

Root-Cause Analysis of Design-time Compliance Violations on the basis of Property Patterns

Amal Elgammal¹, Oktay Turetken, Willem-Jan van den Heuvel, Mike Papazoglou

European Research Institute in Service Science (ERISS), Tilburg University,
Tilburg, the Netherlands
{a.f.s.a.elgammal, o.turetken, w.j.a.m.vdnheuvel, m.p.papazoglou}@uvt.nl

Abstract. Today's business environment demands a high degree of compliance of business processes with business rules, policies, regulations and laws. Compliance regulations, such Sarbanes-Oxley force enterprises to continuously review their business processes and service-enabled applications and ensure that they satisfy the set of relevant compliance constraints. Compliance management should be considered from the very early stages of the business process design. In this paper, a taxonomy of compliance constraints for business processes is introduced based on property specification patterns, where patterns can be used to facilitate the formal specification of compliance constraints. This taxonomy serves as the backbone of the root-cause analysis, which is conducted to reason about and eventually resolve design-time compliance violations. Based on the root-cause analysis, appropriate guidelines and instructions can be provided as remedies to alleviate design-time compliance deviations in service-enabled business processes.

Keywords: Regulatory compliance, Compliance constraint detection and prevention, Design-time compliance management, Formal compliance model, Compliance patterns, root-cause analysis.

1 Introduction

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 [1]. 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.

¹ This work is a part of the research project "COMPAS: Compliance-driven Models, Languages and Architectures for Services", which is funded by the European commission, funding reference FP7-215175.

Business processes form the foundation for 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 litigation risks and even criminal penalties. Compliance regulations, such as HIPAA, Basel II, Sarbanes-Oxley (SOX) and others require all organizations to review their business processes and ensure that they meet the compliance standards set forth in the legislation. In all cases, these new control and disclosure requirements create auditing demands for SOAs.

SOAs should play a crucial role in compliance, allowing management to ascertain that internal control measures that govern their key business processes can be checked, tested, and potentially certified with their underlying web-services.

Compliance is about ensuring that business processes, operations and practices are in accordance with a prescribed and/or agreed on set of norms [2]. A *compliance constraint (requirement)* refers to any explicitly stated rule or regulation that prescribes any aspect of an internal or cross-organizational business process. Compliance constraints may emerge from different sources and can take various forms. They may originate from legislation and regulatory bodies (such as Sarbanes-Oxley and Basel II), standards and code of practices (such as: ISO 9001) and/or business partner contracts.

Not only the large and ever-increasing number of compliance constraints but also the diversity and complexity of these constraints, complicate the compliance management process [3]. Consequently, a comprehensive compliance management solution is of utmost importance to support compliance throughout all the stages of the complete business process lifecycle. A major requirement of a generic compliance management approach is that it should be sustainable [2]. A preventive focus is fundamentally required in order to achieve the sustainability requirement. Compliance should be considered at the very early stages of business process design, thus enforcing compliance by design.

Compliance constraints should be based on a formal foundation of a logical language to facilitate the application of future automatic reasoning techniques for verifying and ensuring business process compliance. However, formal specifications in general are difficult to write and understand by users. The notion of *property specification patterns* (Dwyer's property patterns) was introduced in [4] as high-level abstractions of frequently used logical formulas. Property patterns assist users in understanding and defining formal specifications, which significantly facilitates the work of the user, as she doesn't need to go into the lower-level and complex details of the adapted formal language.

By applying the automated verification tools that are associated with the utilized logical language (e.g. NuSMV2 model-checker [5]), compliance between specifications and the applicable set of compliance constraints can be automatically checked. However, the verification results are usually a list of which compliance rules have been violated and which have been satisfied. Clearly, existing practices and approaches are by far insufficient to effectively assist business process/service designers in resolving potential conflicts or violations between service-enabled processes and associated rules, laws and regulations. A structured approach is critical

to allow designers –many of which are non-experts in formal languages- to formally capture compliance rules and policies, and then semi-automatically detect the root-cause of compliance anomalies and provide heuristics to create corrective actions to resolve them. The main focus of this paper is on *design-time* compliance management and analysis.

In this paper, we use Dwyer’s property specification patterns [4] and Linear Temporal Logic (LTL) [6] to formally represent compliance constraints. Furthermore, we present pattern extensions and we introduce new patterns that are frequently used to specify compliance constraints. Then, a compliance constraint taxonomy is built up on top of these patterns, which represents the backbone of the root-cause analysis conducted in this paper. Finally, the root-cause analysis approach is presented to reason about design-time compliance violations. The *Current Reality Tree (CRT)* of Goldratt’s Theory of Constraints (TOC) [7], [8] is adapted as the root-cause analysis technique. By traversing the CRTs, appropriate remedies are provided as guidelines/suggestions that help the user/expert to resolve the compliance deviations.

The rest of this paper is organized as follows: a design-time compliance management approach is briefly discussed in Section 2. Section 3 presents a scenario used as the running example throughout this paper. Section 4 presents the proposed root-cause analysis approach to reason about design-time compliance violations. Related work is summarized in Section 5. Finally, conclusions and outlook are highlighted in Section 6.

