USING PATTERNS FOR THE ANALYSIS
AND RESOLUTION OF COMPLIANCE VIOLATIONS

AMAL ELGAMMAL
European Research Institute in Service Science (ERISS)
Tilburg University, K704 5000 LE, Tilburg, the Netherlands
a.f.s.a.elgammal@uvt.nl

OKTAY TURETKEN
European Research Institute in Service Science (ERISS)
Tilburg University, K729 5000 LE, Tilburg, the Netherlands
o.turetken@uvt.nl

WILLEM-JAN VAN DEN HEUVEL
European Research Institute in Service Science (ERISS)
Tilburg University, K729 5000 LE, Tilburg, the Netherlands
w.j.a.m.vdnheuvel@uvt.nl

Today’s enterprises demand a high degree of compliance of business processes to meet
laws and regulations, such as Sarbanes-Oxley and Basel II. Compliance should be
enforced during all phases of business process lifecycle, from the phases of analysis
and design to deployment, monitoring and evaluation. In this paper, a taxonomy of
compliance constraints for business processes is introduced based on the notion of com-
pliance patterns. Patterns facilitate the formal specification of compliance constraints
that enable their verification and analysis against business process models. This tax-
onomy serves as the backbone of the root-cause analysis, which is conducted to reason
about and eventually to resolve design-time compliance violations, by providing appro-
priate guidelines as remedies to alleviate design-time compliance deviations. We have
developed and integrated a set of tools to observe and evaluate the applicability of our
approach, and experiment with it in case studies.

Keywords: Regulatory compliance; compliance constraint detection and prevention;
design-time compliance management; formal compliance model; compliance patterns;
root-cause analysis.

1. Introduction

Business processes form the key fabric of all organizations, and as such, are impacted
by industry regulations. Without explicit business process definitions, flexible rule
frameworks, and audit trails that provide for non-repudiation, organizations face lit-
igation risks and even criminal penalties. Compliance regulations, such as Basel II,\(^1\)
Sarbanes-Oxley (SOX)\(^2\) and others require all organizations to review their busi-
ness processes and ensure that they meet the compliance standards set forth in the
legislation.
Compliance is the process of ascertaining the adherence of business processes 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). Nowadays, many organizations achieve compliance in an ad-hoc manner and a case-per-case basis. These ad-hoc solutions are generally handcrafted for a particular compliance problem, which hampers their reusability and evolution and limits their flexibility to rapidly adapt to ever-changing business imperatives. This makes it difficult to verify and ensure continuous guaranteed compliance and limits the possibilities of reuse. A generic compliance management approach is key to alleviate the problems of reusability and adaptability. This approach should be sustainable throughout the business process life cycle. A preventive focus is essential such that compliance is considered from the early stages of business process design, thus enforcing compliance by design. The main focus of this paper is on the design-time aspects of business process compliance verification and analysis.

It is generally acceptable that compliance constraints should be based on a formal foundation to enable automated reasoning and analysis that assist in verifying and ensuring business process compliance. However, it is a well-known phenomenon that the use of formal languages for the specification of compliance concerns creates difficulties for end-users, particularly in terms of poor usability and comprehensibility. This problem is one of the main obstacles for the utilization of sophisticated verification and analysis tools associated with these languages. To surmount such problems, we have developed and integrated a series of compliance patterns, which support shielding the complexity of logical formalisms from business and compliance experts and facilitates the specification of compliance requirements in the abstract.

In previous work, we conducted a thorough comparative analysis between a set of logical formalisms for design-time compliance specification and verification. The comparative analysis has emphasized that the temporal logic family is a robust foundation to expressing compliance concerns. This is due to its maturity and sophisticated tool support, which have been successfully applied to the verification of various large-scale systems. Based on these findings, in this paper we have used Linear Temporal Logic (LTL) as the formal foundation of compliance requirements and SPIN model-checker for their automated verification. However, the verification results of such tools usually merely consist of the list of compliance rules that have been violated. Obviously, existing practices and approaches are by far too simplistic to effectively assist users in actually resolving potential conflicts and violations. A structured approach is critical to allow business experts — many of which are unfamiliar with formal languages — formally capturing compliance rules and policies, to semi-automatically detect the root-causes of compliance anomalies, and to provide appropriate guidelines/warnings to resolve compliance violations. The key contributions presented in this paper are as follows:

- To enable design-time compliance verification and to shield the complexity of logical formalisms from users, we have introduced in an earlier work a
taxonomy of compliance patterns, which we extend in this paper with additional patterns. Compliance patterns are high-level abstractions of frequently used compliance constraints and act as an intermediate step to formal compliance specification.

- This taxonomy constitutes the backbone of the proposed root-cause analysis approach that is able to reason about design-time compliance violations. The Current Reality Tree (CRT)\(^\text{10}\) is exploited as the adopted root-cause analysis technique. In addition to presenting the root-causes/suggestions of violations to users, in this paper, we also provide the user with possible caveats that are not errors by their own, but possible hidden causes.

- To investigate the applicability and technical feasibility of our approach, a software environment comprising a set of integrated tools has been developed and employed in two case studies that deal with real-life business processes of companies in two different industry sectors.

The rest of this paper is organized as follows: In Sec. 2, we briefly outline the overall design-time compliance management approach and pinpoint the scope of the paper. Section 3 presents a running scenario. In Secs. 4 and 5, we briefly discuss the compliance patterns taxonomy and the proposed root-cause analysis approach, respectively. Section 6 describes the implementation of the approach discussing the key components of the software environment and their use in the case studies. Related work is summarized in Sec. 7. Finally, conclusions and outlook are highlighted in Sec. 8.


