Towards Resolving Compliance Violations in Business Process Models

Ahmed Awad, Sergey Smirnov, and Mathias Weske

Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
D-14482 Potsdam, Germany
{ahmed.awad,sergey.smirnov,mathias.weske}@hpi.uni-potsdam.de

Abstract. Keeping business processes compliant with regulations is of major importance for companies. Considering the huge number of models each company possesses, an automation of compliance maintenance becomes essential. Therefore, many approaches focused on automation of various aspects of a compliance problem, e.g. compliance verification. Such techniques allow localizing the problem within the process model. However, they are not able to resolve a detected violation.

In this paper we make the first attempt to systematically approach the problem of automatic violation resolution, addressing violations of execution order compliance rules. We categorize the violations into types and describe possible violation resolution strategies. The problem of choosing the concrete resolution strategy is addressed by the concept of context.

Key words: business process modeling, compliance checking, process model parsing, process model restructuring.

1 Introduction

In todays business, being compliant to regulations is inevitable. Companies are hiring experts to audit their business processes and to evidence process compliance to external/internal controls. Keeping business processes compliant with constantly changing regulations is expensive [11]. Process models provide enterprises an explicit view on their business processes. Thus, it is rational to employ them for the business process compliance checking.

Several approaches have been proposed to handle the divergent aspects of compliance on the level of process models. Most of them are focused on a model compliance checking, i.e. on a model verification problem [1, 10, 16]. The problem of violation visualization was addressed in [3], where the authors proposed a mechanism for identifying and presenting to the user process traces violating a compliance rule. However, the question of compliance violation resolution was discussed in literature (for instance, see [6]). Usually this problem is considered as a task for a human expert. However, it would be possible to (semi) automate the task of resolving compliance violations. This would be valued as an aid to the human expert to speed up the process of ensuring compliance. This paper

tackles the violation resolution problem and at the same time complements our previous work on the compliance problem presented in [1, 3].

In this paper we make the first step towards resolving execution order compliance rule violations. Since the problem is new, we start with an analysis and an identification of the main challenges. The main contribution of the paper is the set of patterns, capturing possible violation situations and their resolution strategies. We also introduce the notion of resolution context, which defines the choice of preferable resolution strategy.

The rest of the paper is organized as follows. Section 2 discusses the state of the art in the area of compliance management in business process modeling. Section 3 frames the problem of compliance rule violation resolution, identifying the main challenges. A formalism employed in resolution strategies is presented in Section 4. Sections 5 and 6 represent the paper contribution where we formally define a catalog of violation patterns and reflect on the influence of a resolution context on the operation choice. Section 7 concludes the paper and discusses the future work.

2   Related Work

An essential problem of compliance is identification of violations. Today a large body of research on compliance checking of business process models is published. Our primary interest is a work on an execution order compliance checking. The research in this area can be divided into two directions: a compliance by design and a compliance checking of existing models. The idea of compliance by design is to enforce a process model compliance already at the stage of design. [7, 8, 14, 18, 19] show how this approach can be realized. The other branch of research employs model checking techniques to verify that existing process models satisfy the compliance rules. [1, 5, 16, 25] consider this problem. Following this approach the authors of [9, 10] use business contracts as a source of compliance rules. To formalize compliance rules Formal Contract Language (FCL) is used. [21] separates process modeling from control objectives; it also uses FCL to express control requirements over annotated process models.

A resolution of violations can be driven by the severity of violations. In this sense an interesting method was introduced in [15]. This formal approach enables measuring the *degree* of compliance. Once a compliance rule is specified, the approach tells the degree of a compliance for a given process model on the scale from 0 to 1. The approach is implemented in an automated tool, which supports management decisions if the compliance violations should be resolved.