2 Design-time Compliance Management

To provide a brief overview of the compliance management approach maintained in this paper, this section briefly discusses important aspects of a comprehensive compliance management framework, underlining the features that deal with managing compliance during the *design-time*. Fig. 1 depicts an overview of the key practices and components of this approach, and highlights the parts that outline the *scope* of this paper. There are two primary roles involved in this approach: (i) a *business expert*, who is responsible for defining and managing service-enabled business processes in an organization while taking compliance constraints into account, and (ii) a *compliance expert*, who is responsible for the internalization, specification and management of 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 are semantically aligned and may reside in a shared environment. Process models including service descriptions are defined and maintained in the business process repository, while the 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) use the same constructs through the usage of a shared domain-specific ontology.

The approach assumes the overall process to start either from the business process side (the right-hand side of Fig. 1) or from the compliance requirements side (left part of Fig. 1). Process models can be specified in Business Process Execution Language (BPEL²) de facto standard; However, as BPEL is not grounded on a formal model, any BPEL specification should be transformed into a formal representation (e.g. a finite state automaton, such as Buchi automata [9]) to enable the verification of these formal definitions against formally specified compliance rules.

On the other hand, the *internalization* of compliance constraints originating from regulations, policies, standards and other compliance sources into a set of organization-specific compliance requirements involves not only compliance but also business process domain knowledge. It may require compliance expert to work in collaboration with the business expert to define and iterate an effective set of requirements to address these constraints.

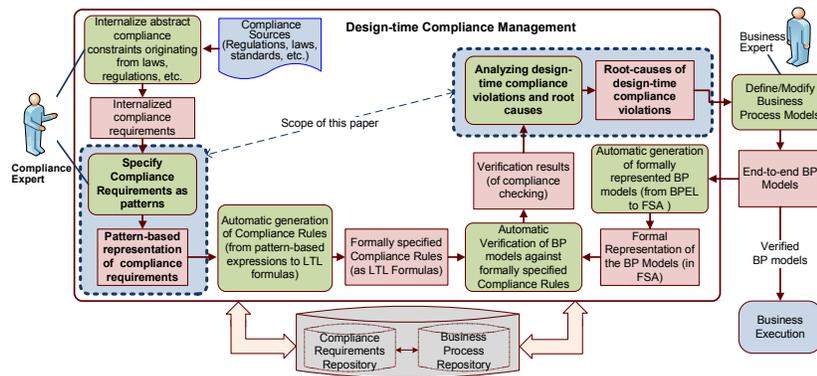


Fig. 1. Design-time compliance management approach

A compliance expert may apply patterns to render compliance constraints, which represents an intermediate step between internalized compliance requirements and formal statements (as LTL formulas for our case). These pattern-based expressions are then automatically transformed into LTL formulas, based on the mapping rules between patterns and LTL. As shown in Fig. 1, the inputs to the ‘automatic verification’ component of the approach are; the formally specified end-to-end business process models; and the LTL rules capturing compliance requirements. Then, automatic verification is supported by ‘model-checkers’ [10].

Analysis of the verification results and their root-causes should be assisted by a component of the approach, which also directs the business expert in modifying the business process model so she may resolve any compliance violation. The counter-example tracing facility, typically provided by the model-checkers, can also aid user by highlighting the fragments in the business process model that are the sources of non-compliance. The business process models are updated based on the compliance verification and analysis results and re-mapped to their formal forms and re-verified

² BPEL: Business Process Execution Language, <http://en.wikipedia.org/wiki/BPEL>

against the set of applicable compliance requirements. This process iterates until no violations are detected.

This paper focuses on the parts in Fig. 1 that are enclosed (with dotted lines), which are: the pattern-based specification of compliance requirements and analyses of design-time violations and root-causes. Our work on the other components of the approach are kept outside the scope of this paper.

3 Running Scenario

The Internet reseller scenario, which is used as the running example throughout this paper, is one of the industry scenarios explored within the EU funded COMPAS research project [12]. The scenario is set in an e-business application domain, and more particularly, online product selling systems.

The scenario starts with the customer checking product information on a website. Next, if the customer chooses a specific item, she submits an order along with her customer data. Next, the sales department validates the order by contacting the credit bureau to check the credit worthiness of the customer. Afterwards, the financial department creates the invoice and checks for payments. Finally, a delivery request is sent to the supplier.

Table 1 shows excerpts of the compliance requirements relevant to this scenario. Each compliance requirement is described in terms of: (i) an ID (ii) internalized compliance requirement (iii) its representation as patterns (as discussed in Section 2), and (iv) an explanation of its pattern representation.

Table 1. An excerpt of the relevant compliance requirements.

