Business Process Compliance Management: 
An Integrated Proactive Approach

Amal Elgammal, Faculty of Computers and Information, Cairo University, a.elgammal@fcicu.edu.eg
Samir Sebahi, LIRIS, CNRS, Université Claude Bernard Lyon 1, samir.sebahi@liris.cnrs.fr
Oktay Turetken, Technische Universiteit Eindhoven, o.turetken@tue.nl
Mohand-Said Hacid, LIRIS, CNRS, Université Claude Bernard Lyon 1, mohand-said.hacid@liris.cnrs.fr
Michael P. Papazoglou, ERISS, Tilburg University, m.p.papazoglou@uvt.nl
Willem-Jan van den Heuvel, ERISS, Tilburg University, w.j.a.m.vdenheuvel@uvt.nl

Abstract

Today’s enterprises demand a high degree of compliance of business processes to meet regulations, such as Sarbanes-Oxley and Basel I-III. To ensure continuous guaranteed compliance, compliance management should be considered during all phases of the business process lifecycle; from the analysis and design to deployment, monitoring and evaluation. This paper introduces an integrated business process compliance management framework that incorporates design-time verification and runtime monitoring approaches. The nutshell of the approach is the Compliance Request Language (CRL), which is a high-level pattern-based language for the abstract specification of compliance requirements. From CRL expressions, formal compliance rules can be automatically generated, thereby eliminating the need for business and compliance experts to learn and use complex low-level formal languages. Formalized compliance rules enable automated approaches to be used for the static verification and dynamic monitoring of business processes. An integrated prototypical tool-suite is developed as a proof-of-concept to help validating the applicability of the approaches, and validated by experiment with two real-life case studies.

Keywords: Regulatory compliance, Compliance patterns, Design-time compliance management, runtime compliance monitoring.

Introduction

Service-oriented architecture (SOA) is an integration framework for connecting loosely coupled software modules into on-demand business processes. Business processes form the foundation for SOAs and require that multiple steps occur between physically independent yet logically dependent software services (Papazoglou et al., 2007). Where business processes stretch across many cooperating and coordinated systems, possibly crossing organizational boundaries, technologies like XML and Web services are making system-to-system interactions commonplace. Ensuring the compliance of service-enabled business processes with applicable laws and regulations is a key concern that has been paid much interest particularly after some high-profile business failures and scandals, such as Enron and WorldCom. These incidents resulted in the enactment of a broad body of strict legislations, e.g. Sarbanes-Oxley act. These laws extend the long-standing requirement for public companies to maintain systems of internal controls, requiring management to certify and the independent auditor to attest to the effectiveness of those systems. Subsequently, organizations are left struggling and spending billions of dollars (Hagerty and Kraus, 2009) on compliance, mainly to avoid the risks of bankruptcy, loss of reputation and even
criminal penalties. Executives and analysts of diverse industry sectors (Ernst et al., 2010) identified regulations and compliance as one of the top business risks.

Compliance is the process of ascertaining the adherence of business processes and enterprise systems to relevant laws and regulations, which may emerge from legislation and regulatory bodies, standards and code of practices (such as, ISO 9001), internal policies and business partner contracts, e.g. service level agreements (SLA). With a preventive focus, compliance assurance activities in should be considered and integrated in two phases of the business process lifecycle:

- Design-time compliance verification: Managing and ensuring the compliance of processes at the business process (BP) analysis and design phase (before the designed processes are deployed and executed).
- Runtime compliance monitoring: Monitoring the execution of the running BP instances by tracking particular event patterns.

Design-time and runtime compliance management are preventive by nature, which means that violations are detected and mostly avoided, and in case they occur, recovery actions may semi-automatically be invoked to mitigate their impacts. Efforts in managing compliance commonly focus on either one of the aforementioned compliance assurance forms, or detective retrospective approaches. However, today’s fast and ever-changing business environment requires both design-time and runtime compliance management approaches to be integrated in accord to pursue a preventive and proactive focus. The current literature lacks a comprehensive BP compliance management framework that combines these approaches towards a lifetime compliance support.

In this paper, we propose an integrated compliance management framework that complements and integrates design-time and runtime compliance verification & monitoring approaches. At the hearth of this framework lies a high-level pattern-based compliance specification language - Compliance Request Language (CRL) - that enables compliance requirements to be specified using abstract domain-specific patterns. Expressions that are designed using CRL can be automatically transformed into formal statements/query expressions, thereby facilitating automated verification and monitoring approaches to be used. The CRL provides significant support for the specification of compliance requirements as the business and compliance experts are not required to go into the details of the underlying complex formalisms. The generated formal statements (rules) are used for automated design-time verification and runtime monitoring. That is, the business processes are verified against these statements (rules) at design-time and their executing instances are monitored at runtime. In particular, we use Linear Temporal Logic (LTL) as the formal basis of CRL for design-time compliance verification, while BPath (Sebahi and Hacid, 2010a) is used for runtime monitoring.