Recently, several requirements frameworks for business process compliance management have been proposed. In [17] the authors formulate requirements for tools. The requirements specify compliance management of semantic constraints (compliance rules), addressing the issues of a lifetime compliance. The focus is also on requirements to languages expressing compliance rules, on rule priorities, and on validation of process models against rules during design time and runtime. Another framework was proposed in [4]. Among other requirements, the authors discuss compliance enforceability. In this context they perceive the violation
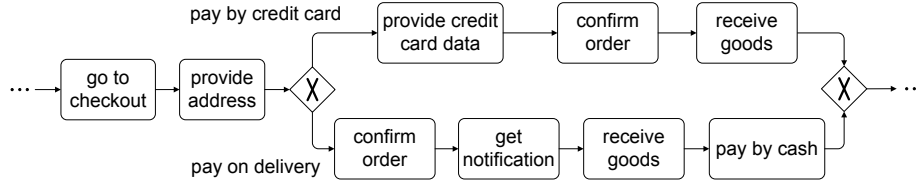
Fig. 1. Fragment of a process model "Buy goods in electronic shop"

resolution as a human task, i.e. only human experts are responsible for taking remedy actions when a violation is discovered.

It should be noticed that [6] discussed possible strategies for resolving compliance violations. However, the discussion was very high level.

The related work outlook demonstrates that the problem of business process compliance is actual, since many of its aspects have been investigated. On the other hand, it reveals that in the area of compliance violation resolution not much has been done. Many mentioned research projects may profit if they are complemented by the technique of violation resolution. Therefore, we perceive these projects as a motivation for the problem of compliance violation resolution.

The problem of violation resolution inevitably involves structural modifications of process models. In this sense, a paper on workflow inheritance and change management [22] provides valuable information. In [20] the authors described adaptations correctness criteria, guiding the change operations. Recently, a set of change patterns has been discussed in [24] along with the discussion of changes correctness.

3   Problem Definition

The goal of this section is to shape the problem of compliance violation resolution. We start with a discussion of execution order compliance rules and illustrate by examples this type of rules and their violations. Afterwards, we outline the challenges to be responded on the way to the resolution of the violations.

Execution order compliance rules restrict an order in which activities appear in a process model. In previous work [1], we distinguished two types of execution order rules: `leads to` and `precedes`. Compliance rule A `leads to` B requires that once there is an occurrence of activity A in a process model, each path leading from A to a process end contains an occurrence of activity B. A `precedes` B rule requires that once there is an occurrence of activity B in a process model, each path leading from a process start to B contains an occurrence of activity A. These are considered two basic rules from which more rules can be derived.

The named rules can be illustrated by the process example in Fig. 1. The figure presents a fragment of a buying process in an electronic shop from the customer perspective. It captures the final stage of a process, when the customer has already selected the goods and decided to checkout. Once a delivery address is provided, the customer chooses a payment method: either credit card, or cash (on delivery). Depending on the preferred method, the process evolves assuring that the customer gets the goods delivered. In case of on delivery payment, the

customer receives a notification, once the order is confirmed. The rule *Confirm order **leads to** Receive goods* holds, since every time a customer confirms an order, the goods are received. The rule *Confirm order **precedes** Get notification* also holds. However, one can impose compliance rules that are violated in this example. For instance, *Go to checkout **leads to** Pay by cash* rule is violated: there is an alternative path which allows to skip paying by cash. One can also notice that *Get notification **precedes** Receive goods* rule is violated, since on the upper branch activity *Get notification* is missing.

The example shows that there are numerous situations when execution order compliance rules are violated. To address these multiple cases we introduce violation patterns. Each pattern captures a certain violation type and describes structural modifications needed to assure model compliance. The modifications operate with fragments of a process model, i.e. a connected subgraph of a graph representing the process model. In the trivial case a fragment is one activity. The structural modifications involve two elementary operations:

Add introduces a new fragment to a business process model.
Remove deletes a fragment, which presents in an initial business process model.

Along with the two basic operations we introduce **Move** operation, which is the composition of Add and Remove. For the sake of convenience we use Move as a shortcut in the paper.

Let us refer to the example in Fig. 1. As we have mentioned *Go to checkout **leads to** Pay by cash* rule is violated in this fragment. To make the model compliant, we have to guarantee that once *Go to checkout* is executed, *Pay by cash* is executed as well. This can be achieved in several ways:

- one more occurrence of activity *Pay by cash* is added between activity *Go to checkout* and the XOR split;
- *Pay by cash* is moved to that position;
- *Go to checkout* is removed from the model.

