concepts from FIBO can be used to provide semantic annotation to existing BPMN models (the semantic labelling link between 'Domain Ontology' and 'BP Ontology' as shown in Figure 4)), and then various reasoning mechanisms can be applied to ensure the correctness of these annotations as proposed in [18]. Examples of concepts in FIBO are: 'Agent', 'Person', 'National id', 'real estate', 'agreement', 'contract', 'ownership', 'Asset', etc.; examples of relations are: 'manages', 'provides', 'represents', 'is issued by', 'is appointed by', etc. Examples of concepts in BPMNO are: 'Activity', 'Event types', 'Task', 'Gateway', etc.

Regulatory ontology is under development that addresses the regulatory domain by capturing concepts and relationships necessary to represent compliance patterns and compliance rules. Key concepts in this ontology are 'Input Data', 'Events', 'Condition', 'Action', 'Boolean Operator', etc. As discussed in Section II, the framework makes use of *compliance patterns* to represent compliance requirements mainly to provide an abstraction layer, so that experts do not have to go into the low-level and intricate details of the underlying formal/logical language. Therefore, the Regulatory ontology also maintains concepts such as 'Compliance Patterns', 'Occurrence Patterns', 'Timed Patterns', 'Operand', 'Label', etc. As shown in Figure 4, Compliance Management Ontology, BP ontology (e.g., BPMNO), Domain Ontology (e.g., FIBO) and Regulatory ontology represent the Terminological part (TBox) of the CMKB. Instances from these ontologies populate the Compliance Requirements Repository (CRR), and Business Process Repository (BPR), representing the Assertional part[5] (ABox) of the knowledge base.

The business process and compliance management activ-

[5]The terms ABox and TBox are used to describe two different types of statements in ontologies. TBox statements describe a conceptualization, a set of concepts and properties for these concepts. ABox are TBox-compliant statements about individuals belonging to those concepts.

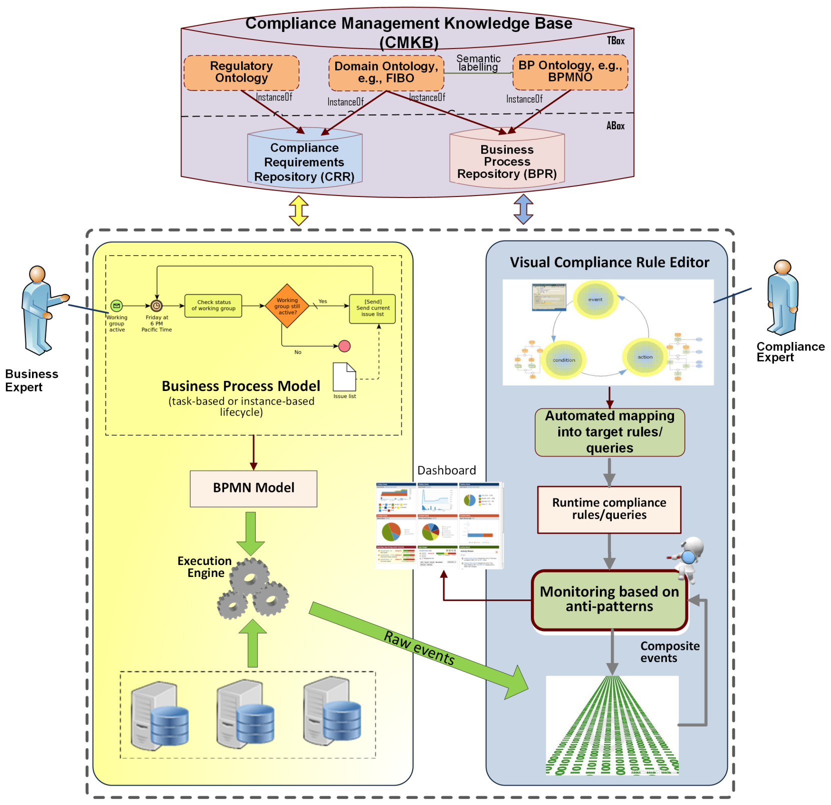

Compliance Management Knowledge Base (CMKB)
TBox
Regulatory Ontology
Domain Ontology, e.g., FIBO
Semantic labelling
BP Ontology, e.g., BPMNO
InstanceOf
ABox
Compliance Requirements Repository (CRR)
Business Process Repository (BPR)
Business Expert
Business Process Model
(task-based or instance-based lifecycle)
BPMN Model
Execution Engine
Raw events
Visual Compliance Rule Editor
Compliance Expert
Automated mapping into target rules/ queries
Runtime compliance rules/queries
Dashboard
Monitoring based on anti-patterns
Composite events

Fig. 4: Framework Overview

ities start independently of each other, but they should be aligned with each other. This agrees with the necessity of the separation of compliance and business process management practices [51], [22], mainly because:(i) they have different objectives: ownership and governance perspectives , (ii) different lifecycles, (iii) different nature: compliance requirements are normative/descriptive by nature (describing what should be done), therefore, declarative languages are more suited to capture them, while business process specifications are prescriptive (describing how business activities should take place), therefore, procedural languages are the best to represent them, and (iv) the two specifications may conflict/contradict with each other. Therefore, they should be separated, however, their interrelationships should be carefully studied and maintained.

As mentioned above, Business process management practices commences with the business expert defining and modelling business process requirements using BPMN. The design process supports both task-based and instance-based lifecycles (cf. Section II). However, different execution environments could have slightly different lifecycle models, where adapters could be utilized to fill in this gap. In this case, adapters have to be implemented to correlate or supplement missing events that are expected by the monitoring component.

This step is highly iterative, such that the BPMN model follows multiple iterations of design and refinements to faithfully represent business logic and requirements. The outcome of this step is a BPMN model capturing the control and data flow of the business logic. The loan approval business scenario introduced in Section III (Figure 3) is an example of a BPMN model in the banking domain. The framework currently relies on the *Oryx* system [16] as its business process modeling environment that supports the BPMN 2.0 exchange format. However, the framework could be easily adapted to support other formats as well.

The BPMN model is then deployed on a business process (BP) engine in a possibly distributed environment. During execution, the BP engine emits different types of defined events reflecting the evolution of the execution of the BP model. These raw events are sent synchronously to the compliance management component (right-hand side of Figure 4), where business process monitoring comes into play. As shown in Figure 4, the interconnection between the business process management component and the compliance management component is through the emitted raw events.

From the compliance management side (right-hand side of Figure 4), the compliance expert starts by the specification of applicable compliance requirements using compliance patterns (as described in Section II) and using concepts from the CMKB. Table I in Section III shows an excerpt of some compliance requirements imposed on the loan approval model shown in Figure 3. To further ease the job of the compliance expert, we have developed a graphical compliance rule editor (on top of *Oryx*) that enables the compliance expert to intuitively build pattern-based expressions in a drag-and-drop fashion (more details about the implementation are presented in Section VI). Graphical pattern-based expression are then automatically mapped into the target formal/query language and then feed the monitoring component. As one possible realization of the framework, we have defined the mapping scheme from graphical pattern-based expressions into Event Processing Language (EPL) queries following a complex event processing approach (details are given next in Section VI). The framework currently adopts its customized format for storing and exchanging the compliance rules. Adopting a standard rule exchange format, e.g. RuleML [13], is left as a future work.

Raw events as data streams sent from the BP engine are then interrelated to generate composite events that are compliant with the lifecycle model on the stream (based on defined event-patterns of interest). The monitoring component then evaluates these rules/queries against the composite events and appropriate actions are taken in case a runtime violation is detected by the monitoring component. More specifically, when a new event is received on the stream, it automatically gets directed to the set of rules that have subscribed for that event for processing and detecting or predicting any possible non-compliance instances. As a result, the rule/query might generate another event on the stream to signal the compliance status of the instance that has generated the event. The monitoring is achieved by means of *anti-patterns*, a novel compliance monitoring evaluation approach we introduce in this paper (details are presented next in section V). Finally, compliance results are presented to the compliance/business experts on dashboards in detailed and aggregated manner with the support of various charts/indicators. Business and compliance experts scrutinize the results presented on the dashboards, which may initiate multiple iterations of the business process or compliance lifecycles for improvements.

Figure 4 highlights, by dashed lines, the components which represent the main focus of this paper. More specifically, the continuous runtime monitoring of compliance requirements on the basis for anti-patterns, which will be discussed in more detail next in Section V. The proof-of-concept of the proposed approach as a runtime monitoring tool-suite is presented in Section VI. This is followed by a discussion of validation and evaluation efforts in Section VII

V. Monitoring Compliance

In this section, we discuss the mechanics of the monitoring process in our framework. In principle, we follow the notion of *anti-patterns*, however, we are mainly focusing on the implementation of this notion for achieving the compliance requirements at the runtime phase instead of the design phase which has been considered by previous related work [4, 5]. In particular, we directly look for sequences of events that indicate that a violation has occurred or likely to occur. To achieve this, a set of anti-patterns are inferred for each compliance pattern. One advantage of looking for anti patterns is that the root cause of the violation is given without incurring additional costs [32]. Whenever a query finds a match, it generates a *Rule Violation Event*, see Definition 5.1, on the stream. This event can then be caught by another query to take an action against the violation.

Definition 5.1 (Rule Violation Event): A rule violation event is a tuple $(rule, case, isViolation, RC, timestamp)$ where:

- $rule \in CR$ is a reference to the rule for which a violation has been detected or predicted,
- $case \in PI$ is a reference to the process instance for which a violation has been detected or predicted,
- $isViolation \in \{true, false\}$ is a flag to indicate whether this an actual or a prediction of a violation,
- RC is the sequence of events that is the root cause of the violation,
- $timestamp \in \mathbb{N}$ is the time stamp at which the event occurred.

We define RVE as the set of rule violation events generated.

As per Definition 5.1, a rule violation event can be thrown even in the case where there is a possible violation. This is distinguished from the actual violation by the flag $isViolation$. Some of the patterns require that process tasks have to take place within a specified time span. Therefore, we assume that when a time span expires, a timer event is thrown to indicate that the time span has expired.

Definition 5.2 (Timer Event): A timer event is a tuple $(rule, case, time\ span, time\ stamp)$ where:

- $rule \in CR$ is a reference to the rule for which a violation has been detected or predicted,
- $case \in PI$ is a reference to the process instance for which a violation has been detected or predicted,
- $time\ span$ is the time span defined in $rule$,
- $time\ stamp \in \mathbb{N}$ is the time stamp at which the event occurred.

We use the notation $timer-event(r, c, time\ span)$ to refer to the timer event generated for rule r, case c and for $time\ span$. We define TE as the set of timer events generated.

Definition 5.3 (Event Stream): Event stream σ is a sequence of events on the form $e_1, e_2, \ldots, e_k$ where $e_i \in RE \cup RVE \cup TE, 1 \leq i \leq k$

As per Definition 5.3, the event stream is heterogeneous as it could hold raw events from the process execution, rule evaluation events and timer events. Moreover, raw events can belong to different process instances. We use Definition 5.4 to retrieve the history of execution of a certain process instance. It is also possible to retrieve a specific scope of the case history. Case history is used by the anti pattern queries to look for violations.