ID	Compliance Requirement	Pattern Representation	Description
R1	Computer-generated sales order confirmations or cancelations are sent to customers after validating the order.	<i>ValidateOrder(x,y)</i> LeadsTo (<i>SendConfirm(x)</i>) MutexChoice <i>SendCancel(x)</i>	<i>ValidateOrder</i> for sales order <i>y</i> and customer <i>x</i> is followed by either sending a confirmation or cancelation to customer <i>x</i> .
R2	Sales orders over a set threshold require approval by management before acceptance by the system.	(<i>SalesOrder(y,threshold)</i> exists) Imply (<i>Approve(y, manager)</i> Precedes <i>Accept(y)</i>)	If there is a <i>salesOrder</i> <i>y</i> that exceeds a threshold <i>threshold</i> then <i>Approve</i> action performed by <i>manager</i> should precedes <i>Accept</i> of order <i>y</i> .
R3	Appropriate segregation of duties is maintained between credit checking and cashing functions.	<i>CreditChecking(x)</i> SegregatedFrom <i>Cashing(x)</i>	<i>CreditChecking</i> function for customer <i>x</i> should be segregated from the <i>Cashing</i> function for the same customer

4 Compliance Patterns and Compliance Constraints Taxonomy

This section presents a taxonomy of pattern-based compliance constraints for business processes. As shown in Fig. 2, 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

in two main classes of patterns; namely *atomic* and *composite*. The lower part of Fig. 2 presents the atomic patterns, which are adapted from Dwyer's property specification pattern system [4].

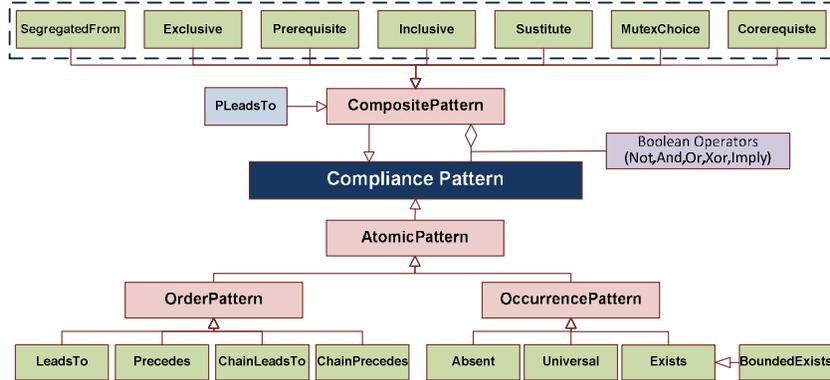


Fig. 2. Compliance constraints taxonomy based on patterns.

Atomic patterns introduce two main sub-classes: *Occurrence* and *Order* pattern classes. Their properties can be described as follows:

Occurrence patterns are:

- *Absent*: Indicates that a given state³ P does not occur within the system.
- *Universal*: Indicates that P occurs throughout the system.
- *Exists*: Indicates that P must occur within the system.
- *Bounded exists*: Indicates that P must occur at least/exactly/at most k times within the system.

Order patterns are:

- *Precedes*: A given state P must always be preceded by a given state Q .
- *LeadsTo*: P must always be followed by Q .
- *Chain precedes*: A sequence of states P_1, \dots, P_n must always be preceded by a sequence of states Q_1, \dots, Q_m .
- *ChainLeadsTo*: A sequence of states P_1, \dots, P_n must always be followed by a sequence of states Q_1, \dots, Q_m .

As shown in the upper part of Fig. 2, compliance patterns can be nested using Boolean logic operators including *Not*, *And*, *Or*, *Xor* and *Imply* to help the definition of complex requirements in terms of other compliance patterns (composite patterns). For instance, the *PLeadsTo* pattern introduced in [11] is an 'And' composition of the two atomic patterns (P Precedes Q) And (P LeadsTo Q).

In addition to the patterns described above, this paper introduces seven new compliance patterns, namely: *Exclusive*, *Substitute*, *Corequisite*, *Inclusive*,

³ *State* represents a node in finite state automata (used for formal representation of a BP model as discussed in Section 2). In our context, it indicates a certain BP activity or a condition on any related artifact. '*ValidateOrder*' activity and '*OrderAmount > 500*' branching condition are examples of states.

Prerequisite, *MutexChoice*, and *SegregatedFrom*. Although these patterns commonly occur within the domain of business process compliance, they are also applicable for the specification of properties in different domains and context.

The *SegregatedFrom* pattern captures the typical separation-of-duties security principle, which mandates that two specific activities should be performed by two different roles. Table 2 presents the mapping from the newly introduced compliance patterns to atomic patterns together with their meaning and their formal representation as LTL formulae

Table 2. Mapping of new compliance patterns.