Each of the named operations assures the compliance, but may introduce an inconsistency into the model. For instance, addition of activity *Pay by cash* before the choice means that the user always pays twice. Thus, the initial logic of the business process is broken. This example illustrates that not every modification making a model compliant is acceptable. The operations are sensitive to *resolution context*. The resolution context is defined by constraints which are imposed on every activity and which are considered to be significant during resolution process. The resolution context is independent from a concrete business process, but describes the business environment. Within the context one or more *aspects* (types of constraints) can be distinguished. Examples of aspects are control flow, data flow, and semantic annotations of activities and their interdependencies. The context may combine these three aspects in different ways. Meanwhile, we do not aim at identification of all possible aspects. Rather, in Section 6 we will discuss the influence of the context, looking at the examples. We also assume initial process models containing violations to be correct with respect to every aspect defined by the context.

Resolution of compliance violations requires structural modifications of a process model. On the contrary, formal definitions of **leads to** and **precedes**

compliance rules operate with temporal relations between the activities. This means that one compliance rule regulates relations between multiple activity occurrences (i.e. between multiple model nodes). For instance, compliance rule *Go to checkout* `leads to` *Confirm order* holds for the process model in Fig. 1, while the rule considers one occurrence of activity *Go to checkout* and two occurrences of activity *Confirm order*. This mismatch can be resolved by a method reducing an analysis of a compliance rule to an analysis of node pairs. Such a method is out of scope of this paper and we assume that we can identify a concrete pair of nodes which causes a violation. Although we assume strict matching between activity labels in rules and models, approaches like the one in [2] can be used to relax this assumption.

Another challenge is resolution of multiple violations within one model (e.g. when a model is checked against several compliance rules). In this case we propose to perform resolution of each violation in isolation from others. Multiple violations can be resolved sequentially, one after another. A problem rises if resolutions of two violations contradict one another, i.e. a correction of one violation causes another violation and vice versa. This case requires interference of a human modeler who resolves the conflict.
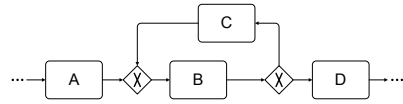
Fig. 2. Allowed loop fragment

Graph-based modeling notations allow creation of models with an arbitrary structure. We assume that process models are structured workflows. [13] defines a structured workflow as one in which each split control element (e.g. `and`, `or`) is matched with a join control element of the same type, and such split-join pairs are also properly nested. At the same time we allow structured loops in the form presented in Fig. 2.

4 Preliminaries

Correction of compliance violations assumes analysis and modification of business process models on the structural level. Therefore, we need a technique efficiently supporting these tasks. We rely on the concept of a process structure tree (PST), which is the process analogue of abstract syntax trees for programs. PSTs can be constructed using various algorithms. We propose to use an algorithm developed in [23]. The assumption that process models are structured workflows is the direct consequence of this algorithm: the technique can recognize the organization of a structured workflow. We foresee that an application of a more efficient technique softens this assumption.

The concept of a process structure tree is based on the unique decomposition of a process model into fragments. One approach is decomposition of a model into *canonical single entry single exit (SESE) fragments*, described in [12, 23]. Informally, a SESE fragment is a fragment with exactly one incoming and exactly one outgoing edge. The node sets of two canonical SESE fragments are either disjoint or one contains the other. Following [23] we consider a maximal sequence of nodes to be a canonical SESE region. If the node set of SESE fragment f_1 is

the subset of the node set of SESE fragment f_2, then f_1 is the *child* of f_2 and f_2 is the *parent* of f_1. If f_1 is the child of f_2 and there is no f_3, such that f_3 is the child of f_2 and f_3 is the parent of f_1, f_1 is the *direct child* of f_2. One example of a SESE fragment is a structured loop with one incoming edge and one outgoing edge. Further we will consider such loops, and to avoid any ambiguity we provide the definition of the loop employed in this paper.