Definition 5.4 (Case History): Case history $\sigma_i(start, end)$, where $i \in PI$, $start, end \in RE$ is a projection on the event stream σ where only events related to instance i are projected. $start$ and end default to the case start and end events respectively. They are used to project a certain scope of the case history rather than the complete one. When an event e is a member of σ_i, we denote that as $e \in \sigma_i$.

Definition 5.5 (Count of Event Occurrence): Given that σ_i is a case history of process instance $i \in PI$ and $e \in RE$, we define a function $count(e, \sigma_i)$ that determines the number of occurrences of e in σ_i.

As shown in Figure 2, compliance rules can be either atomic or composite. In the following sections, we describe the anti-patterns, i.e., the violation scenarios for each atomic pattern.

A. Exists and Absence Anti Patterns

The *Exists* pattern requires that the $antecedent$ event to occur a certain number of times within a certain scope in the process instance where a specific data condition holds. A violation to this rule occurs when the multiplicity constraint is no longer satisfied within the specified scope where the data condition holds with the antecedent event. Specifically, a violation takes place when either:

- A sequence of events where the scope start is observed and then the scope end is observed and in-between the antecedent occurs less than the minimum of the rules multiplicity.
- A sequence of events where the scope start is observed and then the antecedent occurs more than the maximum of the rules multiplicity.

These two possibilities are captured by Definition 5.6.

Definition 5.6 (Exists Violation): A violation to an Exists rule, $exists(pm, antecedent, \bot, condition, scope\_start, scope\_end, multiplicity, \emptyset, \bot, \bot, alert, alert)$, occurs in a process instance $i, i \in PI$ iff:

- $\exists r = \sigma_i(scope\_start, scope\_end) : multiplicity.min > count(antecedent, r)$. Or,
- $\exists r = \sigma_i(scope\_start, antecedent) : multiplicity.max < count(antecedent, r)$.

The *Absence* pattern requires that the $antecedent$ event is never observed within a certain scope in the process instance where a specific data condition holds. Basically, the absence pattern can be seen as a special case of the *Exists* pattern where $multiplicity.min = multiplicity.max = 0$. Thus, the two possible violations scenarios in Definition 5.6 can be applied.

B. Response Anti Patterns

The *Response* pattern can be violated in the following cases:

- The rule's $antecedent$ is observed but the $consequent$ never occurs within the monitoring scope and time span, if defined, when $isBefore = true$,
- The rule's $antecedent$ is observed and then the $consequent$ is observed within the monitoring scope but before time span, if defined, where $isBefore = false$,
- The rule's $antecedent$ occurs and any of the forbidden events $with\ absence$ occurs before the rule's $consequent$,
- There is a (possible) violation if the (alert) time span elapses before the occurrence of the $consequent$ event, when $isBefore = true$.

Definition 5.7 (Response Violation): A violation of a Response rule, $response(pm, antecedent, consequent, condition, scope\ start, scope\ end, \bot, WA, time\ span, alert\ time\ span, isBefore, alert, alert)$, occurs in a process instance $i \in PI$ iff:

1) $\exists r = \sigma_i(scope\ start, scope\ end) : antecedent \in r \ \wedge consequent \notin r \ \wedge consequent.timestamp > antecedent.timestamp$. Or,
2) $\exists r = \sigma_i(scope\ start, timer\ event(response, i, time-span)) : antecedent \in r \ \wedge consequent \notin r \ \wedge consequent.timestamp > antecedent.timestamp$, where $isBefore = true$. Or,
3) $\exists r = \sigma_i(scope\ start, consequent) : antecedent \in r \ \wedge consequent.timestamp > antecedent.timestamp \ \wedge \ \nexists t = timer\ event(response, i, time\ span) : t.timestamp < consequent.timestamp$, where $isBefore = false$. Or,
4) $\exists r = \sigma_i(scope\_start, absent\ event) : absent\ event \in WA \ \wedge antecedent \in r \ \wedge consequent \notin r \ \wedge consequent.timestamp > antecedent.timestamp$.

Definition 5.7 formalizes the different violation scenarios for the *Response* pattern. The essence is that we can observe a sequence of events that starts with the $scope\ start$ event in which the $antecedent$ is observed but not the $consequent$ afterwords, the first, third and fourth case. This is where the time stamp is used for comparison. The difference between the three cases 1, 2 and 4 is in the closing event of the event sequence. In the first case, the $scope\ end$ is observed before having observed the $consequent$. In the third case, the $timer\ event$ which is generated when the time span elapses is observed. In the last case, one of the forbidden events $absent\ event \in WA$ is observed before $consequent$ and thus a violation has occurred. The second case however, detects that the $consequent$ has already occurred. However, there is a violation in this scenario if we fail to observe the timer event before the $consequent$. This is required only when the *Response* has its property $isBefore = false$ which means that the time span *must* elapse before observing the $consequent$, cf.Rule $R6.1$ in Table I.

The *One to One Response* is a special case of the *Response* pattern. In addition to the violation scenarios of the main pattern, *One to One Response* is violated if two occurrences of the $antecedent$ event are observed without observing the $consequent$ in between.

Definition 5.8 (One to One Response Violation): A violation of a One to One Response rule, $one\_to\_one\_response(pm, antecedent, consequent, condition, scope\ start, scope\ end, \bot, WA, time\ span, alert\ time\ span, isBefore, alert, alert)$, occurs in a process instance $i \in PI$ if a violation to its underlying Response rule occurs or $\exists r = \sigma_i(scope\ start, antecedent_1) : antecedent_2 \in r \wedge consequent \notin r \wedge consequent.timestamp > antecedent_1.timestamp$.

C. Sequence and Next Anti Patterns

The *Sequence* pattern requires that $antecedent$ occurs and is then followed by $consequent$ within a scope where the data condition holds. We can see that *Sequence* pattern can be a conjunction of the *Exists* and Response patterns. Thus, the violation of the *Sequence* pattern occurs if any of the violation scenarios of the *Exists* or the *Response* patterns occurs.

The *Next* pattern is a special case of the sequence pattern which in addition to the restrictions imposed by the *Sequence* pattern requires that no other events are observed in the process execution between $antecedent$ and $consequent$. In this regard, the anti patterns of the *Exists* and *Response* can be used. To enforce that no other events are allowed in-between, we can make use of the set WA that indicates the forbidden events. In this case, the set of forbidden events must refer to events of all other tasks in the process to ensure that nothing else happens between the $antecedent$ and the $consequent$.

D. Precedes Anti Patterns

The *Precedes* pattern can be violated in the following cases:

- The rule's $antecedent$ occurs but never the $consequent$ before it within the monitoring scope and time span, if defined,
- The rule's $antecedent$ occurs and the $consequent$ is observed before it within the monitoring scope but $before$ the time span elapses, in case the rule $isBfore = false$,
- The rule's $antecedent$ occurs and any of the forbidden events WA occurred before it and after the rule's $consequent$,

For *Precedes* rules, it is not possible to predict violations as it is a history-looking rule by nature.

Definition 5.9 (Precedes Violation): A violation of a Precedes rule, $precedes(pm, antecedent, consequent, condition, scope\ start, scope\ end, \perp, WA, time\ span, \perp, isBefore, alert, \perp)$, occurs in a process instance $i \in PI$ iff $\exists r = \sigma_i(scope\_start, antecedent)$:

- $scope\ end \notin r \wedge consequent \notin r$. Or,
- $scope\ end \notin r \wedge consequent \in r \wedge antecedent.timeStamp - consequent.timeStamp > time\ span$, if $isBefore = true$. Or,
- $scope\ end \notin r \wedge consequent \in r \wedge antecedent.timeStamp - consequent.timeStamp < time\ span$, if $isBefore = false$. Or,
- $\exists absent\ event \in WA : scope\ end \notin r \wedge consequent \in r \wedge absent\ event \in r \wedge absent\ event.timeStamp > consequent.timeStamp$.

Definition 5.9 formalizes the different violation scenarios for the *Precedes* pattern. The violation occurs when a projection on case history which starts with the $scope\ start$ event and ends with the $antecedent$ event and the scope is active, i.e., the $scope\ end$ was not observed before the $antecedent$ event $scope\ end \notin \sigma_i(scope\ start, antecedent)$ and:

- In the first case, the $consequent$ event was not observed as a member of that projection of case history.
- In the second case, the $consequent$ event was observed in the case history projection but the difference between the time stamp of the $antecedent$ and $consequent$ events is more than the prescribed time span in the rule when the rule's $isBefore$ property is set to $true$.
- The third case is similar to the case above with one difference is that the difference between the $antecedent$ and the $consequent$ events timestamps is less than the required time span. This is a violation only in case the rule has the $isBefore$ property set to $false$.
- In the last case, in addition to observing the $consequent$ event one of the forbidden events, $absent\ event \in WA$ was observed after $consequent$ was observed.

The *One to One Precedes* is a special case of the *Precedes* pattern. In addition to the violation scenarios of the main pattern, *One to One Precedes* is violated if two occurrences of the $antecedent$ event are observed without observing the $consequent$ in between.

Definition 5.10 (One to One Precedes Violation): A violation of a One to One Precedes rule, $one\_to_one\_precedes(pm, antecedent, consequent, condition, scope\ start, scope\ end, \perp, WA, time\ span, \perp, isBefore, alert, \perp)$, occurs in a process instance $i \in PI$ if a violation to its underlying Precedes rule occurs or $\exists r = \sigma_i(scope\ start, antecedent_1) \exists antecedent_2 \in r \nexists consequent \in r : consequent.timestamp > antecedent_2.timestamp$.

E. Separation/Bind of Duty Anti Patterns

The separation of duty pattern is violated whenever the consequent event has the same resource identifier as the antecedent. Note that the pattern does not care about the execution order of the tasks.

Definition 5.11 (Separation of Duty Violation): A violation of a Separation of Duty rule $Separation\_of\_Duty(pm, antecedent, consequent, \perp, scope\_start, scope\_end, \perp, WA, time\_span, \perp, alert, \perp$ occurs in a process instance $i \in PI$ iff:

- $\exists r = \sigma_i(scope\_start, antecedent) : consequent \in r \wedge consequent.resource = antecedent.resource$, or
- $\exists r = \sigma_i(scope\_start, consequent) : antecedent \in r \wedge consequent.resource = antecedent.resource$

As per Definition 5.11, there is a violation if the two events, $antecedent$ and $consequent$ related to the completion of the two separate tasks are observed in any order. In the first case, $consequent$ is observed and then $antecedent$, because we projected the case history until the occurrence of the $antecedent$ where $consequent$ was part of it. Whereas in the second case it was the other way around. In both cases violation occurs if they have the same resource performing the tasks. If $antecedent = completed(A, i)$ and $consequent = completed(B, i)$ then detecting the occurrences of the two events according to Definition 5.11 would signal a violation of a separation of duty rule between tasks A and B. However, there is also a possibility to predict a possible violation if two more separation of duty rules are used where in one of them $antecedent = started(A, i)$ and the $consequent$ remains the same. Whereas in the other rule the $antecedent$ remains the same and $consequent = started(B, i)$. With these rules if a sequence of events $completed(A, i), \ldots, started(B, i)$ is observed as defined by the second case in Definition 5.11, the violation is reported as a warning and there will be a chance to avoid the actual violation by reassigning task B to another resource. So, changing the type of events to detect in case of separation of duty rules helps to predict and avoid violations rather than just reporting its occurrence.