The rest of this paper is structured as follows: First, an introduction to the BP Compliance Management Framework is outlines. Then a running scenario is presented to exemplify the concepts throughout this paper. The Compliance Request Language (CRL) is then described. This is followed by defining the integrated prototypical implementation, which is used to validate the proposed solution in two case studies. Finally, related work in both design-time and runtime dimensions is highlighted, which is followed by conclusions and future research directions.
Integrated BP Compliance Management Framework

This Section briefly discusses important aspects of the business process compliance management framework that integrates both design-time verification and runtime monitoring of business processes. Fig 1 depicts an architectural overview of the key components of the framework. In the following Sections, we give a general overview of the design-time and runtime approaches, as well as their integration.

**Design-time Business Process Compliance Management**

As shown in Fig 1 (upper part), there are two primary abstract roles involved in this approach: (i) a business expert, who is responsible for defining and managing BPs in an organization while taking compliance constraints into account, and (ii) a compliance expert, who is responsible for refining, internalizing, specifying and managing compliance requirements stemming from external and internal sources in close collaboration with the business expert. The approach encompasses two logical repositories: the BP repository and the compliance repository, which may reside in a shared environment. BP specifications including service descriptions are stored in the BP repository, while compliance requirements and relevant concepts are stored and managed in the
Crafting Global Competitive Economies: 2020 Vision Strategic Planning & Smart Implementation

compliance repository. We assume that these specifications share the same constructs – mainly BP elements residing in the BP repository.

The BP definition involves the specification of process models using the de-facto Web Services Business Process Execution Language (WS-BPEL) (refer to the upper right hand-side of Fig 1). However, as BPEL specifications are not grounded on a formal model, they should be transformed into some formal representation to enable their automated verification against formally specified compliance rules. For this transformation, we adopted and integrated the mapping framework proposed in (Fu et al., 2004). We specifically have chosen to exploit this approach due to its support to handle rich data manipulations via XPath expressions. This allows the analysis and validation of data exchanged as messages (message contents) between participating services. Following this mapping approach, a BPEL specification is first mapped to an intermediate representation (guarded automata- GA), then to Promela code - the verification language accepted by SPIN model-checker (Holzmann, 1997) used for design-time verification.

On the other side (Part A in Fig 1), compliance management practices initiate with the refinement of compliance constraints originating from various directives into a set of organization-specific compliance requirements. This involves not only compliance but also business process domain knowledge. Our work on this part is presented in detail in (Turetken et al., 2011). A compliance expert may then apply and combine compliance patterns by using the Compliance Request Language (CRL) to render the organization-specific compliance requirements (Part B in Fig 1). This serves as an auxiliary step to refine internalized requirements into formal statements. In our case, these CRL expressions are then automatically transformed into LTL formulas (for design-time verification) and BPath queries (for runtime monitoring). Due to space limitations, the complete description of CRL is presented in (Elgammal et al., 2014).

The design-time verification of the BP specifications mainly involves checking formal BP specifications against formal compliance rules using the SPIN model-checker (Holzmann, 1997). The expected inputs to SPIN are; a Promela code that captures the behavior of a BPEL specification; and a set of LTL rules capturing relevant compliance requirements. The outcome of SPIN is a yes-no answer indicating whether LTL rules have been satisfied or violated. In case of a detected violation, an analysis on the root-causes of violations is conducted following the approach we propose in (Elgammal et al., 2010, Elgammal et al., 2012) (Part C in Fig 1). This provides the users with guidelines as remedies of how compliance deviations can be resolved. The business experts alter the BP specifications taking the root-cause analysis guidelines into consideration, which is followed by the automated re-mapping of specifications into their formal forms (GA then Promela) and the re-verification against the set of applicable compliance rules. This process iterates until all violations are resolved and a compliant BP model is produced. The part in Fig 1 tagged as ‘Part B’ represents the main focus of this paper, with respect to the design-time compliance management approach.

**Runtime Business Process Compliance Management**

The lower part of Fig 1 depicts the runtime approach for BP compliance management. The inputs to this monitoring phase (which are the outputs of the prior design-time verification phase) are: (i) a compliant BPEL model to be deployed on a BPEL engine, (ii) a set of XML queries that can be automatically generated from CRL expressions, the same way LTL rules are generated for design-time verification. During runtime, multiple BP instances may run independently on the BPEL
engine. For monitoring purposes, an instance is captured as a sequence of states (i.e. state trace). Given an initial state, a BP instance goes through a series of intermediate states, until it reaches its final state. Following the SOA paradigm, the transition from one state to the next is basically triggered by receiving or sending a message, or the assignment of instance variables (e.g. variables defined in the corresponding BPEL specification).