Figure 1 depicts an overview of the key practices and components of our design-time business process compliance management approach, and highlights the parts that outline the scope of this paper. The approach depicted in this figure is a refined version of the generic approach that we have previously introduced in Ref. 11.

There are two primary abstract roles involved in this approach: (i) a business expert, who is responsible for defining and managing business processes 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 business process repository and the compliance requirements repository, which may reside in a shared environment. Process models including service descriptions are defined and maintained in the business process repository, while compliance requirements and all relevant concepts are defined, maintained and organized in the compliance requirements repository. These repositories foster the reusability of business and compliance specifications. We assume that these two specifications (business processes and
compliance requirements) share the same constructs — mainly business process elements residing in the business process repository.

The business process definition (the right-hand side of Fig. 1) involves the specification of process models using the de-facto Web Services Business Process Execution Language (WS-BPEL). 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. The automated mapping of process specifications into a formal representation have been intensively studied in the literature (e.g. Refs. 12–15). For this transformation, we have adopted and integrated the mapping framework proposed in Ref. 15. 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 between participating services. Other BPEL mappings (e.g. Refs. 17 and 18) abstract away the data content, which limits the types of analysis that can be performed on data. The support to data-dependent constraint is particularly important within the compliance context, which are likely to recur in various compliance sources. Following this mapping approach, a BPEL specification is first mapped to an intermediate representation (guarded automata — GA), and then to Promela code; the verification language accepted by SPIN model-checker. A brief description of this BPEL mapping is presented in Sec. 6.2.

On the other side (left-hand side in Fig. 1), compliance management practices initiate with the refinement of compliance constraints originating from various compliance sources into a set of organization-specific compliance requirements (Part A in Fig. 1). This involves not only compliance but also business process

---

domain knowledge. Our work on this part of the approach is presented in Ref. 19. A compliance expert may apply and combine compliance patterns to render the organization-specific compliance requirements (Part B in Fig. 1). This serves as an auxiliary step to refine internalized compliance requirements into formal statements (as LTL formulas for our case). These pattern-based expressions are then automatically transformed into LTL formulas. In Sec. 4, we briefly discuss about this mapping process.

The verification of business process specifications (Part C in Fig. 1) mainly involves checking formal business process specifications against formal compliance rules using SPIN model-checker. SPIN is a popular open-source software tool that is intensively used in both academia and industry for the formal verification of large-scale distributed software systems. 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 list of LTL rules that have been violated or satisfied with respect to the process specification. In case of violations, the business experts alter the process specifications taking the root-cause analysis guidelines into consideration, which is followed by the automated remapping of specifications to their formal forms (GA and Promela) and re-verification against the set of applicable compliance rules. This process iterates until all violations are resolved.

The part in Fig. 1 that is enclosed with dotted lines and tagged as “Part D”, represents the main focus of this paper; more specifically, the root-cause analysis of design-time compliance violations, which we have previously introduced in Ref. 9. In this paper, we have revised and appropriately extended our approach together with its implementation and its use in case studies involving industrial companies.

3. Running Example

The “Loan Origination and Approval” process that takes place in the banking domain is used as the running example throughout this paper. The process is based on one of the industrial case studies conducted in the EU funded COMPAS research project. A simplified, yet realistic process model is depicted in Fig. 2 using BPMN (Business Process Model and Notation) BPMN is used in this Section for illustration and presentation purposes; however as mentioned in Sec. 2, we consider BPEL for business process specifications.

The process flow may be described as follows: Once a customer loan request is received, the credit broker checks if customer’s privileges are suspended, then accesses the customer information and checks whether all loan conditions are satisfied. If the loan amount is less than 1 M (million) 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 it. If the credit amount is greater than € 1M, the supervisor is responsible for performing

\[1^{st}COMPAS official web site: http://www.compas-ict.eu/project.php.\]
the same activities instead of the post-processing clerk. Finally, the manager evaluates the loan risk, after which she normally signs the loan form and sends the form to the customer to sign.

Table 1 shows excerpts of the compliance constraints relevant to this scenario. The first and second columns of the table allocate a unique reference and case scenario specific interpretation of the compliance constraints (refined company internal compliance constraint), respectively. The third column shows the high-level compliance constraint as they are specified in sources. Finally, the fourth column refers to the associated compliance source(s).

Table 1. An excerpt of the relevant compliance requirements.

<table>
<thead>
<tr>
<th>ID</th>
<th>Refined Compliance Constraint</th>
<th>(High-Level) Compliance Constraint</th>
<th>Compliance Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>The activity “Customer bank privilege check” should be segregated from “credit worthiness check”</td>
<td>Duties (in loan processing) should be adequately segregated</td>
<td>Sarbanes-Oxley Sec. 404, ISO 27002-10.1.3</td>
</tr>
<tr>
<td>R2</td>
<td>The customer should receive an automated email notification directly after her personal data is collected by the “Credit Bureau service”</td>
<td>Customer’s personal data should be handled confidentially</td>
<td>95/46/EC (Data protection directive)</td>
</tr>
<tr>
<td>R3</td>
<td>The branch office manager should check whether risks are acceptable and should make either the final approval or rejection of the request</td>
<td>Duties (in loan processing) should be adequately segregated</td>
<td>Sarbanes-Oxley Sec. 404, ISO 27002-10.1.3</td>
</tr>
</tbody>
</table>
4. Compliance Patterns and Compliance Constraints Taxonomy

This section presents a taxonomy of pattern-based compliance constraints for business processes. As shown in Fig. 3, the compliance pattern is the core element of the taxonomy, and each pattern is a sub-type of it. The compliance pattern is sub-divided into two main sub-classes of patterns; namely atomic and composite. Composite patterns are built up by combining atomic patterns via Boolean operators, which can capture complex compliance requirements. The lower part of Fig. 3 presents the set of atomic patterns. The shaded atomic patterns are adopted from Dwyer’s property specification pattern system. A pattern expression comprises compliance patterns and operands. Operands take the form of business process elements (such as activities, events, business objects, etc.), their attributes, or conditions on them, which represent the basic propositions for forming LTL formulas. Examples of operands are “CheckCreditWorthiness” activity, “Loan.Amount > €1M” data-based branching condition in Fig. 2.