Composite Compliance Pattern	Description	Atomic Pattern Equivalence	LTL Representation
P <i>Segregated-From</i> Q	(Activities) P and Q should be assigned to different roles	$(P \text{ Leads } Q) \wedge (P.\text{Role1}) \neq (Q.\text{Role2})$	$G(\neg Q \text{ } W \text{ } P) \wedge G(P \rightarrow F(Q)) \wedge G((P.\text{Role}(\text{Role1}) \rightarrow G(\neg(Q.\text{Role}(\text{Role1}))) \neg F(P) \vee F(Q))$
P <i>Inclusive</i> Q	The presence of P mandates that Q is also present	$(P \text{ exists}) \rightarrow (Q \text{ exists}) = \neg(P \text{ exists}) \vee (Q \text{ exists})$	$\neg G(\neg P) \vee G(\neg(Q))$
P <i>Prerequisite</i> Q	The absence of P mandates that Q is also absent	$(P \text{ isabsent}) \rightarrow (Q \text{ isabsent}) = \neg(P \text{ isabsent}) \vee (Q \text{ isabsent})$	$\neg G(\neg P) \vee G(\neg(Q))$
P <i>Exclusive</i> Q	The presence of P mandates the absence of Q. And presence of Q mandates the absence of P	$(\neg(P \text{ exists}) \vee (Q \text{ isabsent})) \wedge (\neg(Q \text{ exists}) \vee (P \text{ isabsent}))$	$(\neg(F(P)) \vee G(\neg(Q))) \wedge (\neg(F(Q)) \vee G(\neg(P)))$
Q <i>Substitute</i> P	Q substitutes the absence of P	$(P \text{ isabsent}) \rightarrow (Q \text{ exists}) = \neg(P \text{ isabsent}) \vee (Q \text{ exists})$	$\neg G(\neg(P)) \vee F(Q)$
P <i>Corequisite</i> Q	Either activities P and Q should exist together or to be absent together	$(P \text{ exists}) \text{ iff } (Q \text{ exists}) = ((P \text{ exists}) \wedge (Q \text{ exists})) \vee ((P \text{ isabsent}) \wedge (Q \text{ isabsent}))$	$(F(P) \wedge F(Q)) \vee (G(\neg P) \wedge G(\neg Q))$
P <i>MutexChoice</i> Q	Either P or Q exists but not any of them or both of them	$(P \text{ exists}) \text{ Xor } (Q \text{ exists}) = ((P \text{ exists}) \wedge (Q \text{ isabsent})) \vee ((Q \text{ exists}) \wedge (P \text{ isabsent}))$	$(F(P) \wedge G(\neg(Q))) \vee (F(Q) \wedge G(\neg(P)))$

In LTL [6], [10]; G , F and U correspond to the temporal operators ‘always’, ‘eventually’ and ‘until’ respectively. ‘ G ’ denotes that formula f must be true in all the states of the business process model. ‘ F ’ indicates that formula f will be true at some state in the future. ‘ U ’ means that if at some state in the future the second formula g will be true, then, the first formula f must be true in all the subsequent states.

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 utmost importance to provide the compliance expert intelligent 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 why the violation occurred and what strategies can be used as remedies. Based on the compliance constraint taxonomy proposed in Section 2, we have further analyzed and formalized root-causes for each pattern in the taxonomy. Particularly, we investigated and reported all possible causes of a violation of a compliance constraint represented

by a specific pattern. However, based on the root-cause analysis, only the exact deduced cause(s) of the violation(s) is communicated to the user (as explained in Section 5.5).

For this purpose, we have adapted the *Current Reality Tree (CRT)* technique from Goldratt's Theory of Constraints (TOC) [7]. A *current reality tree* is a statement of a core problem and the symptoms that arise from it. It maps a sequence of causes and effects from the core problem to the symptoms arising from one core problem or a core conflict. If the core problem is removed, each of the symptoms may be removed. Operationally the process works backwards from the apparent undesirable effects or symptoms to uncover or discover the underlying core causes [7]. The CRT has been chosen due to its simplicity and the visual representation of the causes and effects.

A CRT usually starts with a list of problems called *Undesirable Effects (UDEs)*, which represent negative or bad conditions. They are also 'effects' because for most part they are caused by something else [8]. The key question begins with 'why a violation occurs?' (the root of the tree). The answer to this question will generate child-(eren) of the UDE under consideration. For each child, which might be a UDE, 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 (in the leaf level of the tree). Incoming connections to an UDE from its children are connected via logical 'or' operator; unless otherwise specified. Due to space limitation, we do not present all the current reality trees corresponding to each pattern given in the taxonomy (in Fig. 2).

5.1 Current Reality Trees for Atomic Patterns

One of the main advantages of using the Current Reality Tree technique (CRT) is that it is self-explanatory. Fig. 3 presents the CRTs for *Exists*, *Precedes*, *LeadsTo*, *PleadsTo*, *Absence* and *Universal* patterns. The root of each CRT represents an undesirable effect (UDEs). For our purpose, an UDE is a violation of a specific pattern. Hence, the root of each tree represents a violation to a specific pattern. For example, as shown in Fig. 3, the violation to '(P *Precedes* Q) pattern' is considered as the UDE of the *Precedes* CRT.

Deeper levels in the tree are guided by answering the same 'why' question. For example, the question that should be addressed here is: why (P *Precedes* Q) is violated. The answer to this question is: because (Q *Exists* is satisfied) and (P *exists* is violated) before it. This is depicted as the second level of the tree. The same 'why' question is applied to the UDE under consideration and analysis continues until the root-causes of the problem, i.e. the leaves of the tree are reached. For each leaf, the user is provided with guidelines as remedies to compliance violations. These guidelines are depicted in the CRTs as squared brackets linked to the leaves, e.g. 'Swap the occurrence of P and Q', where P and Q are business process activities that will be parameterized with the actual activity names. In case the leaf is a composite pattern, it will be replaced by its corresponding CRT. This process iterates continuously until all the leaves of the tree are atomic patterns.