An external component, e.g. Execution Listener, can be deployed in the monitoring environment that keeps track of system’s activities (incoming and outgoing messages), captures and publishes events during business execution (according to the events specified in the design phase). Monitored events are then populated in the State trace and the Execution data source logs. State trace log is an XML document that records BP instances state changes whenever a message is sent or received. Each recorded state is associated with a BP instance-id and a timestamp reflecting when this transition has occurred. The state trace log only contains the sequence of execution states for each BP instance. The real execution data is stored separately in the execution data source, which contains execution variables and their respective values.

The runtime compliance monitoring is based BPath - an XML query languages. BPath extends XPath 1.0 with LTL capabilities. The advantages behind our choice of an XML query language for runtime monitoring instead of using a model-checking approach similar to the design-time verification phase are twofold: First, the resulting XPath queries can be deployed on any XML query engine, which facilitates the integration of our monitoring system to any business process framework supporting XML technologies. Second, using XPath runtime monitoring is efficient, since the computational complexity of checking an XPath 1.0 expression against XML document is shown to be in P-TIME (Gottlob et al., 2003). Additionally, XPath is not restricted on checking formulas, but also on evaluating quantitative queries, which gives us the ability to evaluate statistical indicators on the business process execution.

BPath expressions have a direct mapping to standard XPath queries (that we have introduced in (Sebahi and Hacid, 2010b), which enables us to utilize any standard XML query engine for query evaluation. Having a set of XPath queries capturing relevant compliance requirements and execution data (maintained in relevant logs), a standard XPath query engine is used to evaluate the generated XPath queries (originally generated from CRL expressions) over the State trace and Execution data source logs. The state trace log is linked to the Execution data source log where execution information can be extracted for XPath query evaluation (i.e. the message that leads to state change, its parts and a timestamp that captures the exact time when a message is sent/received).

Runtime monitoring may also involve the provision of statistics on the performance of the system. For this purpose BPath can be used to specify statistical queries to evaluate against completed BP executions stored in the State trace and Execution data source logs. Our work on monitoring process performance through statistical queries are detailed in (Sebahi and Hacid, 2010a). The runtime violations that are identified and the key performance indicators are then displayed on the Dashboard. In case a runtime compliance violation is detected, countermeasures can be enacted in order to mitigate possible negative effects (as detailed in (Daniel et al., 2009)), i.e. enforcement of controllable actions, modifying relevant BP model or the adjustment of internal policies.
Running Scenario

In this section, we introduce one of the industrial case studies that has been performed within the context of EU funded COMPAS research project, which we shall use its simplified version as a running scenario throughout this paper. The scenario involves a service-enabled loan-approval process that takes place in the banking domain. Taking into account the demands for strong regulation compliance schemes, such as Basel III, Sarbanes-Oxley (SOX) and sometimes contradictory needs of the different stakeholders, such business environments raise several interesting and challenging compliance requirements.

The process flow may be described as follows: Once a loan request has been 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 is less than €1M (million), 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 it. If the threshold is greater than €1M, the supervisor is responsible for performing the same activities instead of the post-processing clerk. Next, the manager evaluates the loan risk, after which s/he normally signs the loan form and sends the form to the customer to sign.

Table 1 shows an excerpt of the compliance requirements imposed on the loan approval process. The second column in the table presents the case-specific interpretation of the compliance requirements (refined company internal control). The third column refers to the directives (or compliance sources, i.e., laws, regulations, legislation document, etc.) from which these requirements originate.

<table>
<thead>
<tr>
<th>CR. ID</th>
<th>Refined/ Compliance Requirement</th>
<th>Directive</th>
</tr>
</thead>
</table>
| R1     | The activity ‘Customer bank privilege check’ must be segregated from ‘credit worthiness check’ | - Sarbanes-Oxley Sec. 404  
- ISO 27002-10.1.3 |
| R2     | The customer must receive an automated email notification immediately when his personal data is collected by the “Credit Bureau service”. | - 95/46/EC (Data protection directive) |
| R3     | The branch office Manager checks whether risks are acceptable and makes either the final approval or rejection of the request. | - Sarbanes-Oxley Sec. 404  
- ISO 27002-10.1.3 |
| R4     | The offer in the signed loan contract is valid for 7 working days and afterwards it is closed. | - Internal Bank Policy |

Compliance Request Language

Compliance Request Language (CRL) is a high-level pattern-based language for the abstract specification of compliance requirements. In (2008) we have analyzed a wide range of compliance legislations and frameworks including Basel II-III, Sarbanes-Oxley, IFRS, FINRA, COSO, COBIT and OCEG, and examined a variety of relevant works on the specification of associated compliance requirements. Based on our analysis, we have identified structural patterns of
frequently recurring (compliance) requirements imposed on business processes. Fig 2 presents the core elements of the CRL meta-model represented using UML class diagram. The full-fledged presentation of CRL is reported in (Elgammal et al., 2014). As shown in the figure, a CRL expression comprises compliance patterns and operands. Operands take the form of BP elements (such as activities, events, business objects, etc.), their attributes, or conditions on them, which represent the basic propositions for LTL formulas. Examples of operands are: “CheckCreditWorthiness” activity and “LoanRequest. Amount > €1M” data-based branching condition.