Recall from Sec. 2 that GA is the formal abstraction model of a BPEL specification, which captures its behavior. Informally, 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. Activities cause the transition from one state to the next.

Compliance patterns are formally grounded on LTL. In LTL, each state has one possible future and can be represented using linear state sequences (paths), which corresponds to describing the behavior of a single execution of the system. In the mapping scheme of compliance patterns into LTL, we use the LTL

![Fig. 3. Compliance constraints taxonomy based on compliance patterns.](image-url)
modalities: $G$ (always), $X$ (next time), $F$ (eventually), $U$ (until), $W$ (weak until) and $R$ (release). Table 2 presents an excerpt of some atomic and composite patterns and their mapping rules into LTL, given $P \& Q$ as operands (refer to Ref. 9 for a complete list of compliance patterns, their semantics and the mapping rules into LTL). The $XLeadsTo$, $Release$ and $OrderedSoD$ patterns are introduced in this paper. For the composite patterns that are presented in Table 2 (e.g. $OrderedSoD$, $CoExists$, etc.), their atomic patterns equivalences are also presented in the third column of the table.

4.1. Exemplifying the use of compliance patterns

Following the compliance constraints taxonomy introduced above, constraints $R_1$–$R_3$ (in Table 1) of the Loan Origination and Approval process can be represented using patterns as shown in Table 3. $R_1$ represents a restrict case of the segregation of duties (SoD) compliance constraint, aiming to distribute the authority in a process into two or more individuals and for the involved activities to take place respecting a specific order. We use the $OrderedSoD$ compliance pattern to capture this requirement. Regarding $R_2$, customer information is collected by the credit bureau service to check her credit worthiness. In case the loan requester is already a customer of the bank, customer’s personal information can be directly accessed from the bank database by invoking “request bank information” activity. $R_3$ checks if activity $JudgeHighRiskLoan$ is performed by the manager, which is followed by either activity $SignOfficiallyLoanContract$ or $DeclineDueToHighRisk$, using $LeadsTo$ and $MutexChoice$ compliance patterns.

5. Root-Cause Analysis of Design-Time Compliance Violations

A compliance violation in a business process definition may occur due to a variety of reasons and it is of upmost importance to provide the compliance/business expert with meaningful feedback that reveals the root-causes of these violations and aids their resolution. This feedback should contain a set of rationale explaining the underlying reasons and what strategies can be undertaken as remedies. Based on the compliance constraint taxonomy presented in Sec. 4, we have analyzed and formalized possible root-causes of violations.