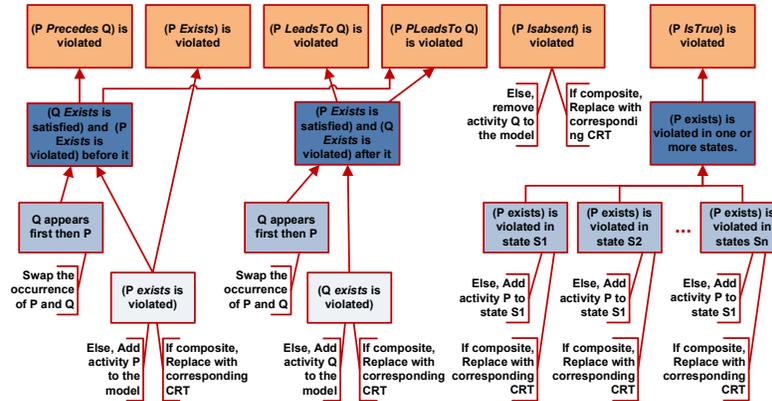


Fig. 3. CRT for Exists, Precedes, LeadsTo and PLeadsTo patterns

5.2 Current Reality Trees for Composite Patterns

Fig. 4 presents the CRTs for the composite patterns that comprise one or more compliance patterns connected with a Boolean operator. An example output from the analysis process could be the UDE ‘(PropertyPattern1 and PropertyPattern2) is violated’. Let this UDE be *UDE1*. According to the truth table of the ‘and’ operator, the ‘and’ statement is only true if its two operands are evaluated to true, otherwise the statement is evaluated to false. By applying the same ‘Why’ question to *UDE1*, the answer is either:

- i. *UDE1.2*: PropertyPattern1 is violated, or
- ii. *UDE1.2*: PropertyPattern2 is violated, or
- iii. *UDE1.3*: PropertyPattern1 is violated and PropertyPattern2 is violated.

UDE1.1, *UDE1.2* and *UDE1.3* correspond to the violation of other compliance patterns. Hence, each UDE corresponds to a compliance pattern will be replaced with its corresponding CRT.

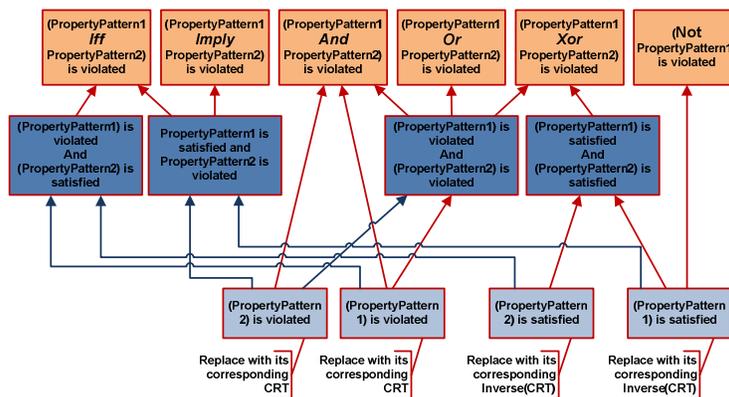


Fig. 4. CRT for composite patterns.

Notably, for the negation operator, ‘(Not PropertyPattern1) is violated’, the undesirable effect in this case is ‘(PropertyPattern1) is satisfied’, which semantically represents the opposite of the CRTs analyzed above. For this purpose, each compliance pattern is re-analyzed the same way, with the undesirable effect (UDE) being ‘property pattern is satisfied’ (e.g. the lower levels of MutexChoice CRT in Fig. 5).

5.3 Current Reality Trees for the New Compliance Patterns

The CRTs of the newly introduced compliance patterns (e.g. SegregatedFrom, Inclusive, etc.) are instances from the CRTs of composite patterns given in Fig. 4. Two examples of the CRTs of these compliance patterns are presented in Fig. 5; namely: *Exclusive* and *Mutexchoice*.