The *Bind of Duty pattern* is actually the negation of separation of duty. So, we can reuse the cases to detect the separation of duty with one change. That is to check that $antecedent.resource <> consequent.resource$.

VI. Implementation

Figure 5 presents the architecture of a proof-of-concept[6] implementation of the *BP-MaaS* framework for compliance monitoring of business processes[7]. In principle, the implementation of our framework consists of three main components: *Compliance rule modeler*, *compliance monitoring engine*, *monitoring dashboard*. Each of these components is described in the following subsections, respectively.

A. Compliance Rules Editor

The compliance rules editor provides the end users with a user-friendly modeling environment where the users can graphically model their compliance rules, in drag and drop style, using custom-built visual notations for the compliance patterns presented in Figure 2. In addition, the modeling environment is equipped with a plugin that exports the rules in XML format to be deployed to the monitoring engine. The *compliance rules editor*is built on top of the *Oryx* editor[8] [16], a popular open source environment for process modeling. Figure 6 shows a snapshot of the rule's editor capturing compliance requirement 3 from Table I, which visualizes compliance rules $R3.1$, $R3.2$ and $R3.3$.

B. Compliance Monitoring Engine

The compliance monitoring engine is responsible for *continuously* evaluating the compliance status of the running process instances against defined compliance rules. The monitoring engine receives the compliance rules in XML format from the rules editor and translates them into a set of queries that are continuously evaluated against the stream of events received from the process execution engine. The engine triggers the execution of the *compliance actions* for any detected violations of the compliance rules in addition to reporting the information of the violated rule and the violating business process instance to the end user by means of updating the monitoring dashboard.

The compliance monitoring engine was built as a service using C# .NET and has used an open source engine for complex event processing (CEP) [23] and event series analysis, *Esper*[9]. In principle, Esper was chosen because of its scalability, ease of modeling as reported in [37]. It provides an environment for developing applications that can process large volumes of incoming messages or events, regardless of whether incoming messages are historical or real-time in nature. It supports filtering and analyzing events in various ways, and responding to conditions of interest. In particular, Esper provides an SQL-like language, Event Processing Language (*EPL*)[10], that provides the standard *SELECT*, *FROM*, *WHERE*, *GROUP BY*, *HAVING* and *ORDER BY* clauses. In this context, streams replace tables as the source of data with events replacing rows as the basic unit of data.

A process execution engine provides the *raw* events the monitoring component needs to keep evaluating the compliance status. To provide these events, we have extended the Activiti [11] business process management platform with event emitters that propagate events reflecting the evolution of process instances and their tasks to the monitoring component. For this, every new instance created or a change of state in an existing process/task instance will be communicated to the monitoring component as a new entry on the respective event stream. From there, ESPER can evaluate the different anti pattern queries against these streams to detect violations. The built-in process model within the Activiti platform has been used to define process models to be executed.

In the rest of this section, we discuss the mapping of anti patterns described in Section V into EPL queries. These are shown as parameterized queries where parameters are enclosed in curly brackets{}. These parameters are actualized at rule registration time at the monitoring component.

```
insert into RuleViolationEvent (processID,Message,RuleID,
RuleType)
select s.ProcessID,'Event␣{Antecedent}({task})␣occurred␣less
than␣{MinOccurs}␣within␣{ScopeStartEvent}({ScopeStartTask})
and␣{ScopeEventEvent}({ScopeEventTask})␣in␣process␣',
 '{RuleID}','{RuleType}'
FROM PATTERN [
every(s={ScopeStartEvent}(cast(s.Task,string)=
'{ScopeStartTask}')->(e={ScopeEndEvent}(cast(e.Task,string)
='{ScopeEndTask}',ProcessID=s.ProcessID) ))] as scope
WHERE {MinOccurs} > (select count(*) from {Antecedent}.win:
keepall() as T
WHERE cast(T.Task,string)='{AntecedentTask}'
and(T.TimeStamp between scope.s.TimeStamp and
scope.e.TimeStamp))
```

Listing 1: Query to detect below-min-occurrences of a rule antecedent

```
insert into RuleViolationEvent (processID,Message,
RuleID,RuleType)
select s.ProcessID,'Event␣{Antecedent}({task})␣occurred␣more
than␣{MinOccurs}␣within␣{ScopeStartEvent}({ScopeStartTask})
and␣{ScopeEventEvent}({ScopeEventTask})␣in␣process␣',
'{RuleID}','{RuleType}'
FROM PATTERN [
every(s={ScopeStartEvent}(cast(s.Task,string)=
'{ScopeStartTask}')->(e={ScopeEndEvent}(cast(e.Task,string)
='{ScopeEndTask}',ProcessID=s.ProcessID)))] as scope
WHERE {MaxOccurs} < (select count(*) from {Antecedent}.win:
keepall() as T
WHERE cast(T.Task,string)='{AntecedentTask}'
and(T.TimeStamp between scope.s.TimeStamp and
scope.e.TimeStamp))
```

Listing 2: Query to detect above-max-occurrences of a rule antecedent

Listings 1 and 2 define the anti patterns in Definition 5.6 as EPL queries that detect *Exists*, *Absence*, *Sequence* and *Next* anti patterns by specifying the values for $MinOccurs$ and $MaxOccurs$. For the *Absence* anti pattern, we can use the query in Lisitng 1 where $MinOccurs = 0$. To detect the first anti pattern for the *Sequence* rule, we can set the $MinOccurs = 1$.

[6]A video demonstration of our framework implementation is available on https://www.youtube.com/watch?v=wRdZKsOi5x4

[7]The source code of BP-MaaS is available on https://github.com/BP-MaaS/BP-MaaS.

[8]https://code.google.com/p/oryx-editor/

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

[10]http://esper.codehaus.org/esper-4.2.0/doc/reference/en/html/epl_clauses.html

[11]http://www.activiti.org/

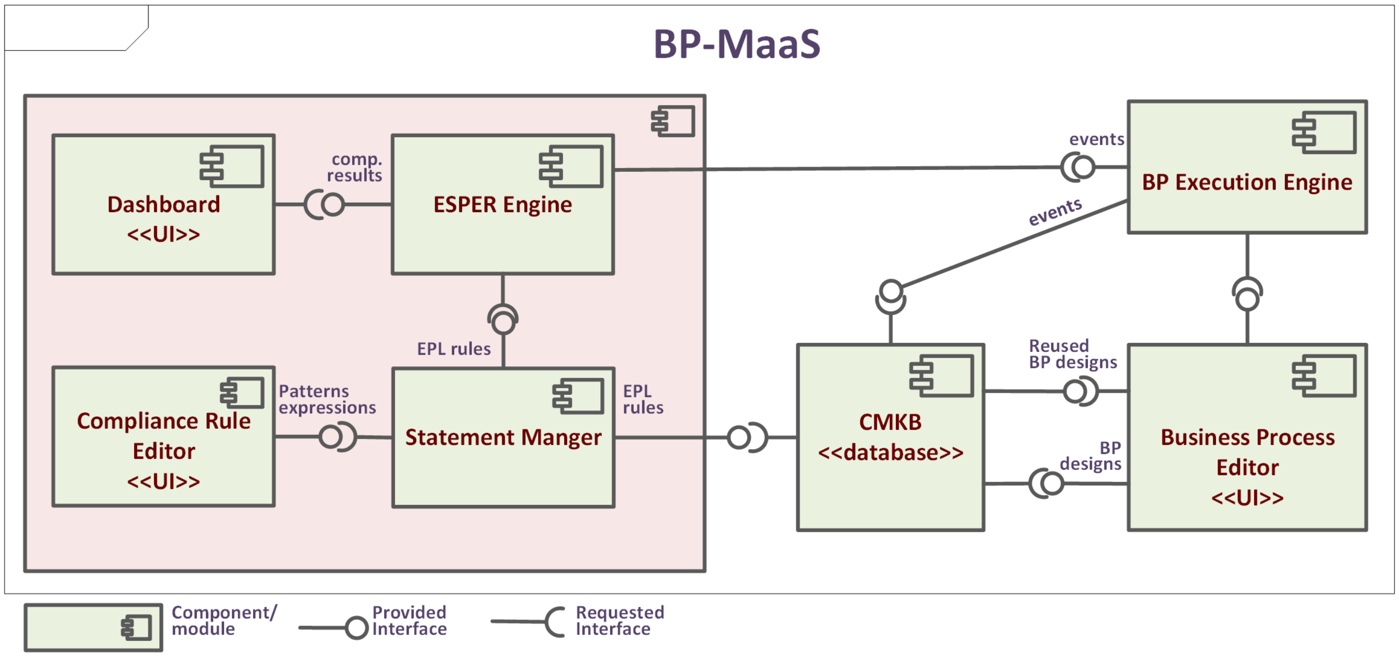


Fig. 5: BP-MaaS architecture represented as a UML component diagram

In Listing 1, EPL *PATTERN* clause is used to look for a sequence of events where the rule's *scope start* is observed and then followed by $->$ *scope end*, this sequence is being matched continuously to the event stream, by means of the $every$ keyword. Whenever there is a match, we get the event instance s matching the start of the scope and the event instance e matching the end of the scope. Then, all event instances of a type matching rule's $antecedent$ whose data payload satisfies the rule's condition and whose $timeStamp$ lies between s and e are counted and compared to the rule's $MinOccurs$. Listing 2 does a similar thing but comparing the occurrences of the $antecedent$ event to the $MaxOccurs$ parameter.

Listing 3, defines an EPL statement for anti patterns of *Sequence* and *Response* rules, cf. 5.7, where the pattern *scope start* followed by $antecedent$ then followed by any of *scope end*, one of the forbidden events, or time span of the rule is observed but never the $consequent$. This query detects the violation for $isBefore = true$ response rules.

```
insert into RuleViolationEvent (processID, Message, RuleID,
RuleType)
select s.ProcessID,'Event␣{AntecedentEvent}({AntecedentTask})
was␣not␣followed␣by␣{ConsequentEvent}({ConsequentTask})within
␣{ScopeStartEvent}({ScopeStartTask})␣and
␣{ScopeEndEvent}({ScopeEndTask})␣in␣process␣','{RuleID}',
 '{RuleType}'
FROM PATTERN [
every(s= {ScopeStartEvent}(cast(s.Task,string)=
{ScopeStartTask})->(every(Antecedent={AntecedentEvent}
(cast(Task,string)={AntecedentTask},
processID=s.processID,Implies(Data, {condition}))
->((e={ScopeEndEvent}(cast(Task,string)={ScopeEndTask},
processID=s.processID) or absent={Absent}(cast(Task,string)
in {WA},processID=s.processID) or timer:interval({TimeSpan})
and not Consequent = {ConsequentEvent}(cast(Task,string)=
{ConsequentTask}, processID=s.processID)))))]
```