Fig 2. Compliance Request Language (CRL) meta-model (core)

A compliance pattern is a high-level domain-specific template used to represent frequently recurring compliance constraints. We distinguish between three main sub-classes of patterns: atomic, resource and composite patterns. Atomic pattern class is then classified into order patterns and occurrence patterns to capture compliance requirements related to the control-flow and the occurrence of BP elements. Atomic patterns are based on Dwyer’s property specification pattern systems (Dwyer et al., 1998). Due to space limitations only commonly used patterns are presented in Table 2. Resource patterns capture recurring requirements related to task assignments and authorization constraints, e.g. the typical segregation of duties constraint. Composite patterns include those that are built up from combinations (nesting) of two or more atomic/composite pattern expressions via Boolean logical operators, which enable the definition of complex requirements in terms of other compliance patterns. CRL Expressions can also be nested via Boolean operators.

CRL is formally grounded on Linear Temporal Logic (LTL) (Pnueli, 1977). LTL is a logic used to formally specify temporal properties of software or hardware designs. In LTL, each state has one possible future and can be represented using linear state sequences (paths), which correspond to describing the behavior of a single execution of a system (the system in our case is a guarded automata- GA). As described above, GA (Fu et al., 2004) is the formal abstraction model of a BPEL specification, which captures its behavior. A GA is composed of a finite number of states, transitions between those states, and activities. A state is a unique configuration of activities, information and any other relevant BP element. For each compliance pattern in CRL, a mapping rule is defined to automatically generate corresponding LTL formulas/BPath queries for design-
time verification and runtime compliance monitoring, respectively. In this mapping scheme, we use the LTL modalities: G (always), X (next time), F (eventually), U (until), W (weak until) and R (release) (Pnueli, 1977). Table 2 and Table 3 define the compliance patterns incorporated in CRL formally and informally, given P & Q as operands. In the next sub-sections, we show how CRL is used as an abstract compliance specification language for both design-time verification and runtime monitoring.

Table 2: Atomic/resource patterns and their mapping into LTL

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
<th>LTL Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>P Exists</td>
<td>P should occur at least once within the BP model</td>
<td>G(P)</td>
</tr>
<tr>
<td>P isUniversal</td>
<td>P should always be true throughout the BP model</td>
<td>G(P)</td>
</tr>
<tr>
<td>P isAbsent</td>
<td>P should never occur throughout the BP model</td>
<td>G(¬P)</td>
</tr>
<tr>
<td>P Precedes Q</td>
<td>Q must always be preceded by P</td>
<td>¬→(P &amp; Q)</td>
</tr>
<tr>
<td>P LeadsTo Q</td>
<td>P must always be followed by Q</td>
<td>G(F(Q))</td>
</tr>
<tr>
<td>P XLeadsTo Q</td>
<td>A strict case of the LeadsTo pattern- requires a P to be directly followed by Q in the next state.</td>
<td>G(P → X(Q))</td>
</tr>
<tr>
<td>P Release Q</td>
<td>The second operand Q has to be true until and including the point (state) where the first operand P first becomes true.</td>
<td>P ∨ tᵢ</td>
</tr>
<tr>
<td>t₁ PerformedBy R</td>
<td>No other role than R is allowed to perform activity t₁.</td>
<td>G(∃(c → t₁.Hole(k)))</td>
</tr>
<tr>
<td>t₁ SegregatedFrom t₂</td>
<td>Activities t₁ and t₂ must be performed by different roles and users.</td>
<td>G(c₁.Hole(k)) → ¬(c₂.Hole(k)) ∧ G(¬(c₁.Hole(k)))</td>
</tr>
<tr>
<td>t₁ BoundedWith t₂</td>
<td>Activities t₁ and t₂ must be performed by the same user</td>
<td>G(c₁.user(uₐ₁) → c₂.user(uₐ₂)) ∧ G(¬(c₁.Hole(k)))</td>
</tr>
</tbody>
</table>