To achieve this goal, we adopt the CRT technique from Goldratt’s Theory of Constraints (TOC). A CRT is a statement of a core problem and the symptoms that arise from it. It maps out a sequence of causes and effects from the core problem to the symptoms. If the core problem is resolved, each of the symptoms may be remedied. Operationally the process works backwards from the apparent undesirable effects or symptoms to uncover or discover the underlying core causes. The CRT has been chosen due to its simplicity and the visual representation of causes and effects. A CRT usually starts with a list of problems called Undesirable Effects (UDEs), which represent negative or bad conditions. The key question begins with “why has the violation occurred?” (root of the tree). The answer to
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
<th>Atomic Pattern</th>
<th>LTL Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$ Exists</td>
<td>$P$ should occur at least once within the BP model</td>
<td>NA</td>
<td>$F(P)$</td>
</tr>
<tr>
<td>$P$ is Universal</td>
<td>$P$ should always be true throughout the BP model</td>
<td>NA</td>
<td>$G(P)$</td>
</tr>
<tr>
<td>$P$ Precedes $Q$</td>
<td>$Q$ must always be preceded by $P$</td>
<td>NA</td>
<td>$\neg Q WP$</td>
</tr>
<tr>
<td>$P$ LeadsTo $Q$</td>
<td>$P$ must always be followed by $Q$</td>
<td>NA</td>
<td>$G(P \rightarrow F(Q))$</td>
</tr>
<tr>
<td>$P$ XLeadsTo $Q$</td>
<td>Represents a strict case of the LeadsTo pattern, which requires a $P$ to be</td>
<td>NA</td>
<td>$G(P \rightarrow X(Q))$</td>
</tr>
<tr>
<td></td>
<td>to be directly followed by $Q$ in the next state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P$ Release $Q$</td>
<td>The second operand $Q$ has to be true until and including the point (state)</td>
<td>NA</td>
<td>$PRQ$</td>
</tr>
<tr>
<td></td>
<td>where the first operand $P$ first becomes true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P$ SegregatedFrom $Q$</td>
<td>BP activities $P$ and $Q$ should be assigned to different roles</td>
<td>NA</td>
<td>$G(P \cdot Role(R) \rightarrow G(\neg(Q \cdot Role(R))))$</td>
</tr>
<tr>
<td>$P$ OrderedSoD $Q$</td>
<td>BP activities $P$ and $Q$ should be assigned to different roles and</td>
<td>$(P \text{ Exists}) \land (Q \text{ Exists})$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>should occur in the sequence $P, Q$</td>
<td>$\land (P \text{ Precedes } Q)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\land (P \text{ LeadsTo } Q)$</td>
<td>$\land (P \text{ SegregatedFrom } Q)$</td>
<td></td>
</tr>
<tr>
<td>$P$ CoExists $Q$</td>
<td>The presence of $P$ mandates that $Q$ is also present</td>
<td>$(P \text{ Exists } \rightarrow Q \text{ Exists}) = $</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\neg(P \text{ Exists } \lor (Q \text{ Exists})$</td>
<td>$\neg F(P) \lor F(Q)$</td>
<td></td>
</tr>
<tr>
<td>$P$ CoAbsent $Q$</td>
<td>The absence of $P$ mandates that $Q$ is also absent</td>
<td>$(P \text{ is Absent } \rightarrow (Q \text{ is Absent}) = $</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\neg G(\neg P) \lor G(\neg Q)$</td>
<td>$\neg G(\neg P) \lor G(\neg Q)$</td>
<td></td>
</tr>
<tr>
<td>$P$ Exclusive $Q$</td>
<td>The presence of $P$ mandates the absence of $Q$, And presence of $Q$</td>
<td>$(\neg(P \text{ Exists}) \lor (Q \text{ is Absent})) \land $</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mandates the absence of $P$</td>
<td>$(\neg(Q \text{ Exists}) \lor (P \text{ is Absent}))$</td>
<td></td>
</tr>
<tr>
<td>$P$ CoRequisite $Q$</td>
<td>Either $P$ and $Q$ should exist together or to be absent together</td>
<td>$(P \text{ Exists}) \iff (Q \text{ Exists}) = $</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(P \text{ Exists}) \land (Q \text{ Exists}) \lor $</td>
<td>$(F(P) \lor F(Q)) \lor G(\neg(P) \land G(\neg(Q))$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(P \text{ is Absent}) \land (Q \text{ is Absent})$</td>
<td>$(F(P) \lor F(Q)) \lor G(\neg(P) \land G(\neg(Q))$</td>
<td></td>
</tr>
<tr>
<td>$P$ MutexChoice $Q$</td>
<td>Either $P$ or $Q$ exists but not any of them or both of them</td>
<td>$(P \text{ Exists}) \text{ xor } (Q \text{ Exists}) = $</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(P \text{ Exists}) \land (Q \text{ is Absent}) \lor (Q \text{ Exists}) \land (P \text{ is Absent})$</td>
<td>$(F(P) \land G(\neg(Q)) \lor (F(Q) \land G(\neg(P)))$</td>
<td></td>
</tr>
</tbody>
</table>
Table 3. Pattern based expressions of the example constraints.

<table>
<thead>
<tr>
<th>ID</th>
<th>Refined Compliance Constraint</th>
<th>Pattern Based Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>The activities “Customer bank privilege check” and “credit worthiness check” should occur sequentially and should be segregated from each other</td>
<td>(CheckCustomerBankPrivilege OrderedSoD CheckCreditWorthiness)</td>
</tr>
<tr>
<td>R2</td>
<td>The customer should receive an automated email notification directly after her personal data is collected by the “Credit Bureau service”.</td>
<td>(RequestBankInformation XLeadsTo NotifyCustomer) And (CheckCreditWorthiness XLeadsTo NotifyCustomer)</td>
</tr>
<tr>
<td>R3</td>
<td>The branch office Manager should check whether risks are acceptable and should make either the final approval or rejection of the request</td>
<td>(JudgeHighRiskLoan.Role (“Manager”) LeadsTo (SignOfficiallyLoanContract MutexChoice DeclineDueToHighRisk))</td>
</tr>
</tbody>
</table>

this question generates child-(ren) of the UDE under consideration. For each child, which might be a UDE by itself, the same “why” question is applied, and the answer is depicted as a deeper level in the tree. This process continues iteratively until the UDE under consideration is the root-cause(s) of the problem (leaf rectangles of the tree). Incoming connections to an UDE from its children are connected via logical “or” operator; unless otherwise specified as “and”.

For each pattern given in the taxonomy (in Fig. 3), we analyzed the root-causes of its violation by constructing the corresponding CRT. The violation of a compliance pattern represents the UDE of the CRT. Due to space limitation, we do not present all the CRT corresponding to each pattern given in the taxonomy (in Fig. 3).

5.1. Current reality trees for compliance patterns

Figure 4 presents the CRTs for some of the atomic patterns presented in Fig. 3. We have introduced the CRTs for LeadsTo and isUniversal patterns (among all other compliance patterns) in Ref. 9, which have been extended in this paper by the concept of caveats (details are given below). The CRTs for XLeadsTo and Release are introduced in this paper.

As shown in Fig. 4, effects/causes are drawn in CRTs using rectangles. The root of each CRT represents a UDE. For our purpose, a UDE is a violation to a specific pattern. Hence, the root of each tree represents a violation to this pattern. Effects/Causes are connected to each other using arrows forming a hierarchical tree of effects/causes. The root-causes are depicted as leaf rectangles of the tree. In Fig. 4, a state sequence is denoted by $S_1, S_2, \ldots, S_n$; where $S_1$ is the start state, $S_n$ is the end state, and there is a transition relation from each $S_i$ to $S_{i+1}$, where $1 \leq i < n$. $S_k, S_y$ and $S_m$ are intermediate states, where $1 \leq k, y, m \leq n$.