Listing 3: Query to detect *Sequence* and *Response* violations

The query in Listing 4 detects the violation when a *Response* rule requires that the time span elapses before the consequent can take place, i.e., $isBefore = false$.

```
insert into RuleViolationEvent(processID,Message,RuleID,
RuleType)
select s.ProcessID,'Event␣{AntecedentEvent}({AntecedentTask})
was␣followed␣by{ConsequentEvent}({ConsequentTask})␣within
{ScopeStartEvent}({ScopeStartTask})␣and␣{ScopeEndEvent}
({ScopeEndTask})␣in␣process␣but␣before␣{TimeSpan}␣elapses',
'{RuleID}','{RuleType}'
from pattern
[every(s={ScopeStartEvent}(cast(Task,string)=
{ScopeStartTask})->(every  Antecedent={AntecedentEvent}
(cast(Task,string)={AntecedentTask},processID=s.processID,
Implies(Data, {condition}))->((e={ConsequentEvent}
(cast(Task,string)={ConsequentTask},processID=s.processID)
and not timer:interval({TimeSpan})))))]
```

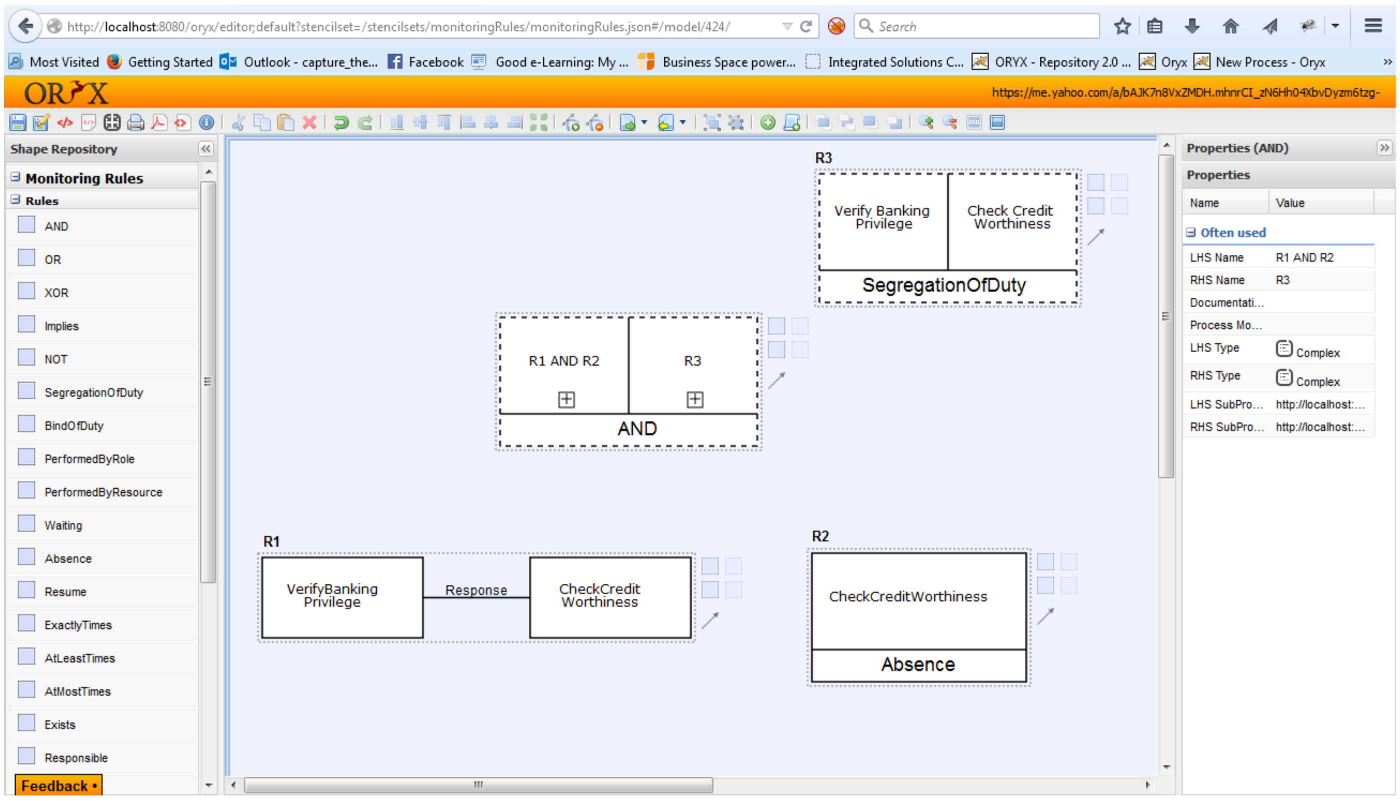
Listing 4: Query to detect *Response* violations for $isBefore = false$

```
insert into RuleViolationEvent (processID, Message, RuleID,
RuleType)
select s.ProcessID,'One␣to␣one␣response␣violation,␣successive
occurrences␣of␣Event␣{AntecedentEvnt}({AntecedentTask})
were␣detected␣without␣detecting␣Event
{ConsequentEvent}({ConsequentTask})␣in␣between␣within
{ScopeStartEvent}({ScopeStartTask})␣and␣{ScopeEndEvent}
({ScopeEndTask})␣in␣process','{RuleID}','{RuleType}'
from pattern
[every(s={ScopeStartEvent}(cast(Task,string)={ScopeStartTask})
->(every  Antecedent={AntecedentEvent}(cast(Task,string)=
{AntecedentTask}, processID=s.processID)
->(Antecedent2={AntecedentEvent}(cast(Task,string)=
{AntecedentTask}, processID=s.processID)
and not Consequent={ConsequentEvent}(cast(Task,string)=
{ConsequentTask},processID=s.processID))))]
```

Listing 5: Query to detect one to one response anti pattern

Listing 5, is dedicated to the detection of *One to One Response* anti pattern, cf. Definition 5.8. The query looks for two occurrences of the $antecedent$ event, after the *scope start* without having any occurrences of $consequent$ event in between.

```
insert into RuleViolationEvent (processID, Message, RuleID,
```



Name	Value
LHS Name	R1 AND R2
RHS Name	R3
Documentati...	
Process Mo...	
LHS Type	Complex
RHS Type	Complex
LHS SubPro...	http://localhost:...
RHS SubPro...	http://localhost:...

Fig. 6: Visual representation of compliance req. 3 ($R3.1$, $R3.2$, $R3.3$) from Table I

```
RuleType)
select s.ProcessID,'Precedes␣Rule␣violation:
Event␣{ConsequentEvent}({ConsequentTask})␣never␣occurred
before␣{AntecedentEvent}({AntecedentTask})␣within
{ScopeStartEvent}({ScopeStartTask})␣and
{ScopeEndEvent}({ScopeEndTask})␣in␣process','{RuleID}',
'{RuleType}'
FROM PATTERN [
every(s={ScopeStartEvent}(cast(Task,string)=
'{ScopeStartTask}')->((not Consequent={ConsequentEvent}
(cast(Task,string)='{ConsequentTask}', ProcessID=s.ProcessID)
and not e={ScopeEndEvent}(cast(Task,string)='{ScopeEndTask}',
ProcessID=s.ProcessID))
 until Antecedent={AntecedentEvent}(cast(Task,string)=
 '{AntecedentTask}', ProcessID=s.ProcessID)))]
```

Listing 6: Query to detect Precedence anti pattern where consequent never occurred

Listing 6, realizes the *Precedes* anti pattern, cf. Definition 5.9, where the *scope start* is observed and later on followed by *antecedent* where there is neither *scope end* nor *consequent* events observed in-between. Listing 7, on the other hand detects the other possibility of violation where after *scope start consequent* is observed but no *scope end* followed by either a timer indicating that the rule's $timeSpan$ has elapsed or one of the forbidden events and then observing the rule's *antecedent* event occurrence.

```
insert into RuleViolationEvent (processID, Message, RuleID,
RuleType)
select s.ProcessID, 'Event␣{ConsequentEvent}
({ConsequentTask})␣is␣observed␣and␣either␣{TimeSpan}␣or
one␣of␣the␣tasks␣in␣{WA}␣were␣observed␣and␣before␣that
␣{AntecedentEvent}({AntecedentTask})␣was␣observed␣within
␣{ScopeStartEvent}({ScopeStartTask})␣and
␣{ScopeEndEvent}({ScopeEndTask})in␣process',
'{RuleID}','{RuleType}'
FROM PATTERN [
every(s={ScopeStartEvent}(cast(s.Task,string)=
'{ScopeStartTask}')->(every(Consequent={ConsequentEvent}
(cast(Consequent.Task,string)='{ConsequentTask}',ProcessID=
s.ProcessID)->(e={ScopeEndEvent}(cast(e.Task,string)=
'{ScopeEndTask}',ProcessID=s.ProcessID)
or absent={Absent}(cast(absent.Task,string) in ({WA}),
ProcessID=s.ProcessID)
or timer:interval({TimeSpan}))
->(Antecedent={AntecedentEvent}(cast(Antecedent.Task,string)=
'{AntecedentTask}',ProcessID=s.ProcessID)))))]
```

Listing 7: Query to detect Precedes anti pattern where forbidden or timer events occur

Listing 8 is an EPL query to detect a violation to a *Precedes* rule where not enough time has elapsed between the occurrence of *antecedent* and *consequent*, $isBefore = false$.

```
insert into RuleViolationEvent (processID, Message, RuleID,
RuleType)
select s.ProcessID, 'Event␣{ConsequentEvent}({ConsequetTask})
occurred␣before␣{AntecedentEvent}({AntecdedentTask})␣within
{ScopeStartEvent}({ScopeStartTask})
and␣{ScopeEndEvent}({ScopeEndTask})␣but␣sooner␣than␣the␣time
span␣{TimeSpan}␣in␣process','{RuleID}','{RuleType}'
FROM PATTERN [
every(
s={ScopeStartEvent}(cast(s.Task,string)='{ScopeStartTask}')
->(every(Consequent={ConsequentEvent}(cast(Consequent.Task,
string)='{ConsequentTask}',ProcessID=s.ProcessID)
and not e={ScopeEndEvent}(cast(e.Task,string)=
'{ScopeEndTask}',ProcessID=s.ProcessID)
->(Antecedent={AntecedentEvent}(cast(Antecedent.Task,string)=
'{AntecedentTask}',ProcessID=s.ProcessID)
and not timer:interval({TimeSpan})))))]
```

Listing 8: Query to detect Precedes anti pattern where time span has not elapsed between antecedent and consequent

Listing 9, is dedicated to the detection of *One to One Precedes* anti pattern, cf. Definition 5.10.

```
insert into RuleViolationEvent (processID, Message, RuleID,
RuleType)
select s.ProcessID, 'Two or more occurrences of Event
{AntecedentEvent}({AntecedentTask}) were detected without
detecting Event {ConsequentEvent}({ConsequentTask})
in between within {ScopeStartEvent}({ScopeStartTask}) and
{ScopeEndEvent}({ScopeEndTask}) in process',
'{RuleID}','{RuleType}'
FROM PATTERN [
every(s={ScopeStartEvent}(cast(s.Task,string)=
'{ScopeStartTask}')->(every
(Antecedent={AntecedentEvent}(cast(Antecedent.Task,string)=
'{AntecedentTask}',ProcessID=s.ProcessID)->(
(not Consequent={ConsequentEvent}(cast(Consequent.Task,
string)='{ConsequentTask}',ProcessID=s.ProcessID))
until Antecedent2={AntecedentEvent}(cast(Antecedent2.Task,
string)='{AntecedentTask}',ProcessID=s.ProcessID)))))]
```

Listing 9: One to one Precedes anti pattern in EPL

```
insert into RuleViolationEvent (processID, Message, RuleID,
RuleType)
select s.ProcessID, 'Events {AntecedentEvent}
({AntecedentTask}) and {ConsequentEvent}({ConsequentTask})
were performed by Resource' + s.Antecedent.resource+ ' within
{ScopeStartEvent}({ScopeStartTask}) and
{ScopeEndEvent}({ScopeEndTask}) in process',
'{RuleID}','{RuleType}'
FROM PATTERN [
every(
s={ScopeStartEvent}(cast(s.Task,string)='{ScopeStartTask}')
->(every(Antecedent={AntecedentEvnt}(cast(Antecedent.Task,
string)='{AntecedentTask}',ProcessID=s.ProcessID)
-> every(Consequent={ConsequentEvent}(cast(Consequent.Task,
string)='{ConsequentTask}',ProcessID=s.ProcessID)))))]
WHERE Consequent.Resource = Antecedent.Resource
```

Listing 10: Separtion of Duty anti pattern

10 is the EPL statement to detect *Separation of Duty* anti patterns. Actually, there has to be another query where the match of $antecedent$ and $consequent$ are swapped to address the fact that the two events can occur in an arbitrary order. Requirement 2 " If a manager needs to travel, the request has to be approved by another manager." can be represented as a separation of duty rule.