Table 3: Composite patterns and their mapping rules into LTL

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
<th>Atomic Patterns Equivalence</th>
<th>LTL Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>P CoExists Q</td>
<td>The presence of P mandates that Q is also present</td>
<td>(P exists) → (Q exists) = ¬(P exists) ∨ (Q exists)</td>
<td>¬F(P) ∨ F(Q)</td>
</tr>
<tr>
<td>P CoAbsent Q</td>
<td>The absence of P mandates that Q is also absent</td>
<td>(P isAbsent) → (Q isAbsent) = ¬(P isAbsent) ∨ (Q isAbsent)</td>
<td>¬G(¬(P)) ∨ G(¬Q)</td>
</tr>
<tr>
<td>P Exclusive Q</td>
<td>The presence of P mandates the absence of Q. And presence of Q mandates the absence of P</td>
<td>(¬(P exists) ∨ (Q isAbsent)) ∧ (¬(Q exists) ∨ (P isAbsent))</td>
<td>¬(¬(F(P)) ∨ ¬G(¬Q)) ∧ ¬(¬(F(Q)) ∨ ¬G(¬P))</td>
</tr>
<tr>
<td>P MutexChoice Q</td>
<td>Either P or Q exists but not any of them or both of them</td>
<td>(P exists) Xor (Q exists) = ((P exists) A (Q isAbsent)) ∨ ((Q exists) A (P isAbsent))</td>
<td>¬F(P) ∧ G(¬Q) ∨ G(F(Q)) ∧ ¬F(Q)</td>
</tr>
</tbody>
</table>

771
Generating Formal Rules for Design-time Compliance Verification

This section exemplifies the use of CRL to enable automated generation of formal rules based on CRL expressions. This enables automated design-time compliance verification following the design-time compliance management approach described above. Table 4 presents the CRL expressions that correspond to compliance requirements R1, R2 and R3 of the running scenario that are listed in Table 1

Table 4: CRL expressions for CRs of the running scenario and generated LTL rules

<table>
<thead>
<tr>
<th>CRL Expression</th>
<th>Corresponding LTL formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 (CheckCustomerBankPrivilege SegregatedFrom CheckCreditWorthiness)</td>
<td>$G((CheckCustomerBankPrivilege \land \neg CheckCreditWorthiness) \rightarrow G(\neg CheckCreditWorthiness \land Role(R)))$</td>
</tr>
<tr>
<td>R2 (RequestBankInformation OR CheckCreditWorthiness) XLeadsTo NotifyCustomer</td>
<td>$G((RequestBankInformation \lor CheckCreditWorthiness) \rightarrow X(\neg NotifyCustomer \land CheckCreditWorthiness))$</td>
</tr>
<tr>
<td>R3 (JudgeHighRiskLoan LeadsTo (SignOfTotallyLoanContract MutexChoice DeclineDueTo HighRisk))</td>
<td>$G((JudgeHighRiskLoan \rightarrow (F((SignOfTotallyLoanContract \land \neg DeclineDueToHighRisk)) \lor F(\neg DeclineDueToHighRisk \land \neg SignOfTotallyLoanContract))))$</td>
</tr>
</tbody>
</table>

Compliance requirement R1 represents the typical segregation-of-duties (SoD) security principle, which mandates certain activities to be performed by different individuals. As the exact users that are authorized to carry out specific activities are not known until runtime, this requirement is only checked on the role level during design-time. Checking the SoD requirement on the user-level is reserved for the subsequent runtime monitoring as explained in the next section. Similarly, as ‘time’ information (the exact time that each BP element takes place) is not encoded in BPEL models, requirement R4 is considered for runtime monitoring. Regarding R2 in Table 4, customer information is collected by conducting a credit bureau service to check the credit worthiness of the customer. The generated rules, such as those listed in Table 4, are input to SPIN model-checker for their automated verification against relevant BPEL specifications (automatically mapped into Promela code (Fu et al., 2004). Next, design-time violations are analyzed for their root-causes by following the approach detailed in (Elgammal et al., 2010, Elgammal et al., 2012).

Generating Formal Rules for Runtime Compliance Monitoring
Following the same approach applied for generating design-time rules, CRL expressions are used as sources for the formal rules to be used for runtime compliance monitoring. Table 5 presents the CRL expressions that correspond to the compliance requirements R1, R2 and R4 of the running scenario (that are listed in Table 1), and their mapping into corresponding BPath queries. Since BPath queries are evaluated against XML documents (state trace and execution data source logs), the expected operands for runtime CRL expressions are XPath expressions.

Runtime monitoring complements the prior design-time compliance verification. Compliance requirements in Table 1 that could not be verified at design-time due to the absence of necessary runtime contextual information are considered in this verification phase; i.e. R4 and the runtime aspect of R1. The SoD requirement (R1) in this case is checked on the user level (complementing the role-level verification at design time) to warrant guaranteed compliance. Monitoring of requirement R4 requires temporal information that is available during runtime. In addition, requirement R2, which has been verified at design-time, is also considered for runtime compliance monitoring, due to the sensitivity of the constraint as it includes customer’s personal information, thus a high-cost associated with its violation.