For example, as shown in Fig. 4, the violation to “(P LeadsTo Q)” pattern is considered as the UDE of the LeadsTo CRT. Deeper levels in the tree are guided by
answering the same “why” question: Why \((P \text{ LeadsTo } Q)\) is violated? The answer to this question is: Because \((P \text{ Exists is satisfied})\) and \((Q \text{ Exists is violated})\) after the occurrence of \(P\). This is depicted as the second level of the tree. The same “why” question is applied to the \textit{UDE} under consideration and the analysis continues until the root-causes of the problem, i.e. the leaves (depicted as rectangles) of the tree are reached. As reflected by the \textit{LeadsTo} CRT, the root-cause of its violation is mainly because \(Q\) does not exist after \(P\) (assuming that \(P\) exists). In addition to this cause, it might be the case that \(Q\) exists before \(P\). Although this situation does not violate the “\textit{LeadsTo}” pattern, it might give an indication of misplacement in the specification that might have acted as a hidden reason that led to the violation. We present these cases as caveats (or warnings) for the users. For example, the user might decide to swap the occurrence of \(P\) and \(Q\) to resolve the violation. To model and distinguish possible hidden causes from definite root-causes, in this paper we extend the CRT notation with rounded rectangles, which represent the checks for identifying hidden causes, i.e. the cases which do not constitute the definite causes of violations but might act as underlying likely reasons for it.

A CRT presents all possible causes of a violation. By traversing the corresponding CRT of the violation to a specific pattern, there might be more than one reason (root-cause) underpinning this violation. Our goal is to provide the user only with the valid reason(s) and pertinent suggestive guidelines/caveats. For example, by traversing the CRT of the violation to the \textit{isUniversal} pattern as shown in Fig. 4, the root-cause(s) of the violation is that BP element \(P\) is false at one or more
states in $S_1, S_2, \ldots, S_n$. Only the states where this violation has occurred will be communicated to the user. In case the operand in the leaf rectangle (root-cause) is a pattern expression by itself, it is replaced by its corresponding CRT. This process iterates continuously until all operands in the leaf rectangles of the tree are atomic BP elements.

Composite patterns are built up by nesting atomic patterns via Boolean operators. Therefore, the CRTs for composite patterns are built up by composing the CRTs of the constituting atomic patterns, capturing the semantics of the used Boolean operator(s). For example, the CRTs of Exclusive and MutexChoice composite patterns are presented in Fig. 5. The MutexChoice pattern is an “Xor” composition between two atomic patterns: $(P \text{ Exists}) \text{ Xor} (Q \text{ Exists})$. The semantics of the Xor Boolean operator mandates that one and only one of its operand should hold (not neither of them nor both of them), which is captured in the third level of the corresponding CRT. The CRTs of Boolean operators is fully presented in Ref. 9. Current reality trees of Boolean operators are instantiated and combined with each other and with atomic patterns CRTs to build up the CRTs of composite patterns.

5.2. Application of the root-cause analysis approach on the running example

Figure 6 presents the CRTs of the violations to $R_1$ and $R_2$ of the running scenario (Table 1). The CRT of the violation to $R_2$ constitutes the analysis of $XLeadsto$ compliance pattern. The CRT of the violation to the (design-time) restricted

![Diagram](image-url)

Fig. 5. CRTs for exclusive and MutexChoice composite patterns.
Fig. 6. CRTs for the violation to \( R_1 \) and \( R_2 \).
Table 4. Generated LTL statements based on CRTs.