1) Detecting Violation of Composite Patterns: In principle, we define an event stream for *rule violation* events as there are event streams for the different process and activity lifecycle events. In particular, activity event streams are supplied by events from the execution environment, cf. 4, whereas the *rule violation* stream is supplied by events based on the matching of the different anti pattern queries. This is shown as starting with `Insert into RuleViolationEvent` in all previous EPL statements. So, whenever a match to the anti pattern occurs, a new instance of the complex event *RuleViolationEvent*, cf. 5.1, is created and sent over that stream.

In case that the initial rule was a composite rule, another query can pick the violation event and act upon it based on the rule's logic. For instance if the rule is of the form $r : r1 \wedge r2$, the query in Listing 11 is defined to detect if there is any occurrence of a rule violation event for either $r1$ or $r2$ and generates another rule violation event for r.

```
insert into RuleViolationEvent
select processID,Message,{r.RuleID} from RuleViolationEvent
where RuleID = {r1} or RuleID = {r2}
```

Listing 11: Monitoring violation of AND composite rule

For complex rules on the form $r = r1 \vee r2$, a query as in Listing 12 is defined to detect the occurrences of violation events of both $r1$ and $r2$ to generate a violation event of the composite rule r. We actually define another query where the sequence of violation events of $r1$ and $r2$ is swapped as the violations might occur in any order. If the composite rule would have more operands, we depend on the associativity of the OR operator to break it into a sequence of binary operators, i.e., $r1 \vee r2 \vee r3 \equiv ((r1 \vee r2) \vee r3)$.

```
insert into RuleViolationEvent
select processID,Message, {r.RuleID} from pattern
[every(
r1=RuleViolationEvent(RuleID={r1})
->(
r2=RuleViolationEvent(RuleID={r2},processID={r1}.processID)))]
```

Listing 12: Monitoring violation of OR composite rule

C. Monitoring Dashboard

The monitoring dashboard is a user-friendly interface for the end-user to monitor the stream of events and manipulate (e.g., adding, removing, activating, deactivating) the set of registered compliance rules in addition to being able to receive the notifications and statistics about the running process instances and detected non-compliance instances (Figure 7). The monitoring dashboard component has been implemented as a .NET program which is responsible for communicating the information between the compliance monitoring engine and the end user. All the three component are loosely coupled as there are no direct dependencies among them. They communicate via messages so that each of them can be replaced as long as the message contract is kept the same.

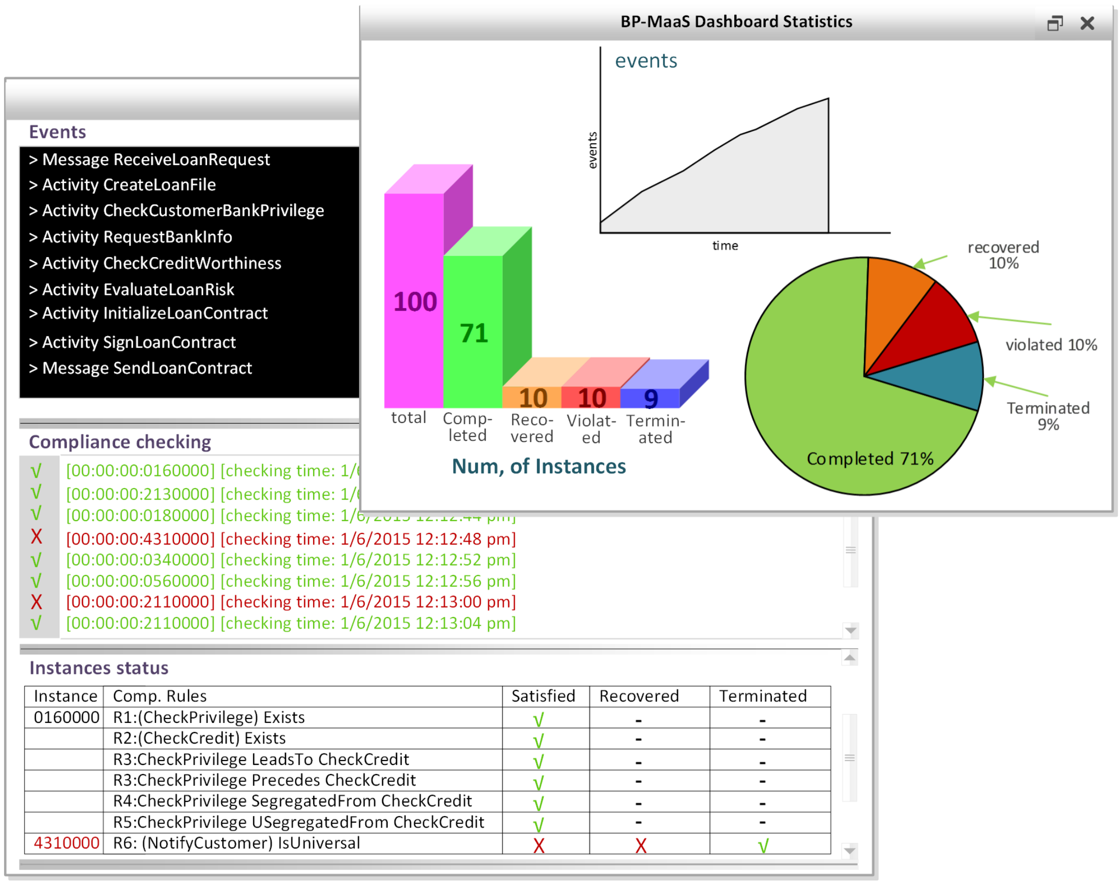
It also should be noted that our framework remains agnostic towards the different systems which can be used for implementing the different components. For example, any process execution environment can be the source of raw events as long as the events are generated according to the reference lifecycle models of processes and tasks. The Esper engine can be replaced with any other stream processing engine (e,g. *StreamBase*[12], *Apache Storm*[13]).

VII. Evaluation

The utility of a design artifact must be rigorously demonstrated via well-executed evaluation methods [28]. Observational methods, such as case studies and field studies, allow an in-depth analysis of the artifact and the monitoring of its use in multiple projects within the technical infrastructure of the business environment.

In Section III, we have introduced two case studies that involved processes operating in the banking domain, addressing different business concerns and entailing a faithful set of rich and diverse compliance requirements that any financial institution is required to comply with. The case studies involved representing relevant compliance requirements in compliance patterns (introduced in Section II-B) with the main objective of investigating the applicability and utility of the overall monitoring approach proposed in this paper, to represent and continuously monitor applicable compliance requirements against running BPMN instances, by means of

[12]http://www.streambase.com/
[13]http://storm.incubator.apache.org/



Events

```
> Message ReceiveLoanRequest
> Activity CreateLoanFile
> Activity CheckCustomerBankPrivilege
> Activity RequestBankInfo
> Activity CheckCreditWorthiness
> Activity EvaluateLoanRisk
> Activity InitializeLoanContract
> Activity SignLoanContract
> Message SendLoanContract
```

Compliance checking

```
√ [00:00:00:0160000] [checking time: 1/
√ [00:00:00:2130000] [checking time: 1/
√ [00:00:00:0180000] [checking time: 1/6/2015 12:12:44 pm]
X [00:00:00:4310000] [checking time: 1/6/2015 12:12:48 pm]
√ [00:00:00:0340000] [checking time: 1/6/2015 12:12:52 pm]
√ [00:00:00:0560000] [checking time: 1/6/2015 12:12:56 pm]
X [00:00:00:2110000] [checking time: 1/6/2015 12:13:00 pm]
√ [00:00:00:2110000] [checking time: 1/6/2015 12:13:04 pm]
```

Instances status

Instance	Comp. Rules	Satisfied	Recovered	Terminated
0160000	R1:(CheckPrivilege) Exists	√	-	-
	R2:(CheckCredit) Exists	√	-	-
	R3:CheckPrivilege LeadsTo CheckCredit	√	-	-
	R3:CheckPrivilege Precedes CheckCredit	√	-	-
	R4:CheckPrivilege SegregatedFrom CheckCredit	√	-	-
	R5:CheckPrivilege USegregatedFrom CheckCredit	√	-	-
4310000	R6: (NotifyCustomer) IsUniversal	X	X	√

Fig. 7: BP-MaaS Monitoring Dashboard Screenshot.

anti-patterns. This validation and evaluation step also enabled us to identify the limitations of the approach and the potential enhancement points.

The loan approval case study used as the running scenario throughout this paper compromised 7 high-level compliance requirements (compliance constraints as they originate from various compliance sources). By applying the compliance refinement methodology in [60, 59], this has resulted in 15 organization-specific compliance requirements based on the BPMN model shown in Figure 3. Examples of these refined/internalized compliance requirements are given in Table I.

The compliance patterns used to realize the 15 requirements are the *Response* pattern, which constitutes 7 compliance rules (note that a compliance requirement can be represented by one or more compliance rules) as shown in Table I). *Exists* pattern is used in one rule, *Precedes* pattern in 3 compliance rules, *Next* and *segregationOfDuty* patterns are used in one compliance rule, respectively. *PerformedBy* resource pattern is utilized in 5 rules, *Mutual Exclusive Choice* composite pattern is used to represent one compliance rule. And *Time span and isBefore = false* real time pattern is used in one rule in conjunction with the *Response* pattern.

The second case study targets the anti-money laundering banking scenario. The U.S. Patriot act of 2001 [44] was considered as the main source of compliance requirements of this case study, which constitutes a large number of compliance requirements structured into twelve sections. As discussed previously in Section III, the interpretations of embedded requirements and encoding them in the Semantics of Business Vocabulary and Business Rule (SBVR) standard [45] as a structured natural language is ongoing work in GRCTC [20]. For this case study, we have only considered the suspicious activity detection and reporting part of the AML practices.. From the U.S. Patriot act, we found 27 compliance requirements relevant to the suspicious activity detection and reporting business process.