Definition 1. A *loop* is a SESE fragment containing at least two gateways: a join and a split. The join is incident to the entry edge of the fragment, the split—to the exit edge of the fragment; the loop has two branches: one leading from the join to the split is always executed (we call it *mandatory*) and the other leading from the split to the join may be skipped (*optional*). It is allowed that one branch does not contain nodes.

Canonical SESE fragments can be organized into a hierarchy according to the parent-child relation. The hierarchy is represented with a directed tree called *process structure tree*. The tree nodes represent canonical SESE fragments.

Definition 2. A *process structure tree* $P = (N, E, r, t)$ is a tree, where:
- N is a finite set of nodes, where nodes correspond to canonical SESE fragments
- $r \in N$ is the root of the tree
- $E \subseteq (N \times (N \setminus \{r\}))$ is the set of edges. Let tree nodes $n_1, n_2 \in N$ correspond to SESE fragments f_1 and f_2, respectively. An edge leads from n_1 to n_2 if SESE fragment f_1 is the direct parent of f_2
- Every node has exactly one parent
- $t : N \to \{\texttt{act}, \texttt{seq}, \texttt{and}, \texttt{xor}, \texttt{or}, \texttt{loop}\}$ is a function assigning a type to each node in N: $\texttt{act}$ corresponds to activities, $\texttt{seq}$—sequences, $\texttt{and}$, $\texttt{xor}$, $\texttt{or}$—blocks of corresponding type, $\texttt{loop}$—fragments described in Definition 1
- $N_{<type>} \subseteq N$ denotes the set of nodes with specific type, e.g. N_{seq} are the nodes of type $\texttt{seq}$.

The definition distinguishes 6 node types: activities, sequences of activities, parallel blocks, exclusive choice blocks, inclusive choice blocks, and loops. In the illustrations depicting PST sequences are visualized with horizontal lines, choices—with diamond-shaped figures, loops—with unfilled circles. If the node type is unimportant, it is captured as a black-filled circle. A process model may contain several occurrences of one activity (e.g. activity A). Then, the model's PST has the set of nodes which corresponds to occurrences of A. To address such a set of nodes we denote it with $N_a \subseteq N$. The node representing the fragment on the mandatory branch is connected with its parent node (of type loop) with solid line, while for the optional branch a dashed line is used.

We introduce a family of auxiliary functions $exec_{<type>}$. Every function checks if a given activity is *always* executed within the given fragment. To answer this question a recursive analysis of all fragment's children is done. The following definition formalizes this function family in terms of PST.

Definition 3. The family of functions $exec_{<type>} : N_{<type>} \times N_{act} \to \{true, false\}$ is defined as:

$$exec_{xor}(p, a) = \begin{cases} true, & \text{if } \bigwedge_{\forall x:(p,x)\in E} exec_{t(x)}(x, a) = true, \\ false, & \text{otherwise.} \end{cases}$$

where $p \in N_{xor}$ and $a \in N_{act}$.

$$exec_{or}(p, a) = \begin{cases} true, & \text{if } \bigwedge_{\forall x:(p,x)\in E} exec_{t(x)}(x, a) = true, \\ false, & \text{otherwise.} \end{cases}$$

where $p \in N_{or}$ and $a \in N_{act}$.

$$exec_{and}(p, a) = \begin{cases} true, & \text{if } \bigvee_{\forall x:(p,x)\in E} exec_{t(x)}(x, a) = true, \\ false, & \text{otherwise.} \end{cases}$$

where $p \in N_{and}$ and $a \in N_{act}$.

$$exec_{seq}(p, a) = \begin{cases} true, & \text{if } \bigvee_{\forall x:(p,x)\in E} exec_{t(x)}(x, a) = true, \\ false, & \text{otherwise.} \end{cases}$$

where $p \in N_{seq}$ and $a \in N_{act}$.

$$exec_{loop}(p, a) = \begin{cases} true, & \text{if for the direct child node } x \text{ of } p \text{ laying on} \\ & \text{its mandatory branch holds } exec_{t(x)}(x, a) = true, \\ false, & \text{otherwise.} \end{cases}$$

where $p \in N_{loop}$ and $a \in N_{act}$.

$$exec_{act}(p, a) = \begin{cases} true, & \text{if } p \in N_a, \\ false, & \text{otherwise.} \end{cases}$$

where $a, p \in N_{act}$.