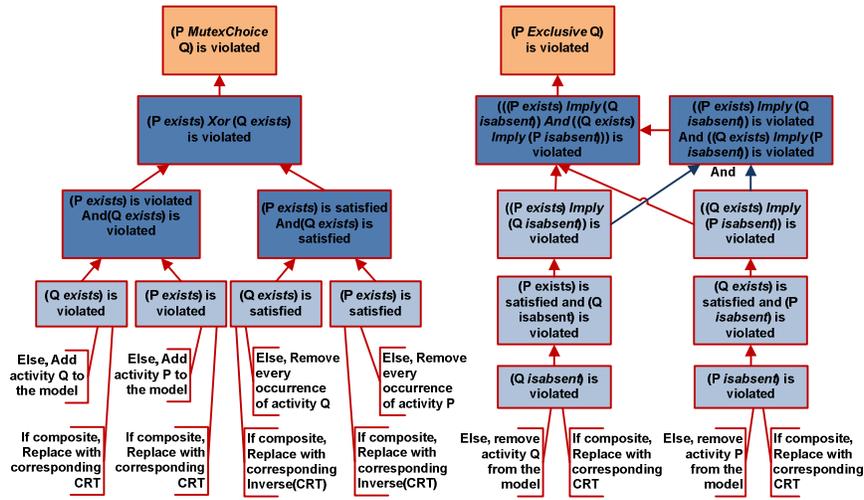


Fig. 5. CRTs for Exclusive and MutexChoice Composite Patterns

As shown in Fig. 5, the *MutexChoice* composite pattern is an ‘Xor’ composition between two atomic patterns: (P Exists) and (Q Exists). Hence, for the *MutexChoice* composite pattern, the CRT of the ‘Xor’ composite pattern is instantiated. The instantiation process starts from the outermost pattern to the innermost pattern. Similarly, the CRT of the *Exclusive* pattern is built based on the CRTs of ‘And’, ‘Implies’ composite patterns and *isabsent* atomic pattern.

5.4 Current Reality Trees of the Internet Reseller Scenario

This section presents briefly due to space limitations the application of the pattern based representation approach and relevant CRTs of the second and third compliance constraints (R2 & R3) given in Table 1 from the Internet reseller scenario.

In case violations are detected to R2 and R3 (e.g. the model-checker detects the violations), the CRTs to reason about violations are automatically constructed and

traversed. Fig. 6 presents the CRTs of the violations to R2 and R3. The CRT of the violation to R2 is an 'Imply' composition between two atomic patterns; *exists* and *precedes*. The CRT of the violation to the segregation-of-duty compliance constraints (R3) is shown in the right-hand side of Fig. 6.

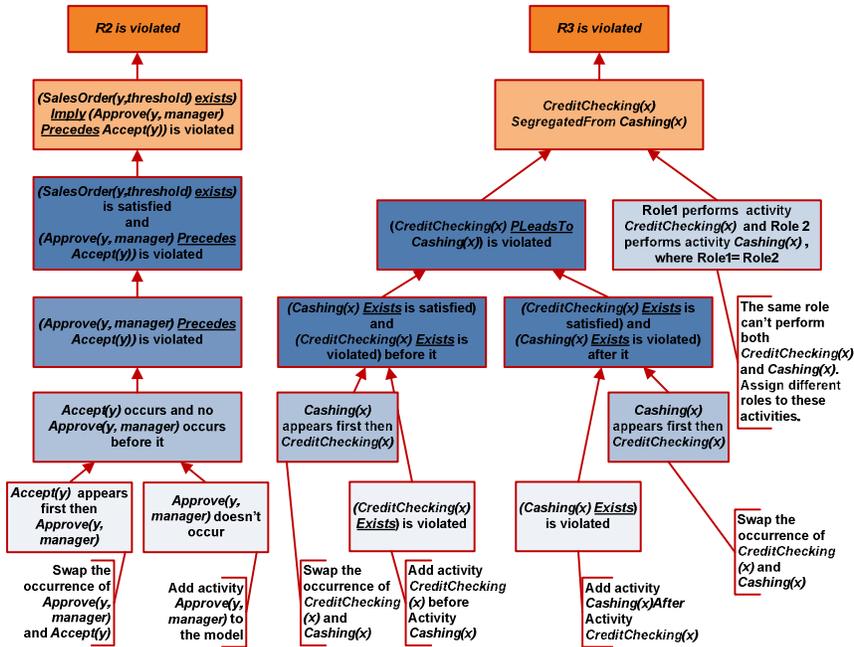


Fig. 6. CRTs for the violation to R2 and R3

5.5 Implementation of the Root Cause Analysis Approach

An effective and scalable implementation of the concepts discussed above is a challenging yet necessary step to help to ascertain the soundness of the approach proposed in this paper. We are currently implementing an environment as a part of a comprehensive tool-suite for business process compliance management, based on the concepts described in above sections. The prototype is a web-based environment⁴, which also incorporates standalone tools for building graphical representation of requirements using patterns. The web-based environment is implemented using 'PHP'⁵ as the main scripting language and Oracle database (ver.8i)⁶ as the repository for compliance data and meta-data. The integration with Reo toolkit [13], which is used for process verification, is ongoing. The integration is achieved through a group

⁴ <http://eriss.uvt.nl/compas>

⁵ <http://en.wikipedia.org/wiki/PHP>

⁶ http://en.wikipedia.org/wiki/Oracle_Database

of asynchronous web services, which mainly forwards BPEL representation and relevant formal compliance rules specified in LTL as input to Reo toolkit and retrieve back the verification result listing the rules that have been checked and whether they are satisfied or not.

Fig. 7 presents one of the user interfaces from the implementation reflecting how the results of the root cause analysis are communicated to 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 Internet reseller scenario, where R1 is satisfied, while, R2 and R3 are violated.