The compliance patterns used to realize the 27 requirements are the *Response* pattern 11 requirements, the *Exists* pattern 1 requirement, the *Performed By Role* pattern 4 requirements, the *Precedence* pattern 3 requirements, the *Absence* pattern 1 requirement, and the *Next* pattern 1 requirement. Among the 11 *Response* rules, one rule used the *time span* property whereas one other rule used the *with absence* property. Out of the three *Precedence* rules, two rules used the *with absence* property. Table II shows a excerpt of these requirements, by referring to its clause number inside the U.S. Patriot act.

Table III gives the type and number of compliance requirements covered within the case studies, and whether the case study participants were able to express these requirement effectively using compliance patterns and continuously monitor them against running business process instances by means of anti-patterns, using the prototypical implementation discussed

TABLE II: An excerpt of the Compliance Requirements relevant to the AML case study

Compliance Requirements	Source	Control	Pattern Representation
R1: Required Standards for AML MSB programs	Section $1022.210 (d)(1)(i)(A)(B)(C)(D)	Anti-money laundering programs for money services business: policies, procedures, and internal controls	R1.1: Response(Send Payment Order ,Verify Customer Identification) R1.2: Response(Send Payment Order ,Retain Supporting Documents)
R2: Identity and Reporting related provisions	Section $1022.210 (d)(1)(iv)	It is necessary that customer identification and verification includes name, date of birth, address and identification number of a person.	R2.1: Exists(Verify Customer Identification, Check{ CustomerIdentification.name is not null, CustomerIdentification.DoB is not null, CustomerIdentification.address is not null, CustomerIdentification.ID is not null })
R3: Identity and Reporting related provisions	section $1022.210 (d)(1)(iv)	It is obligatory that customer identification for amounts above $10, 000 to take place within one calendar day	R3.1: Response(Initiate Money Transfer, Verify Customer Identification, time span = before 1 day)
R4: Retention of Record of related provisions	section $1022.320(c)	It is obligatory that each money service business maintains copy of each Suspicious Activity Report-MSB filed and business record of any supporting documentation for 5 calender years.	R4.1: Absence(Delete Suspicious Transaction Records) R4.1: Next(Receive Add Doc Request, Send Add Doc)
R5: Confidentiality	Section$1022.320 (d)(ii)(A)1	It is permitted that each money service business to disclose a suspicious activity report to FinCEN or an appropriate law enforcement agency if the person involved in a suspicious transaction is not notified that a suspicious activity report has been filed.	R5.1: Precedence(Disclose Suspicious Activity Report, Process.Start, WA={ Receive Deferment Notification})

TABLE III: Categories and Numbers of Compliance Req. Covered in the Case Studies

Type of Controls		Number of Comp. Requirements: Case St.1: Loan Processing	Case St.2: Anti-money Laundering	TOTAL	Supported by our approach?: Yes	No
PROCESS	*- Control flow*	*1*	*2*	*3*	***3***	-
	- Data Requirements	*-*	*3*	*3*	***3***	-
	- Resources	*2*	*-*	*2*	***2***	-
	- Control flow- Data	*1*	*7*	*8*	***8***	-
	- Control flow- Resources	*3*	*1*	*4*	***4***	-
	- Control flow- Real time	*1*	*1*	*2*	***2***	-
	- Resource-data	*1*	*-*	*1*	***1***	-
	- Control flow-Resources-Real time	*1*	*-*	*1*	***1***	-
	- Control flow-Resources-Data	*1*	*2*	*3*	***3***	-
	- Control flow-Data-Real time	*-*	*1*	*1*	***1***	-
TOTAL		**11**	**17**	**28**	**28**	-
TECHNICAL/ MANUAL	*- Data Requirements*	*3*	*3*	8	-	***8***
	- Others	*-*	*7*	7	-	***7***
PHYSICAL		*1*	-	*1*	-	***1***
TOTAL		**15**	**27**	**42**	**28**	**16**

in Section VI. As shown in Table III, compliance requirements are classified into three distinct classes:

- *Process*: compliance requirements that are relevant to the policies and practices concerning the design and execution of business process models. Authorizations, approvals, inspections, segregation of duties applied through business tasks and other elements are examples of such requirements.
- *Technical/Manual*: are requirements that involve the use of devices or systems mainly for authentication, encryption or security purposes. Examples include firewalls and intrusion prevention/detection systems.
- *Physical*: are requirements that involve largely the institution of physical means, such as locks, fences and alarms, to guard critical assets.

As shown in Table III, the requirements that are classified as process constituted the majority. These requirements were of particular interest to us, as this type of constraints is the main target for the runtime monitoring approach introduced in this paper. They involved rules concerning mainly segregation of duties, access-rights, condition-based sequencing of activities, data processing requirements and real time constraints. For this category, we further classified it into classes corresponding to the four structural facets of business processes; i.e., control flow, data requirements, employed resources, and real-time, and their combination, as a single compliance requirements might be, for example, addressing the control flow and data aspects of BPs. Real-time constraints usually do not exist by their own, that is why it was not included as one of the subclasses of the process category. As show in Table III, 28 compliance requirements in total from the two case studies belong to this category, and they all could be effectively modelled, executed and monitored by applying our approach.

Technical requirements mainly involve constraints regarding data processing, e.g., rules that are related to the structure and integrity of the data manipulated within the processes. They typically demanded for sequential numbering of certain business objects, such as orders or invoices, or the retention of data for a specific period of time. Also, it involved the requirements of manual management reviews and reconciliations, which are inherently manual by nature. Other constraints in this category mandate the existence of authentication, encryption or security devices and/or software components, which could be checked on other enterprise systems level, but not on this business process level. Requirements from the technical and physical categories are subsequently can not be supported by our approach. In total, 28 compliance requirements out of 42 are fully supported by our approach, which represents a ratio of around 70%. Despite the limitations discussed above, we can conclude that within the process category of compliance requirements, the proposed runtime compliance monitoring approach is an effective means for expressing runtime compliance requirements in a user-friendly and abstract form, and supports the continuous monitoring of these constraints by applying a novel evaluation mechanism based on anti-patterns. We can also conclude that compliance requirements fall in the *process* category represent a major subset of the compliance requirements imposed on real-world scenarios. Future work involves intensifying this validation and evaluation step by applying our approach on a larger scale case studies in different domains.

VIII. Related Work

Compliance management has been an active research area recently from both the academic and industrial communities, given the high-cost associated with non-compliance, including business failures, bankruptcy, significant fines and even criminal penalties. In the following, prominent related-work efforts in runtime compliance monitoring is summarized and appraised against the work proposed in this paper. For a detailed comparison framework, we refer the reader to [32].

Runtime monitoring requires business process models to be reduced to some abstract representation, which are built up by collecting runtime information (e.g. exchanged messages sequences, performed activities). On the other hand, runtime monitoring also requires compliance requirements to be structurally/formally represented using a formal/structural language, e.g. LTL, CTL, ECA rules. In addition, various querying languages could also be utilized, such as *BP-Mon* [11] and XPath [62]. The actual compliance checking between abstract traces and formal rules/queries is performed by a runtime compliance checker (engine), which is usually an external component that is incorporated into the execution environment, but could also be an internal component. The checker can check the adherence to the requirements either after the execution is completed, or synchronous with the execution, following a more proactive approach. In the following, we classify related work into four categories; *graph-based* approaches, *formal-based* approaches, *XML querying* approaches and *complex-event* processing, which will be discussed in the next subsections and appraised against the work presented in this paper.

A. Graph-based Monitoring Approaches

Graph-based approaches mainly target the design-time phase of the business process lifecycle for (sub-)process models querying, substitution,and compliance checking; examples are: [29], [52], [3], [53], [17]. On the other hand,few studies have addressed runtime compliance monitoring, which include [11, 33]. *Business Process Monitoring* (*BP-Mon*) is a graphical query language proposed in [11] to visually represent monitoring requirements against BPEL models, abstracted into event traces. Graph matching techniques (*homomorphism*) are then exploited to evaluate the compliance of *completed* running BPEL instances, focusing on control-flow and timing constraints. Similarly, the study in [33] adopts a graph-based compliance rule language to capture compliance requirements, supporting sequence, data and real-time constraints, where runtime compliance checking is done *synchronously* with the execution.

B. Formal Monitoring Approaches

Influential *formal monitoring* approaches are reported in [27] , [36], [38], [9], [10], [24], [35] by founding compliance requirements on a formal/mathematical language. The study in [36] uses *Event Calculus* (EC) as the formal basis of monitored constraints against BPEL models. EC is an expressive language; however it is excessively difficult to be used. Monitoring is implemented as integrity-checking technique on *completed* executions. EC is also used in [38], however to cope with the complexity of EC, Declare language [47] is utilized as a graphical intermediate representation. Logic programming reasoning is then used to dynamically reason about partial, evolving execution traces. These approaches [7, 38] focus on control-flow and timing constraints.

Model-checking formal approaches is adopted in [7, 27, 34]. *LTL-FO+* is proposed in [27] as an extension to LTL that includes full first order quantification over data, focusing on control-flow and data requirements. In [34], *Declare* [47] is used, which is mapped into LTL, only supporting control-flow constraints, where monitoring is done synchronously with the execution. The same approach is applied in [35] using Declare and LTL to capture compliance requirements, while declarative process models are considered instead, mainly to detect conflicting compliance requirements.

Metric first-order temporal logic is used in [10], supporting past and bounded future operators. This approach [10] provides an optimized monitoring technique addressing control-flow and timing constraints, however the complexity of the adopted logic is not tackled. An extension is made in [9] to support data-constraints. The REALM model is proposed in [24] which constitutes (among others) a conceptual model and metadata. The conceptual model captures the concepts and relationships related to a certain domain (domain ontology), which are used to build compliance rules. To ensure the rigor of the framework, compliance rules are first represented formally using Past LTL, then mapped to proprietary notations. Compliance checking is also performed by a proprietary component (Active Correlation Technology (ACT)) that correlate events to detect runtime violations. The approach supports control-flow and real-time constraints.

C. Query-based and Rule-based Approaches

Prominent *XML querying* approaches are [26], [61], [54]. In [26] and [61], requirements in LTL are translated into equivalent XQuery expressions, and an XQuery engine is used to evaluate the compliance, focusing on sequence and data constraints. *BPath* [54] is proposed as an XPath extension with LTL modalities. BPath expressions are then mapped into XPath, and a native XML query engine is utilized, supporting sequence and timing constraints.

Influential Rule-based proposals include [41], [8], [14], [42], [7]. In [8], desired properties and constraints on BPEL systems are specified in WS-CoL (Web Service Constraint Language), a special-purpose assertion specification language that borrows its roots from JML (Java Modeling Language) and extends it with constructs to gather data from external sources. WSCoL are interweaved into BPEL specification, and a dedicated monitoring manager evaluates the compliance by focusing on data constraints. In [14], compliance requirements are represented in Prolog and verified against a workflow language, supporting sequence and timing constraints using a rule engine.

In [41] a generic runtime compliance management framework is proposed, which is based on a set of Dwyer′s property specification patterns [19], and provides a high-level conceptual model for compliance requirements refinements and the definition of recovery actions, as response to detected violations. The framework is realized by implementing it using BPMN models and Event-Condition-Action (ECA) rules. This approach is closely related to the work presented in this paper, however, our work relies on a wider set of novel compliance patterns; moreover, our approach addresses the four structural facets of the BP lifecycle; and we define a novel evaluation approach based on anti-patterns.