Table 5: Runtime CRL expressions and generated BPath queries

<table>
<thead>
<tr>
<th>CRL Expression Corresponding BPath Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
</tr>
<tr>
<td>R2</td>
</tr>
<tr>
<td>R4</td>
</tr>
</tbody>
</table>

Prototypical Implementation

In order to help evaluating the soundness and applicability of the overall approach proposed in this paper, it is important to implement the concepts described above, including the mapping schemes (from CRL expressions into LTL/BPath) and the verification/monitoring components. We have developed a comprehensive experimental tool-suite for business process compliance management (BPCM), which integrates the following key software components:

- The Compliance Requirements Manager to define and manage compliance requirements and relevant entities in the Compliance Repository.
- The Compliance Rule Modeler (CRM) to create graphical representations of requirements using patterns and automatically transform them into formally specified rules.
- The Design-time Compliance Verification Manager (DCVM) to formulate compliance requests for verifying end-to-end business process specifications against formal rules.
- The Runtime Compliance Monitoring component to monitor the executing process instances to detect runtime compliance violations.
Each key component integrates several sub-components, such as SPIN, or WSAT open-source software tools. WSAT implements the mapping approach in (Fu et al., 2004) for transforming BPEL specifications into Promela code. CRM component is one of the key enabler of the approach proposed in this paper. It is a standalone application developed with Microsoft Visual Studio environment by using C# programming language. It implements CRL as the graphical pattern-based language, and automatically generates corresponding formal compliance rules and queries (in LTL and BPath, respectively) based on the defined mapping scheme. The process elements that are used in building CRL expressions are retrieved from the BP Repository.

Fig 3. Screenshot from the Compliance Rule Modeler (CRM)
Fig 4. Screenshot from the runtime monitoring component

Fig 3 presents a screenshot of the user interface of the CRM, which shows the graphical representation of compliance requirements R1 and R2 listed in Table 1, as well as the generated rules for the same examples. These rules are transferred and stored in the Compliance Repository, where they are retrieved and used subsequently for design-time verification and runtime monitoring. Table 5 presents a screenshot of the runtime monitoring component, which is also a C# based component including user interfaces for both the Checker component, and the Business Activity Monitoring (BAM) component. The runtime infrastructure is based on Apache Tomcat application server, Apache ODE process engine, and ActiveMQ message broker.

Case Studies

In order to observe the applicability and utility of the overall approach and concepts proposed in this paper, we conducted two case studies in the e-business and banking domains using the tools presented in above section. The cases from these domains brought challenging compliance requirements due to the strict and diverse regulations applied in these business environments. We have introduced a part of the first case study on ‘loan origination and approval’ in the Running Scenario Section, which takes place in the banking domain as the running scenario. The second case study was performed within an Internet reseller company and covered processes, such as order processing, invoicing, payments, ledger maintenance, and delivery. In total, these processes were constrained by 59 high-level compliance requirements of different concerns (such as segregation of duties, information processing, authorizations, etc.) and originating mainly from ISO/IEC 27000 (2009), Sarbanes-Oxley (2002) and internal policies. Table 1 lists examples of these requirements.

The case study team consisted two business process domain experts and three compliance experts in total, who collectively worked for the refinement of 59 high-level compliance requirements into 135 concrete constraints. Next, the team developed graphical pattern-based expressions of the constraints by using the Compliance Rule, which generated corresponding formal LTL rules and BPath queries. These compliance rules were used in the design-time and runtime compliance verification of business process specifications covered in the case studies. Not all refined constraints could be represented using patterns discussed in this paper. Out of 135 constraints,
pattern-based expressions and corresponding formal rules of 74 constraints could be effectively used for automated compliance verification and monitoring. The verification and monitoring of remaining constraints (such as those involving data integrity, detective management reviews and reconciliations, data encryptions, or data retention) has to be supplemented with manual checks for guaranteed assurance.

The first round verification of business processes took the initial versions of the BPEL specifications of the processes in both case studies; mapped them to their formal representations, and checked against generated formal compliance rules, using the Design-time Compliance Verification Manager tool. Several compliance violations were revealed and presented to the user on the Dashboard together with guidelines for resolution or warnings if applicable. In order to further exemplify the case for all applicable compliance requirements and observe the utility of the toolset, several scenarios were designed by altering BP specifications (so that they violate applicable requirements). We considered conducting empirical tests to investigate the degree of efficiency brought to the business experts as a future work, as the main objective was to validate the applicability and implementability of the overall approach. However, considering the initial positive responses from the case study participants, we expect the proposed approach to bring about advantages and contributions in the efficiency and usability of such systems.

Related Work

Design-time compliance verification and runtime monitoring are two related and complementary research areas; yet, they have been studied distinctly in the literature. In the following discussion, we highlight key efforts from both directions and compare them to the work presented here.