The screenshot shows a 'Compliance Profile' window for the business process 'ORDER PROCESSING' as of March 17, 2010 11:23. It contains a table with the following data:

Compliance Constraint	Constraint Source	Pattern	Satisfied?	Result Description / Remedy
1 Computer-generated sales order confirmations or cancellations are sent to customers after validating the order	- SOX Section 404	(ValidateOrder(x)) LeadsTo (SendConfirm(x) MutesChoice SendCancel(x))	Yes	Control satisfied
2 Sales orders over a set threshold require approval by management before acceptance by the system	- SOX Section 404 - Internal Policy P-S01	(SalesOrder(y)/reshold) exists imply (Approve(y, manager) Precedes Accept(y))	No	Add 'Approve' activity to the definition
3 Appropriate segregation of duties are maintained between 'Credit Checking' and 'Cashing' functions	- SOX Section 404 - ISO 17799 - 10.1.3	CreditChecking(x) SegregatedFrom Cashing(x)	No	Swap the occurrence of 'Credit Checking' and 'Cashing' activities

ID	Rule Description	Rule Statement (LTL)	Design Time	Run Time	Satisfied?	Result Desc./ Remedy
3.1	'Credit checking' activity should exist	G(-InitiaCWC) W (PostCWC)	✓	✓	Yes	'Credit checking' activity exists
3.2	'Cashing' activity should exist	G(PostCWC) → G (PostCWC.Role(Supervisor))	✓	✓	Yes	'Cashing' activity exists
3.3	'Credit Checking' should be preceded by 'Cashing'	G(-Credit Checking) W (Cashing)	✓	✓	No	Swap the occurrence of 'CreditChecking' and 'Cashing' activities
3.4	'Credit Checking' and 'Cashing' should be performed by two different roles	G(Credit Checking.Role) → G (-Cashing.Role)	✓	✓	Yes	'Credit Checking' and 'Cashing' activities performed by different roles

Fig. 7. A user interface implementation for the running scenario.

6 Related Work

Deontic logic and temporal logic families have been successfully utilized in the literature as the formal foundation of compliance constraints. Key work examples utilizing languages based on Deontic logic are: [2], [14], [15], [16], [17], [18], [19] and [20]. On the other hand, major works built on top of temporal logic are: [5], [11], [21], [22], [23], [24], and [25]. Due to space limitation, we are listing here key works grounded on temporal logic.

Authors in [5] proposed a static-compliance checking framework that includes various model transformations. Compliance constraints are modeled using the graphical Business Property Specification Language (BPSL) tool. Next, NuSMV2 model checker is used to check the compliance. The study in [21] utilized π -Logic to formally represent compliance constraints. On the other hand, business process models are abstractly represented using BP-Calculus. Using HAL toolkit, a BPEL program equivalent to the abstract representation can be automatically generated if the two specifications are compliant. The study in [23] utilized past LTL (PLTL), where

properties about the past can be represented. However sequential compliance constraints are just considered. On the other hand, the study in [24] utilizes the original pattern-based system, however, it considers aspects relevant to monitoring compliance during runtime. Furthermore, authors in [25] have extended Dwyer's property pattern to capture time-related property specifications. E.g. activity A must always be followed by activity B within k time units. Integrating real-time dimension to the proposed approach entails an ongoing research direction. The study in [11] has utilized Dwyer's patterns for the verification of service compositions. In [22], real-time temporal object logic was proposed for the formal specification of compliance requirements based on a pre-defined domain ontology. Real-time temporal object logic is an expressive logic, however it is excessively difficult to be used.

Assisting the user to resolve non-compliance during design-time has been addressed in [26], [27] and [23]. The notion of *proximity relation* has been introduced in [26] that quantitatively compare how much a modified business process model deviated from the original one. The goal is to resolve non-compliance violations by identifying minimally different process models. They also introduced heuristic guidance for detecting and resolving compliance violations. A major distinction to our work is that we provide concrete guidelines and our work is based on a compliance constraint taxonomy based on extended patterns. The notion of *compliance distance* has been introduced in [20, 27], as a quantification of the effort required to transform a non-compliant business process model to a compliant one, which can take the values between 0 and 1. A visualization of compliance violations has been introduced in [23] by utilizing Temporal Logic Querying (TLQ). To the best of our knowledge, this is the first study that considers an exhaustive analysis of root-causes of compliance violations, and providing the user with only relevant guidelines/suggestions as remedies to resolve the compliance deviations based on high-level patterns.

7 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 auditing SOA framework to ensure compliance of service-enabled processes, organizations face litigation risks and even criminal penalties. One of the significant provisions towards business process compliance is a framework that would enable service engineers to define compliance constraints and weave them into service-enabled processes. Compliance management should be considered from the very early stages of the business process design, such that compliance constraints are designed into service-enabled processes. To enable automatic reasoning techniques for verifying and ensuring compliance, these compliance constraints should be grounded on a formal language. Using property specification patterns to specify compliance constraints and automatically generate formal specifications significantly facilitate the work of the compliance expert.

Moreover, recovering from compliance violations in service-enabled processes is an important issue that has not paid much attention by the research community. The compliance expert should be provided with intelligent feedback that reveals the root-causes of these violations and aids their resolution; not merely an indication whether the constraint is violated. To address this problem, we have proposed a taxonomy of compliance constraints based on Dwyer's property patterns and extended this taxonomy with patterns that are frequently used to specify compliance constraints. Next, we have introduced a root-cause analysis approach to automatically reason about design-time compliance violations rooted on the proposed taxonomy. Based on the root-cause analysis, the compliance expert is provided with only relevant guidelines/suggestions.