D. Complex Event Processing Monitoring Approaches

Complex Event Processing (CEP) technology is utilized in [40], [57], [64], [56]. Prominent efforts in this direction use Event Pattern Languages (EPLs) to capture relevant requirements and constraints. In [40], a model-driven engineering approach is adopted, such that a high-level DSL language is introduced for the abstract specification of compliance constraints, with support for sequence and resource constraints.

The work in [64] only considers sequential requirements, where an approach is also introduced to filter and aggregate query results to provide compact feedback on deviations. Business processes are modelled in [57] as event flows where compliance requirements are structurally represented in a conceptual graphical rule model the authors also proposed; and then a CEP engine (SARI) [39] is utilized to check the compliance, with support for sequence and timing constraints. Major approaches in this category check compliance synchronously with the execution.

E. Discussion

Guided by [28] to help validating the utility, novelty and applicability of the approach proposed in this paper, we have investigated, categorized and analyzed related work efforts in the area of Business process and Web services runtime monitoring, and aligned them against the work proposed in this paper. Table IV presents a summary and evaluation of prominent related work proposals discussed in the above sub-sections. The criteria used in the comparative analysis presented in Table IV are as follows:

- *Category*: refers to the five categories discussed above; i.e.,Graph-based, Formally-based, Query-based, rule-based and CEP.
- *Generic*: takes the value ”Yes” or ”No”, which signifies whether the approach provides a generic compliance monitoring framework.
- *BP Language*: the Business Process Language considered by the approach.
- *Observer*: indicates whether the observer that listens to event-patterns of interest is an internal or external component.
- *Runtime info*: the collected runtime information as events, against which compliance rules are evaluated.
- *Rule Support*: indicates whether the approach propose a solution to tackle with the complexity of the underlying formal/query/rule specification language. ’N/A’ means that no support is provided.
- *Rule language*: the formal/query/rule language as the formal foundation for compliance requirements specification.
- *Evaluation approach*: the runtime verification approach used.
- *Synchronous*: takes the value ”Synchronous” or ”Asynchronous”, and indicates whether the monitoring is done step-by-step synchronous with the execution, or after the execution is completed, respectively.
- *BP facets*: lists the support of the approach to the four structural facets of the BP lifecycle; i.e., control-flow, data, employed resources and real time.
- *Recovery Actions*: takes the value ”Yes”, ”No” or ”Partial”, and indicates whether the approach provides a mechanism to reason about detected violations and/or invoke (semi-) automated recovery actions. Partial means that the framework partially support this criterion; e.g., halting the execution and informing appropriate personnel, however, automated recovery actions are not supported to resolve the non-compliance anomalies, etc.

We can conclude from Table IV that the approach proposed in this paper is a generic compliance monitoring approach that could be concretized to any of the approaches in the above sub-sections. As a proof-of-concept, we have implemented it in CEP (Section VI), which is one of the possible realizations. Table IV distinguishes our work by:

1) We adopt a graphical high-level pattern-based specification compliance language, incorporating a wide range of rich compliance patterns accepted by the community. Compliance patterns are mostly used in the literature in design-time compliance checking, whilst in this paper we applied them to runtime monitoring.
2) We have implemented the adopted patterns in an intuitive graphical manner, which further ease the work of the business and compliance experts.
3) Our approach supports the four structural facets of BPs; control-flow, data, employed resources and timing. The highlighted related-work approaches only support a subset of these classes.
4) Compliance monitoring is performed step-by-step and synchronously with the executions, which is crucial for

TABLE IV: Summay and Evaluation of Related Work

Category	Approach	Generic	BP Language	Observer	runtime info	Rule support	Rule language	Evaluation Approach	Synchronous	BP facets	Recovery Actions
Graph-based	[11]	No	BPEL	Internal	Event traces (activities start /completion)	Graphical BP-Mon	Graph representation	Graph homomorphism	Synchronous	control-flow real time	No
	[33]	No	ADEPT [49]	External	Event traces (activities start /completion)	graphical	Compliance Rule Graphs	Graph Pattern matching	Synchronous	control-flow Data real time	Yes
Formally -based	[27]	No	Workflow language	External	Event traces (activities start /completion)	N/A	LTL-FO+	Model Checking	Asynchronous	control-flow Data	No
	[36]	No	BPEL	External	Event traces (activities start /completion)	N/A	Event Calculus	integrity-checking	Asynchronous	control-flow real time	No
	[38]	No	distributed control systems	External	Event traces (defined events)	Declare [47]	Event Calculus	Logic programming	Synchronous	control-flow real time	No
	[34]	No	Workflow language	External	defined event traces	Declare [47]	LTL	Model Checking	Synchronous	control-flow	Yes
	[10], [9]	No	distributed systems	External	Event traces (activities start /completion)	N/A	MFOTL	Model Checking	control-flow Data real time Data	Asynchronous	No
	[24]	No	proprietary systems	Internal	defined Event traces	Textual patterns	PLTL mapped into proprietary IBM notations	proprietary correlation engine(ACT)	control-flow Data real time	Synchronous	No
Query-based	[26], [61]	No	Web service choreograph	External	event traces (exchanged messages)	N/A	XQuery	XQuery evaluation	Asynchronous	control-flow Data	No
	[54]	No	BPEL	External	event traces (exchanged messages)	N/A	BPath	XPath evaluation	Synchronous	control-flow real time	No
Rule -based	[8]	No	BPEL	External	event traces (exchanged messages)	N/A	WS-CoL	Rule-based reasoning	Asynchronous	control-flow Data	No
	[41]	Yes	BPMN	External	Event traces (defined events)	Dwyer's Patterns [19]	ECA	Rule-based reasoning	Synchronous	control-flow Data employed resources	Yes (No impl.)
	[14]	No	Workflow language	External	Event traces (activities start /completion)	Rule template	Prolog	Rule-based reasoning	Asynchronous	control-flow real time	No
	[7]	No	BPEL	External	Event traces (activities start /completion)	RTML	Java Code	Code-based matching	Asynchronous	control-flow real time	No
CEP	[40]	No	BPMN	Internal	Event traces (activities start /completion)	textual DSL	EPL	Esper engine	Synchronous	control-flow employed resources real time	No
	[56]										
	[64]	No	BPMN	Internal	Event traces (activities start /completion)	N/A	EPL	Esper engine	Synchronous	control flow	Yes
	[57]	No	event-driven process chains model	Internal	Event flows	N/A	Compliance rule model	CEP engine (SARI) [39]	Synchronous	control-flow real time	Yes
Our Approach		Yes	BPMN	Internal	event traces (task-based and instance-based lifecycle)	visual compliance patterns	EPL (PoC)	Anti-Patterns evaluation approach	Synchronous	control-flow Data employed resources real time	Partial

a proactive compliance support. The majority of the approaches discussed check compliance on completed executions.

5) The approach presented in this paper supports event traces complying with both the task-based, and instance-based lifecycles [50] (presented in Section II-A), as opposed to other proposals, which mainly consider the start/completion of activities, or the sending/receiving of exchanged messages.
6) The evaluation technique based on the concept of anti-patterns is novel and technology independent as it doesn't assume specific technologies in place, and can be implemented in various platforms. In addition, the concept of anti-patterns could also be applied to verify compliance in other BP life cycle phases.
7) As discussed in Section VII,the utility and applicability of the approach has been validated on two real-life case studies addressing the banking domain, and the findings indicate that our approach supports a major subset of real-life compliance requirements imposed on the considered scenarios.

As discussed in Section VI, compliance violations reasoning and analysis is supported partially by the proposed framework, such that, when a runtime violation is detected, a dedicated actor is notified to take appropriate action and the violated execution is halted. Extending the proposed framework with an efficient root-cause analysis approach that reason about compliance violations, and enables the (semi-) automatic invocation of defined recovery actions is considered as an ongoing work direction.

IX. Conclusions and Future Work

Business processes form the foundation for all organizations, and as such, are impacted by industry regulations. Business process compliance management is an emergent *business need*, as it has been witnessed that without explicit BP definitions, effective and expressive compliance frameworks, organizations may face litigation risks and even criminal penalties. Therefore, Compliance management should be one of the integral parts of business process management. This paper contributes by presenting a *generic proactive* compliance monitoring framework;i.e. BP-MaaS, addressing the runtime phase of the BP lifecycle. The framework adopts a wide range of expressive high-level compliance patterns for the abstract specification of monitoring requirements. From high-level pattern expressions, corresponding runtime queries can be automatically generated for the actual monitoring. As a PoC, we have adopted the CEP technology as the structural basis of runtime queries. Compliance monitoring is then performed based on the notion of *anti-patterns*, a novel runtime evaluation approach that is technology-independent. As an instantiation artifact, anti-patterns are implemented that evaluates CEP queries against running BP instances to detect runtime compliance anomalies.

Some of the lessons learned are that managing compliance of business processes is a complex and multi-disciplinary task. It requires a multi-faceted approach to the problem involves not only technical aspects rooted in various fields that have to be bridged (such as computer science, business process management, formal methods, and legal studies) but also, social and organizational aspects as it highly involves knowledge work. No matter how sophisticated the offered solutions and the underlying technologies are, BP compliance management cannot be fully automated. Having efficient techniques and solutions in place can only facilitate and improve the quality of the work involved. As also shown by the case studies we conducted, the automated verification and monitoring of compliance are possible only for a certain segment of requirements. Technical and physical related requirements necessitate checks and controls that have to be performed manually by compliance experts.

Future research and development are on-going in several directions to enhance and fully-support the compliance monitoring framework proposed in this paper. First, defining automated recovery actions that could take place whenever a violation is detected, which would necessarily involve the notion of business transactions [46]. Second, providing an efficient technique that enables the prediction of potential runtime violations that support not only that of timing constraints, but also other compliance classes as well. This will require the application of some statistical and analytical models, such as Bayesian networks. Third, self-adapting the running BP instance once a compliance violation has occurred or is predicted to occur, to recover the impacts of the violation, and continue its execution normally after the adaptation.

Future work also includes basing the compliance monitoring framework on semantic repositories. This involves building a set of interrelated semantic ontologies (e.g. business process ontology, compliance ontology, organizational ontology, etc.) using the Ontology Web Language (OWL2.0) standard as part of a central compliance management knowledge base. This will allow us to: (i) conduct a set of preliminary structural analysis using the reasoning tools associated with these technologies, (ii) ensure the ontological alignment between compliance and business specifications, (iii)facilitate the communication between different stakeholders with diverse backgrounds and removes any ambiguity, and (iv)assists in the integration between heterogeneous systems. Last but not least, The validation of the proposed approach will be further intensified by its application on larger scale real-life case studies in different domains, such as healthcare and manufacturing, which is expected to raise some interesting challenges. For example, in the healthcare domain, physicians should be provided with higher levels of flexibility to override some compliance rules (weak constraints), as the patient treatment process is tightly related to the physicianś knowledge and judgment.