<table>
<thead>
<tr>
<th>Requirement Description</th>
<th>Pattern-Based Expression</th>
<th>Formal (LTL) Statement</th>
<th>Violation Description/Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: The activities “Customer bank privilege check” and “credit worthiness check” should occur sequentially and should be segregated from each other</td>
<td>( (\text{CheckCustomerBankPrivilege OrderedSoD CheckCreditWorthiness}) )</td>
<td>( F(\text{CheckCustomerBankPrivilege}) )</td>
<td>“CheckCustomerBankPrivilege” exists is violated. Add activity “CheckCustomerBank Privilege” before “CheckCreditWorthiness”.</td>
</tr>
<tr>
<td>ID</td>
<td>Statement Description</td>
<td>Formal (LTL) Statement</td>
<td>Violation Description/Remedy</td>
</tr>
<tr>
<td>----</td>
<td>----------------------</td>
<td>------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>R1.1</td>
<td>“CheckCustomerBankPrivilege” should exist</td>
<td>( F(\text{CheckCustomerBankPrivilege}) )</td>
<td>“CheckCustomerBankPrivilege” exists is violated. Add activity “CheckCustomerBank Privilege” before “CheckCreditWorthiness”.</td>
</tr>
<tr>
<td>R1.2</td>
<td>“CheckCreditWorthiness” should exist</td>
<td>( F(\text{CheckCreditWorthiness}) )</td>
<td>“CheckCreditWorthiness” exists is violated. Add activity “CheckCreditWorthiness” after “CheckCustomerBankPrivilege”.</td>
</tr>
<tr>
<td>R1.3</td>
<td>CheckCreditWorthiness shall precede “CheckCustomerBank Privilege”</td>
<td>( G((\neg \text{CheckCreditWorthiness}) W (\text{CheckCustomerBankPrivilege})) )</td>
<td>Warning: “CheckCustomerBank Privilege” is true after “CheckCreditWorthiness”. You might need to swap them.</td>
</tr>
<tr>
<td>R1.4</td>
<td>“CheckCustomerBankPrivilege” shall lead to “CheckCreditWorthiness”</td>
<td>( G((\text{CheckCustomerBankPrivilege}) \rightarrow F(\text{CheckCreditWorthiness})) )</td>
<td>Warning: “CheckCustomerBankPrivilege” is true after “CheckCreditWorthiness”. You might need to swap them.</td>
</tr>
<tr>
<td>R1.5</td>
<td>“CheckCustomerBankPrivilege” and “CheckCreditWorthiness” should be segregated</td>
<td>( G((\text{CheckCustomerBankPrivilege}.\text{Role (Role}}1) \rightarrow G((\neg (\text{CheckCreditWorthiness}.\text{Role (Role}}1))) )</td>
<td>The same role can’t perform both “CheckCustomerBankPrivilege” and “CheckCreditWorthiness”. Assign different roles to these activities.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement Description</th>
<th>Pattern-Based Expression</th>
<th>Formal (LTL) Statement</th>
<th>Violation Description/Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2: The customer should receive an automated email notification directly after her personal data is collected by the “Credit Bureau service”</td>
<td>( (\text{RequestBankInformation XLeadsTo NotifyCustomer}) \ AND \ (\text{CheckCreditWorthiness XLeadsTo NotifyCustomer}) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Statement Description</td>
<td>Formal (LTL) Statement</td>
<td>Violation Description/Remedy</td>
</tr>
<tr>
<td>----</td>
<td>----------------------</td>
<td>------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>R2.1</td>
<td>“RequestBankInformation” should immediately lead to “NotifyCustomer”</td>
<td>( G(\text{RequestBankInformation} \rightarrow X(\text{NotifyCustomer})) )</td>
<td>Add activity “NotifyCustomer” immediately after “RequestBankInformation”.</td>
</tr>
<tr>
<td>R2.2</td>
<td>“CheckCreditWorthiness” should immediately lead to “NotifyCustomer”</td>
<td>( G(\text{CheckCreditWorthiness} \rightarrow X(\text{NotifyCustomer})) )</td>
<td>Add activity “NotifyCustomer” immediately after “CheckCreditWorthiness”.</td>
</tr>
</tbody>
</table>
formal representations from BPEL specifications implementing the BPEL mapping approach proposed in Ref. 15. The BPEL mapping approach underpinning WSAT is briefly described in Sec. 6.2. LTL rules (generated from CRM) and formal BPEL representation (generated from WSAT) are then passed as inputs to the subsequent DCVM component. Design-time compliance verification manager is used for verifying BP compliance and analyzing the root-causes of violations, is at heart of this paper's subject matter and is discussed in Sec. 6.3.

6.1. **Compliance Rule Manager (CRM)**

Compliance rule manager is a standalone component used for modeling compliance requirements with visual pattern-based expressions, and automatically generating corresponding formal compliance rules based on the mapping scheme between patterns and the selected formal language (LTL) taking also the CRTs into consideration. It is built in Microsoft Visual Studio environment using C# programming language. The process elements that are used in building the pattern-based expressions are retrieved from the Business Process Repository. In case the process elements that are required to build expressions do not exist in the repository, CRM is used to create these elements and store them in the Business Process Repository for possible future use in business process or compliance constraint specifications. We provide in Ref. 19 brief information regarding the overall architecture and functionalities of the CRM.

6.2. **Web Service Analysis Tool (WSAT)**

Web service analysis tool is an advanced open source tool that implements the BPEL mapping approach proposed in Ref. 15. Based on this mapping framework, a BPEL specification is first abstracted to a conversation protocol (intermediate representation), which is a Guarded Automaton (GA) representing the global sequence of messages exchanged between participating services. Guarded Automaton is an
FSA augmented with unbounded queues for incoming messages. Guards can be specified on transitions, which are represented as XPath expressions.

Next, GA is mapped to Promela code. Promela (program meta language) is the input language accepted by SPIN model-checker. Having BPEL specifications specified as GA as an intermediate step decouples the BP specification languages and formal verification tools from the translator. In addition, it enables the application of other static analysis techniques, e.g., synchronizability and realizability analysis (we refer the reader to Refs. 15 and 23 for more details about these techniques). Figure 8 presents a snapshot of SPIN tool displaying the Promela code that corresponds to the running example (Loan Approval scenario) described in Sec. 3. This Promela code has been automatically generated from WSAT.

To be able to verify resource allocation and authorization constraints, such as the segregation of duties requirement exemplified as R1 in Table 1, it is necessary to formally represent some important business process elements along with their relationships, i.e., Roles, Actors and Tasks (tasks correspond to business process activities). We are following the approach proposed in Ref. 24 for this purpose. More precisely, we assume the existence of three sets, i.e., Actors, Roles and Tasks. The set (Actors X Roles X Tasks) represents the allocation of tasks to particular actors assuming specific roles. The set Roles is a partial order set forming a role hierarchy. This means that a role inherits the privileges assigned to its related lower-level roles in the role hierarchy. Based on this representation framework, the restrict SoD requirement (as exemplified in R1 in Table 1) can be extended to check for the particular actors that perform specific activities, other role-based SoD variants and binding of duties requirements. If a model-checking approach
is utilized for the subsequent runtime verification phase, these SoD rules (on both Role and Actor levels) can be reused to check their satisfaction during runtime (which is left outside the scope of this paper). When mapping BPEL to Promela, Roles are captured from the “PartnerRole” attribute of the “partnerLink” element in the BPEL specification. Partner links are used to link a BPEL specification to its interacting services. The corresponding partner link is then linked to BPEL basic activities (i.e. Invoke, Receive, and Reply) via the “partnerLink” attribute.