The root-cause analysis approach including its compliance constraint taxonomy is validated in three ways. Firstly, the internal and construct validity are verified by formalizing the taxonomy, and particularly, the atomic and composite patterns in LTL. Secondly, the implementability of our approach is ascertained with an experimental prototype. Lastly, we have explored and tested our approach with several case studies drawn from industrial partners in the COMPAS EU project in which we participate. Furthermore, 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 prototype toolset.

Design-time and runtime compliance management are complementary and indispensable phases for ensuring and enforcing the compliance. The main focus of this work is on design-time verification and analysis. 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 another important ongoing research direction. This course of research will pave the way for a comprehensive compliance management solution that verifies, analyses and ensures the compliance of business processes on both design-time and runtime dimensions. Future work will concentrate on extending the compliance constraints taxonomy with additional domain-specific compliance patterns. This requires intensive involvement in the specification of various industrial large-scale use case scenarios.

References

1. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. *Computer* 40, 38-45 (2007)
2. Sadiq, S., Governatori, G., Naimiri, K.: Modeling Control Objectives for Business Process Compliance. 10th International Conference on BPM, pp. 149-164, Australia (2007),
3. ITIL: Information Technology Infrastructure Library. (2010)
4. Dwyer, M., Avrunin, G., Corbett, J.: Property Specification Patterns for Finite-State Verification. Workshop on Formal Methods on Software Practice, pp. 7-15, USA (1998),
5. Liu, Y., Muller, S., Xu, K.: A Static Compliance-Checking Framework for Business Process Models. *IBM Systems Journal* 46, (2007)

6. Pnueli, A.: The Temporal Logic of Programs. In: 18th IEEE Symposium on Foundations of Computer Science, pp. 46–57. (1977)
7. Dettmer, H.: Goldratt's Theory of Constraints: a systems approach to continuous improvement. ASQC Quality Press 62-119 (1997)
8. Mosely, H.: Current Reality Trees: An Action Learning Tool for Root Cause Analysis. (2006), www.jhuccp.org/training/scope/starguide/toc/rootcauseanalysis.ppt
9. Buchi, K.: On a Decision Method in Restricted Second Order Arithmetic. International Congress on Logic, Method, Philosophy of Science, pp. 1-11, Stanford (1960),
10. Clarke, E., Grumberg, J., Peled, D.: Model Checking. MIT Press, Cambridge (2000)
11. Yu, J., Manh, T., Han, J., Jin, Y.: Pattern-Based Property Specification and Verification for Service Composition. WISE06, pp. 156-168, China (2006),
12. COMPAS official web site – Project description, <http://www.compas-ict.eu/project.php>
13. Arbab, F., Kokash, N., Meng, S.: Towards Using Reo for Compliance-Aware Business Process Modeling. ISOLA08, pp. 108-123, Greece (2008),
14. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance Checking Between Business Processes and Business Contracts. EDOC 2006, pp. 221-232, Hong Kong (2006),
15. Governatori, G., Milosevic, Z.: Dealing with Contract Violations: Formalism and Domain-Specific Language. EDOC 2005, pp. 46-57. (2005)
16. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes with Obligations and Permissions. the International BPM Workshops, pp. 5-14, Austria (2006),
17. Governatori, G., Rotolo, A.: Logic of Violations: A Gentzen System for Reasoning with Contrary-to-duty Obligations. Australasian Journal of Logic (2006)
18. Governatori, G.: Representing Business Contracts in RuleML. International Journal of Cooperative Information Systems (2005)
19. Milosevic, Z., Sadiq, S., Orłowska, M.: Translating business contract into compliant business processes. EDOC 2006, pp. 211-220. (2006)
20. Lu, R., Sadiq, S., Governatori, G.: Compliance Aware Business Process Design. 5th International Conference on BPM, pp. 120-131, Brisbane (2007),
21. Abouzaid, F., Mullins, J.: A Calculus for Generation, Verification, and Refinement of BPEL Specifications. WWV'07, pp. 43-68. (2007)
22. Giblin, C., Liu, A., Muller, S., B., P., Zhou, X.: Regulations Expressed As Logical Models. 18th Conference of legal knowledge and information systems, pp. 37-48, Belgium (2005),
23. Awad, A., Weidlich, M., Weske, M.: Specification, Verification and Explanation of Violation for Data Aware Compliance Rules. ICSOC'09, pp. 500-515. Springer, (2009)
24. Namiri, K., Stojanovic, N.: Pattern-based Design and Validation of Business Process Compliance. Lecture Notes in Computer Science 59-76 (2007)
25. Gruhn, V., Laue, R.: Specification Patterns for Time-Related Properties. In: 12th Int'l Symposium on Temporal Representation and Reasoning, pp. 198-191. (2005)
26. Ghose, A., Koliadis, G.: Auditing Business Process Compliance. In: Service-Oriented Computing – ICSOC'07, pp. 169-180. (2007)
27. Lu, R., Sadiq, S., Governatori, G.: Measurement of Compliance Distance in Business Processes. Information Systems Management 25, 344-355 (2008)