Temporal logic has been successfully utilized in the literature to formalize and reason about the correctness of software and hardware systems and their adherence to desired properties and constraints in diverse application domains. This includes design-time BP compliance verification. Proposals based on temporal logic use either formal specifications or abstract specifications of BPs for verifications. The core idea underpinning the approaches that use formal BP specifications includes: using a temporal logic language (e.g., LTL, CTL) to formalize compliance requirements; abstracting existing low-level business process models (e.g., modeled as BPEL) to a corresponding formal representation (e.g., finite state automata, process algebraic representation); and finally, using model-checking and analysis tools for compliance verification. For example, the study in (Liu et al., 2007) proposes a static-compliance checking framework, such that compliance requirements are first modeled using the graphical Business Property Specification Language, then LTL rules are automatically generated.

Similarly in (Awad et al., 2011), BPMN-Q is introduced as a graphical language that enables user to specify compliance requirements in a way analogous to the modeling of BP models in BPMN. From BPMN-Q, corresponding CTL formal statements can be automatically generated and a model-checking approach is also utilized for static compliance verification. Another example for such approaches is the study in (Halle et al., 2009), which extends CTL in order to capture data-dependent constraints. The design-time compliance management approach presented in this paper is distinguished from the works discussed above by a wide set of expressive compliance patterns that are driven by common industrial standards and real-life case studies. The classes of compliance patterns that are incorporated support for representing frequently used requirements regarding the control-flow, data and resource perspectives of BPs.
In temporal logic approaches that employ abstract BP specifications for compliance verification, first; business processes are modeled using an abstract high-level language. Next, compliance requirements and desired properties are formalized using a temporal logic language. Finally, model-checkers are used for verification. If the abstract model is compliant with the set of compliance rules, a corresponding BP model may be (semi-) automatically generated. For example, (Abouzaid and Mullins, 2008) employs \( \pi \)-Logic to formally represent compliance requirements. BP models are abstractly modeled using BP-Calculus. If business and compliance specifications are compliant, an equivalent BPEL program is automatically generated from the BP-calculus specification. Similar to approaches that use formal BP specifications for verification, the approaches discussed here address only the control and/or data aspects of processes. In addition, both types of approaches assume users to possess a certain level of expertise in formalisms.

Deontic logic is also common in specifying compliance constraints for design-time verification. Influential works include (Sadiq et al., 2007) and (Governatori et al., 2006), which provide the foundations of the FCL (Formal Contract Language) language focusing on business partner contracts. Compliance is then verified based on the Idealness notion. However, lack of domain-specific patterns (e.g., CRL introduced in this paper) to be used by practitioner (business/compliance experts) stands as a significant barrier for the wide adoption of these works in practice. The modeling and verification of task allocation and authorization constraints is a significant research topic, particularly in the information systems security field. The authors in (Wolter et al., 2009) extend BPMN to capture task allocation constraints to resources (roles/users). BPMN models are mapped to Coloured Petri net (CPN) and a model-checking approach is used for verification. In addition to other perspectives (control and data), the approach we propose in this paper addresses BP resource perspective as well, and introduces specific patterns for capturing such requirements.

Business process runtime compliance monitoring side also 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 (Beeri et al., 2007) and XPath. 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. 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 principle, we classify related work into four categories; graph-based approaches, formal-based approaches, XML querying approaches and complex-event processing.

Prominent graph-based proposals are (Beeri et al., 2007), (Ly et al., 2011). Business Process Monitoring (BP-Mon) is a graphical query language proposed in (Beeri et al., 2007) 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 (Ly et al., 2011) 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.

Influential formal monitoring approaches are reported in (Barbon et al., 2006), (Mahbub and Spanoudakis, 2004), (Halle and Villemaire, 2008a), (Montali et al., 2013) by founding compliance requirements on a formal/mathematical language. The study in (Mahbub and Spanoudakis, 2004) 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 techniques on completed executions. EC is also used in (Montali et al., 2013), however to cope with the complexity of EC, Declare language (Pesic et al., 2007) is utilized as a graphical intermediate representation. Logic programming reasoning is then used to dynamically reason about partial, evolving execution traces. These approaches (Barbon et al., 2006), (Montali et al., 2013) focus on control-flow and timing constraints.

Model-checking formal approaches is adopted in (Barbon et al., 2006), (Halle and Villemaire, 2008a), (Maggi et al., 2011). LTL-FO+ is proposed in (Halle and Villemaire, 2008a) as an extension to LTL that includes full first order quantification over data, focusing on control-flow and data requirements. In (Barbon et al., 2006), Past LTL is used, which is evaluated against BPEL executions, supporting sequence and timing constraints. In (Maggi et al., 2011), Declare (Pesic et al., 2007) is used, which is mapped into LTL, only supporting control-flow constraints, where monitoring is done synchronously with the execution.