### 6.3. Design-time Compliance Verification Manager (DCVM)

Design-time compliance verification manager is a web-based environment (http://eriss.uvt.nl/compas), coded in “PHP” (scripting language). It interacts with the Compliance and Business Process repositories, which share the same environment running Oracle database (ver.9i). As shown in Fig. 7, DCVM comprises two sub-components; first, the Verification Handler enables users to formulate compliance verification requests. It retrieves formal compliance rules (in LTL) from the Compliance Repository and business process specifications encoded as Promela code (using the WSAT), and feeds those to SPIN for formal verification.

Second, the Verification Handler retrieves the checking results from SPIN, reports the root-causes based on the approach presented in the preceding sections and posts the verification results and causes of violations — together with their remedies and caveats — to the design-time verification Dashboard. Figure 9 presents a screenshot from the Dashboard component of the DCVM reflecting how the results of the root-cause analysis are communicated to the business/compliance experts. Only relevant remedies extracted from traversing the appropriate CRTs are displayed in the last column of the table in the user interface (“Result Description/Remedy” column). The user interface exemplifies the case of Loan Origination and Approval business process, where $R_1$ is violated.

### 6.4. 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 sections. The cases from these domains brought challenging compliance requirements due to the strict and diverse regulations applied in these business environments. Section 3 has introduced a part of the first case study on “loan origination and approval” that 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 in Sec. 3 lists examples of these requirements.
The case study team consisted of two business process domain experts and three compliance experts in total, who collectively worked for the refinement of 59 high-level compliance requirements into 127 concrete constraints. Next, the team developed graphical pattern-based expressions of the constraints by using the CRM (Sec. 6.1), which generated corresponding formal rules. These compliance rules were used in the design-time 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 127 constraints, pattern-based expressions and corresponding formal rules of 90 constraints could be effectively used for automated compliance verification. The verification of remaining constraints (such as those involving data integrity, completeness and 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 DCVM. Several compliance violations were revealed and presented to the user on the Dashboard together with guidelines for resolution or warnings if applicable. These violations — some of which are shown in Fig. 9 as examples — were subsequently resolved by the business process experts in accordance with the guidelines. 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.

7. Related Work

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 business process compliance analysis and verification. For example, the study in Ref. 12 proposes a static-compliance checking framework that exploits various model transformations. Compliance constraints are formally grounded on LTL, and NuSMV2 model checker is used for compliance checking. In Ref. 26, past LTL (PLTL) is utilized, where properties about the past can be represented. In Ref. 14, real-time temporal object logic is utilized for the formal specification of compliance requirements based on a pre-defined domain ontology. In Ref. 27, an extension to CTL is proposed to capture data-dependent constraints; i.e. CTL-FO+. Next, NuSMV model-checker is utilized to ensure their conformance against BPEL specifications.

Other business process compliance verification approaches start by the abstract specification of business process models. For example, Ref. 13 employs $\pi$-Logic to formally represent compliance requirements. Business process models are abstractly modeled using BP-Calculus, which is a formal business process modeling language based on $\pi$-Calculus. If business and compliance specifications are compliant, an equivalent BPEL program is automatically generated from the abstract BP-calculus representation. Authors in Ref. 28 exploits LTL as the formal foundation of compliance requirements. From a set of LTL rules (capturing applicable compliance requirements) and by applying process mining techniques, abstract business process models are synthesized semi-automatically.

We differentiate our approach by the fact that the aforementioned approaches do not consider the analysis of violations’ causes to aid the user in resolving compliance deviations. Besides, these approaches assume experienced users who are familiar and versed with formal languages. In the context of business process compliance, formal languages are usually difficult to be used and understood by business/compliance experts. For this, we have introduced a series of high-level compliance patterns as an intermediate step to their corresponding formal rules, which helps to address the usability issue and serves as the backbone of the root-cause analysis approach proposed in this paper.
Deontic logic is also common in specifying compliance constraints especially in the context of business partner contracts. Dominant work examples include Refs. 3 and 30. The study in Ref. 30 provides the foundations of the FCL (Formal Contract Language) language focusing on business partner contracts. Compliance between business and compliance specifications can be verified based on the Idealness notion.\(^{30}\) In Ref. 3, FCL is used to express different types of requirements emerging from laws and regulations. In addition, business process models are visually annotated with compliance requirements via the notion of Control tags. Similarly, these approaches do not consider the analysis of root-causes of violations. Extensions to FCL are proposed in several directions. For example, in Ref. 31, an extension is proposed to FCL to incorporate real-time compliance dimension, i.e. Temporalized Violation Logic.

The modeling and verification of task allocation and authorization constraints have also gained significant interest, particularly in the information systems security field. Some studies in this area focus on the modeling and visualization of authorization constraints inside business process models, without offering any verification facility. Works in Refs. 25 and 32 fall in this category. Wolter et al.\(^{25}\) proposed to extend BPMN notations to enable the modeling and visualization of task-based authorization constraints. More specifically, they have proposed to structurally annotate authorization constraints into BPMN models. Although, Ref. 25 have considered variants of the SoD requirements, our approach is distinguished by having a concrete verification and root-cause analysis methods to reason about such requirement violations, which also considers hidden/explicit causes and possible caveats. We also consider a wider range of compliance requirements that spans over the control-flow and the data-validation compliance perspectives. We are also aligned with the argument in Ref. 3 that business and compliance specifications should be handled separately. This is mainly because business and compliance specifications have different objectives (business perspective versus ownership and governance perspectives), nature (procedural versus declarative), and lifecycles.