Acknowledgment

This project was funded by the National Plan for Science, Technology and Innovation (MAARIFAH) – King Abdulaziz City for Science and Technology - the Kingdom of Saudi Arabia – award number (11-INF1991-03). The authors also, acknowledge with thanks Science and Technology Unit, King Abdulaziz University for technical support

References

[1] W M P Van Der Aalst and A K A De Medeiros. Process Mining and Security : Detecting Anomalous Process Executions. In *(WISP)*, 2004.

[2] Ahmed Awad, Ahmed Barnawi, Amal Elgammal, Radwa Elshawi, Abduallah Almalaise, and Sherif Sakr. Runtime detection of business process compliance violations: An approach based on anti patterns. In *ACM SAC*, 2015.

[3] Ahmed Awad and Sherif Sakr. On efficient processing of BPMN-Q queries. *Computers in Industry*, 63(9):867–881, 2012.

[4] Ahmed Awad, Matthias Weidlich, and Mathias Weske. Specification, Verification and Explanation of Violation for Data Aware Compliance Rules. In *ICSOC/ServiceWave*, 2009.

[5] Ahmed Awad and Mathias Weske. Visualization of compliance violation in business process models. In *BPM Workshops*, 2009.

[6] R. Baldwin, M. Cave, and M. Lodge. *Understanding regulation: theory, strategy, and practice*. Oxford University Press, 2011.

[7] Fabio Barbon, Paolo Traverso, Marco Pistore, and Michele Trainotti. Run-time monitoring of instances and classes of web service compositions. In *ICWS*, 2006.

[8] Luciano Baresi and Sam Guinea. Towards dynamic monitoring of ws-bpel processes. In Boualem Benatallah, Fabio Casati, and Paolo Traverso, editors, *Service-Oriented Computing - ICSOC 2005*, volume 3826 of *Lecture Notes in Computer Science*, pages 269–282. Springer Berlin Heidelberg, 2005.

[9] David Basin, Matus Harvan, Felix Klaedtke, and Eugen Zalinescu. Monpoly: Monitoring usage control policies. In *Proceedings of the 2nd International Conference on Runtime Verification (RV 2011)*, pages 360–364, 2012.

[10] David Basin, Felix Klaedtke, Samuel Müller, and Birgit Pfitzmann. Runtime Monitoring of Metric First-order Temporal Properties. In Ramesh Hariharan, Madhavan Mukund, and V Vinay, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49–60, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[11] Catriel Beeri, Anat Eyal, Tova Milo, and Alon Pilberg. Monitoring business processes with queries. In *VLDB*, 2007.

[12] M. Bennett. Fibo: Best practice in big data. *J Bank Regul*, 14(3-4):255–268, Jul 2013.

[13] Harold Boley, Said Tabet, and Gerd Wagner. Design Rationale for RuleML: A Markup Language for Semantic Web Rules. In *SWWS*, 2001.

[14] Federico Chesani, Paola Mello, Marco Montali, Fabrizio Riguzzi, Maurizio Sebastianis, and Sergio Storari. Checking compliance of execution traces to business rules. In Danilo Ardagna, Massimo Mecella, and Jian Yang, editors, *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pages 134–145. Springer Berlin Heidelberg, 2009.

[15] EDM Council and OMG-FDTF. Financial industry business ontology (fibo). Technical report, 2013.

[16] Gero Decker, Hagen Overdick, and Mathias Weske. Oryx - an open modeling platform for the BPM community. In *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, volume 5240 of *Lecture Notes in Computer Science*, pages 382–385. Springer, 2008.

[17] Patrick Delfmann, Sebastian Herwig, Lukasz Lis, Armin Stein, Katrin Tent, and Jrg Becker. Pattern specification and matching in conceptual models - a generic approach based on set operations. *Enterprise Modelling and Information Systems Architectures*, 5(3):24–43, 2010.

[18] Chiara Di Francescomarino, Chiara Ghidini, Marco Rospocher, Luciano Serafini, and Paolo Tonella. Reasoning on semantically annotated processes. In Athman Bouguettaya, Ingolf Krueger, and Tiziana Margaria, editors, *Service-Oriented Computing ? ICSOC 2008*, volume 5364 of *Lecture Notes in Computer Science*, pages 132–146. Springer Berlin Heidelberg, 2008.

[19] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE*, 1999.

[20] A. Elgammal and T. Butler. Towards a framework for semantically-enabled compliance management in financial services. In *International workshop on Knowledge-Aware Service-Oriented Applications, ICSOC2014 workshops*, 2014.

[21] Amal Elgammal and Tom Butler. Towards a framework for semantically-enabled compliance management in financial services. In *1st International Workshop on Knowledge Aware Service Oriented Applications (KASA?15), co-located with ICSOC 2015*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014.

[22] Amal Elgammal, Oktay Turetken, Willem-Jan van den Heuvel, and Mike Papazoglou. Formalizing and applying compliance patterns for business process compliance. *Software and Systems Modeling*, pages 1–28, 2014.

[23] Opher Etzion and Peter Niblett. *Event processing in action*. Manning, 2010.

[24] Christopher Giblin, Samuel Mueller, and Birgit Pfitzmann. From regulatory policies to event monitoring rules: Towards model-driven compliance automation, 2006.

[25] Object Management Group. Business process model and notation specification 2.0.2. Technical report, 2013.

[26] Sylvain Hall and Roger Villemaire. XML Methods for Validation of Temporal Properties on Message Traces with Data. In *OTM*, 2008.

[27] Sylvain Hallé and Roger Villemaire. Runtime monitoring of message-based workflows with data. In *EDOC*, 2008.

[28] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Q.*, 28(1):75–105, March 2004.

[29] Stefan Kühne, Heiko Kern, Volker Gruhn, and Ralf Laue. Business process modeling with continuous validation. *Journal of Software Evolution and Process*, 22(7):547–566, 2010.

[30] Barbara Staudt Lerner, Stefan Christov, Leon J. Osterweil, Reda Bendraou, Udo Kannengiesser, and Alexander Wise. Exception Handling Patterns in Process-Aware Information Systems. *IEEE TSE*, 36(2), 2010.

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

[32] Linh Thao Ly, Fabrizio Maria Maggi, Marco Montali, Stefanie Rinderle-Ma, and W M P Van Der Aalst. A Framework for the Systematic Comparison and Evaluation of Compliance Monitoring Approaches. In *EDOC*, 2013.

[33] Linh Thao Ly, Stefanie Rinderle-Ma, David Knuplesch, and Peter Dadam. Monitoring Business Process Compliance Using Compliance Rule Graphs. In *OTM*, 2011.

[34] Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and W M P Van Der Aalst. Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In *BPM*, 2011.

[35] FabrizioMaria Maggi, Michael Westergaard, Marco Montali, and WilM.P. van der Aalst. Runtime verification of ltl-based declarative process models. In Sarfraz Khurshid and Koushik Sen, editors, *Runtime Verification*, volume 7186 of *Lecture Notes in Computer Science*, pages 131–146. Springer Berlin Heidelberg, 2012.

[36] Khaled Mahbub and George Spanoudakis. A framework for requirements monitoring of service based systems. In *ICSOC*, 2004.

[37] Vuk Mijovic and Sanja Vranes. A survey and Evaluation of CEP Tools. In *YUINFO*, 2011.

[38] Marco Montali, Fabrizio Maria Maggi, Federico Chesani, Paola Mello, and Wil M. P. van der Aalst. Monitoring business constraints with the event calculus. 2013.

[39] Emmanuel Mulo, Uwe Zdun, and Schahram Dustdar. Monitoring web service event trails for business compliance. In *SOCA*, pages 1–8. IEEE, 2009.

[40] Emmanuel Mulo, Uwe Zdun, and Schahram Dustdar. Domain-specific language for event-based compliance monitoring in process-driven SOAs. *Service Oriented Computing and Applications*, 7(1), 2013.

[41] Kioumars Namiri and Nenad Stojanovic. Pattern-based design and validation of business process compliance. In *Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Internet Systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I*, OTM'07, pages 59–76, Berlin, Heidelberg, 2007. Springer-Verlag.

[42] N.C. Narendra, V.K. Varshney, S. Nagar, M. Vasa, and A. Bhamidipaty. Optimal control point selection for continuous business process compliance monitoring. In *Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI 2008. IEEE International Conference on*, volume 2, pages 2536–2541, Oct 2008.

[43] OASIS. Web services business process execution language version 2.0. Technical report, 2007.

[44] FinCEN-United States Department of the Treasury. Usa patriot act, 2001.

[45] OMG. Semantics of business vocabulary and business rules (sbvr), version 1.0, 2008.

[46] M Papazoglou. Web services and business transactions. *World Wide Web*, 6(1):49–91, 2003.

[47] Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. Declare: Full support for loosely-structured processes. In *EDOC*, 2007.

[48] P. Reuter and E.M. Truman. *Chasing Dirty Money: The Fight Against Money Laundering*. 2004.

[49] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. Flexible support of team processes by adaptive workflow systems. In *Distributed and Parallel Databases*, pages 91–116, 2004.

[50] Nick Russell, W M P Van Der Aalst, Arthur H. M. ter Hofstede, and David Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In *CAiSE*, 2005.

[51] Shazia Sadiq, Guido Governatori, and Kioumars Namiri. Modeling control objectives for business process compliance. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 149–164. Springer Berlin Heidelberg, 2007.

[52] Sherif Sakr and Ahmed Awad. A framework for querying graph-based business process models. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1297–1300, New York, NY, USA, 2010. ACM.

[53] Sherif Sakr, Ahmed Awad, and Matthias Kunze. Querying Process Models Repositories by Aggregated Graph Search. In *Business Process Management Workshops*, 2012.

[54] Samir Sebahi and Mohand-Said Hacid. Business process monitoring with bpath - (short paper). In *OTM Conferences (1)*, 2010.

[55] Financial Action Task. The fatf recommendations, 2012.

[56] R. Thullner, S. Rozsnyai, J. Schiefer, H. Obweger, and M. Suntinger. Proactive business process compliance monitoring with event-based systems. In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International*, pages 429–437, Aug 2011.

[57] Robert Thullner, Szabolcs Rozsnyai, Josef Schiefer, Hannes Obweger, and Martin Suntinger. Proactive business process compliance monitoring with event-based systems. In *EDOC Workshops*, 2011.

[58] O. Turetken, A. Elgammal, W. van den Heuvel, and M. Papazoglou. Capturing compliance requirements: A pattern-based approach. *IEEE Software, special issue on Software Engineering for Compliance*, 29(3):28–36, 2012.

[59] Oktay Türetken, Amal Elgammal, Willem-Jan van den Heuvel, and Michael P. Papazoglou. Capturing compliance requirements: A pattern-based approach. *IEEE Software*, 29(3), 2012.

[60] Oktay Turetken, Amal Elgammal, Willem-Jan van den Heuvel, and Mike Papazoglou. ENFORCING COMPLIANCE ON BUSINESS PROCESSES. In *ECIS 2011 PROCEEDINGS*, 2011.

[61] Marcus Venzke. Specifications using xquery expressions on traces. *Electron. Notes Theor. Comput. Sci.*, 105:109–118, December 2004.

[62] W3C. Xml path language (xpath) 2.0 (second edition), 2011.

[63] W3C. Owl 2 web ontology language structural specification and functional-style syntax (second edition). Technical report, December 2012.

[64] M Weidlich, H Ziekow, and J Mendling. Event-based Monitoring of Process Execution Violations. In *BPM*, 2011.