Complex Event Processing (CEP) techniques are utilized in (Mulo et al., 2013), (Thullner et al., 2011), (Matthias Weidlich, 2011) such that Event Pattern Languages (EPLs) are used to capture relevant constraints. In (Mulo et al., 2013), 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 (Thullner et al., 2011) only considers sequential requirements. Business processes are modelled in (Matthias Weidlich, 2011) as event flows where compliance requirements are structurally represented in a rule model the authors proposed, and a CEP engine is utilized to check the compliance, with support for sequence and timing constraints. Major approaches in this category check compliance synchronously with the execution.

Examples of XML querying proposals are (Halle and Villemaire, 2008b), (Sebahi and Hacid, 2010a). In (Halle and Villemaire, 2008b), requirements in LTL are translated into an equivalent XQuery expression, and an XQuery engine is used to evaluate the compliance, focusing on sequence constraints. BPath which has been used in this paper (Sebahi and Hacid, 2010a) extends XPath with LTL modalities. BPath expressions are then mapped into XPath, and a native XML query engine is utilized, supporting sequence and timing constraints. The runtime monitoring approach utilized in this paper has the advantages of being more expressive by providing both hybrid logic, past and future temporal logic operators.

In addition to its advantages highlighted above, the work proposed in this paper is first in addressing and integrating the two complementary compliance verification phases (design-time and runtime) for lifetime proactive compliance support that crosscuts the four structural facets of BP; i.e., control-flow, data, employed resources and timing constraints.
Conclusions and Future Work

Industry regulations and laws impact almost every aspect of running a business. Without explicit business process definitions, effective and expressive compliance frameworks, organizations face litigation risks and even criminal penalties. Compliance management should be one of the integral parts of BP management, such that compliance requirements should be based on a formal foundation of a logical language to enable automated future reasoning techniques for verifying and ensuring BP compliance starting from the early stages of business process management.

To ensure continuous guaranteed compliance, a preventive focus should be adopted, such that design-time compliance verification should be complemented and integrated with the subsequent runtime monitoring. In this paper, we introduce the basic building blocks of a comprehensive business process compliance management framework as a major step towards a lifetime compliance support. Compliance Request Language (CRL) is central to this framework. It allows abstract specification of compliance requirements using a rich and novel set of compliance patterns, which are synthesized from a comprehensive analysis we performed in (2008). It enables capturing diverse types of compliance requirements that span the control-flow, data and employed resources structural perspectives of business processes.

From abstract CRL expressions, design-time rules (as LTL formulas) and runtime queries (as BPath expressions) are automatically generated. This significantly facilitates the adoption of the framework in business practice, as it shields the complexity of learning and using multiple low-level languages from the users. Generated LTL rules are then verified against relevant BP models following a model-checking approach for design-time verification. Similarly, generated BPath queries are evaluated against the running instances of the statically compliant BP model via an XML evaluation engine, for runtime compliance monitoring.

We have also developed an integrated tool-suite that incorporates the Compliance Rule Manager (CRM) software component, which allows the specification of compliance requirements in a drag-and-drop fashion and the automated generation of corresponding formal rules applicable for design-time verification and runtime monitoring. The approach proposed in this paper is validated in three ways: firstly, the internal and construct validity of the approach are verified by formalizing its underpinnings to validate its logical consistency. Secondly, the implementability of our approach is ascertained with an experimental integrated tool-suite. Lastly, we have explored and tested our approach with case studies drawn from industrial partners in the COMPAS EU project in which we participate. A detailed discussion of this validation step is presented in (Turetken et al., 2011) and (Turetken et al., 2012).

Our research and development work is ongoing in several directions to enhance and fully support the business process compliance management lifecycle. In particular, we are currently developing and experimenting with the offline compliance management phase that is grounded on the design-time/runtime foundation we introduced in this article. The validation of the proposed concepts, and the usability and efficacy of the tools should further be intensified by their application on various case studies and empirical tests on prospective users. In addition, ongoing work involves defining vertical compliance management solutions to cater for industry specific compliance requirements in domains such as healthcare, environment, energy and manufacturing. On-going work also involves basing the compliance management framework on semantic repositories. This involves building a set of interrelated semantic ontologies (e.g. business process ontology,
compliance ontology, etc.) using the Ontology Web Language (OWL2.0) standard as part of a central compliance management knowledge base. Last but not least, provisioning compliance solutions as service on the cloud (i.e., CaaS)

References


ELGAMMAL, A., TURETKEN, O., VAN DEN HEUVEL, W. & PAPAZOGLOU, M. 2010. Root-Cause Analysis of Design-time Compliance Violations on the basis of Property Patterns. 8th International Conference on Service-Oriented Computing (ICSOC10). USA.