There are also studies on task allocations that consider the verification of authorization constraints against business process models. For example, an extension to Ref. 25 is presented in Ref. 33 to enable the automated verification of these constraints following a model-checking approach. Extended BPMN notations are first mapped to Colored Petri net (CPN) enriched with authorization constraints, which are formally represented using CPN ML language. Our approach is distinguished by separating business logic from compliance logic. It is also differentiated by providing a reasoning mechanism to detect core reasons of compliance violations that consider a wide range of compliance constraints beyond the SoD requirements.

Assisting the user to resolve compliance violations during design-time has also been studied in Refs. 12, 26, 34, 35 and 36. The notion of proximity relation has been introduced in Ref. 34 that quantifies the degree of deviation of a modified business process model from the original one. Their goal is to resolve non-compliance by
identifying the least modified process models. The authors also introduce heuristics for detecting and resolving compliance violations. This approach\textsuperscript{34} can help the user to identify the minimal set of modifications to be performed on a non-compliant business process model to transform it to a compliant one. Furthermore, a visualization of compliance violations has been introduced in Ref. 26 by utilizing Temporal Logic Querying (TLQ). In this approach, the user is given feedback by highlighting the fragment in the corresponding BPMN model that is considered as the source of violations. Similarly, the approach in Ref. 12 highlights the fragments in the BPEL models that are the source of violations. The study in Ref. 36 introduces the notion of compliance distance as a quantification of the effort required to transform a non-compliant business process model to a compliant one, which takes the value between zero and one. However, these approaches do not direct the user to modify the identified fragment by analyzing the root-causes of violations.

The study in Ref. 35 follows an automated planning-based approach to (semi-) automatically resolve design-time compliance violations, which is based on a predefined resolution context. It focuses on representing and checking the constraints regarding only the control-flow of process specifications (which correspond to Precedes and LeadsTo patterns in our pattern system). A catalogue of violations (violation patterns) has been identified for these sequential constraints and a resolution algorithm has been proposed for each identified violation pattern. Our approach considers a wide range of compliance patterns in addition to the precedence patterns (LeadsTo and Precedes), which have been analyzed for violations root-causes. Besides, the implementability status of this approach\textsuperscript{35} is unclear, as no implementation details have been discussed.

8. Conclusions and Outlook

Business processes — many of which are implemented as a SoA these days — form the foundation for all organizations, and as such, are impacted by laws, policies and industry regulations. Without an explicit framework to ensure compliance of service-enabled processes, organizations face litigation risks and even criminal penalties. One of the significant provisions toward business process compliance is a framework that would enable business/compliance experts to define compliance constraints and weave them into service-enabled processes. Compliance management should be considered from the early stages of the business process design, such that compliance constraints are designed into service-enabled processes. To enable automated reasoning techniques for verifying and ensuring compliance, these compliance constraints should be grounded on a formal language. Using compliance patterns to specify compliance constraints and automatically generate formal specifications significantly facilitate the work of the compliance expert.

Moreover, recovering from compliance violations is an important issue that has not received much attention by the research community. The business process/compliance experts should be provided with feedback that reveals the
root-causes of these violations and aids their resolution; not merely an indication that a particular constraint is violated. To address this problem, in this paper we have proposed a taxonomy of compliance constraints based on the notion of compliance patterns. Next, we have introduced a root-cause analysis approach to automatically reason about violations rooted on the proposed taxonomy. Based on this analysis, the business process/compliance expert is provided with relevant guidelines, suggestion and caveats for compliance resolution.

The root-cause analysis approach including its compliance constraint taxonomy is validated in three ways. First, the internal and construct validity are verified by formalizing the taxonomy, and particularly, the atomic and composite patterns in LTL. Second, the implementability of our approach is ascertained with a prototype. Also, we have explored and tested our approach with case studies drawn from industrial partners in the COMPAS EU project in which we participate. The validation of the proposed approach will further be intensified by its application on various empirical experiments and/or case studies on prospective users of the developed toolset.

The main focus of this work is on design-time compliance verification and analysis. However, design-time and runtime compliance management are complementary and indispensable phases for ensuring and enforcing full guaranteed compliance. For example, design-time verification of segregation-of-duties constraints will only provide partial compliance and therefore it should be extended with runtime verification to guarantee compliance throughout the entire business process lifecycle. Addressing compliance verification and analysis during runtime, based on the proposed compliance pattern taxonomy, and integrating it to the proposed design-time verification and analysis approach entails the next step toward a comprehensive compliance management framework. Currently, our approach also inherits the limitations posed by the adopted technologies and methods (such as BPEL) used for process specifications and verifications. Future work will also be directed towards working on the extensions and enhancements to leverage the capabilities of such technologies. Moreover, extending the approach to reason about data aspects of business processes and to address false negatives\textsuperscript{37} (that usually result from the under-specification of relevant requirements) also entail extension points to our root-cause analysis approach.

The patterns we study in this paper have been synthesized by investigating various compliance sources and studies. These patterns relate mainly to the control-flow, resource and data aspects of business processes. However, as mentioned in Sec. 6.4, this set was not sufficient to handle all types of compliance constraints encountered in our case study conducts. Hence, it is necessary to enrich the taxonomy with additional patterns to address a wider range of compliance constraints, such as those handling the time aspect of process specifications, and those focus on different business domains (such as health, safety, environment, etc.). Future work will therefore concentrate on extending the compliance constraints taxonomy with additional domain-specific compliance patterns.
Acknowledgments

The authors gratefully acknowledge PricewaterhouseCoopers (Netherlands), Thales Services (France) and other COMPAS Project Partners for their effort in providing and participating in the case studies and scenarios, and their valuable contributions. Thanks to Dinh Khoa Nguyen (Tilburg University) for fruitful discussions.

References