Since a tree has no cycles, a path between two nodes is a unique node sequence.

Definition 4. A *path* between two nodes $n_0, n_k \in N$, is a sequence of nodes $path(n_0, n_k) = (n_0, n_1, \ldots, n_k)$ where $(n_i, n_{i+1}) \in E, 0 \leq i < k$. If there is no path between n_0 and n_k we denote it with $path(n_0, n_k) = \perp$. We write $n \in path(n_0, n_k)$ to express the fact that n lies on the path from n_0 to n_k.

Definition 5 formalizes the notion of the least common parent for two nodes.

Definition 5. The *least common parent* of two nodes $n, m \in N$ in tree $P = (N, E, r, t)$ is a node $lcp(n, m) = \{p : p \in N \wedge p \in path(r, n) \wedge p \in path(r, m) \wedge \nexists p'(p' \neq p \wedge p' \in path(r, n) \wedge p' \in path(r, m) \wedge p \in path(r, p'))\}$.

If a node is of type `seq` the execution order for its direct children is defined.

Definition 6. The *order* of execution of node $n \in N$ with respect to node $p \in N_{seq}$ is a function:

$$order : N_{seq} \times N \to \mathbb{N},$$

defined if $(p, n) \in E$ and where the first argument is the parent node and second—its child.

The notion of order can be extended to all the children of a node with type `seq`.

Definition 7. The *order** of execution of a node $n \in N$ with respect to node $p \in N_{seq}$ is a function:

$$order^* : N_{seq} \times N \to \mathbb{N},$$

defined as follows if $path(p, n) \neq \bot$:

$$order^*(p, n) = \begin{cases} order(p, n) \text{ , if } (p, n) \in E, \\ order(p, k) \text{ , where } (p, k) \in E \wedge k \in path(p, n), \text{ if } \nexists(p, n) \in E. \end{cases}$$

Then, the execution order compliance rules can be expressed as follows.

Definition 8. A process model is compliant with rule A `leads to` B if the process model lacks activity A or in the corresponding PST $P = (N, E, r, t)$ $\forall a \in N_a \exists x \in N : t(lcp(a, x)) = \texttt{seq} \wedge order^*(lcp(a, x), a) < order^*(lcp(a, x), x) \wedge exec_{t(x)}(x, b) = true.$

Definition 8 states that a process model is compliant with `leads to` rule in two cases. The first case is when there is no occurrence of activity A in the model. The second case describes the situation when the model contains at least one occurrence of A. Then, for each occurrence of A there must be a fragment which is executed in a sequence after A and which assures that B is always executed within it.

Definition 9. A process model is compliant with rule A `precedes` B if the process model lacks activity B or in the corresponding PST $P = (N, E, r, t)$ $\forall b \in N_b \exists x \in N : t(lcp(b, x)) = \texttt{seq} \wedge order^*(lcp(b, x), x) < order^*(lcp(b, x), b) \wedge exec_{t(x)}(x, a) = true.$

Definitions 8 and 9 reveal that the two compliance rules are structurally similar and symmetric. Profiting from this similarity, in the remainder of the paper we will discuss only one compliance rule type, namely `leads to`. All the conclusions made for `leads to` rule can be trivially deduced for `precedes`.

5   Catalog of Violation Patterns

In this section we present the catalog of compliance rule violation patterns. For each pattern we give the name and briefly discuss the problem it addresses. We motivate patterns providing examples. Finally, we formalize patterns in terms of PST.

Compliance rule A `leads to` B is violated in two cases: either there is no execution path leading from A to B, or there is an execution path leading from A

(a) PST representation   (b) Example of inverse order violation
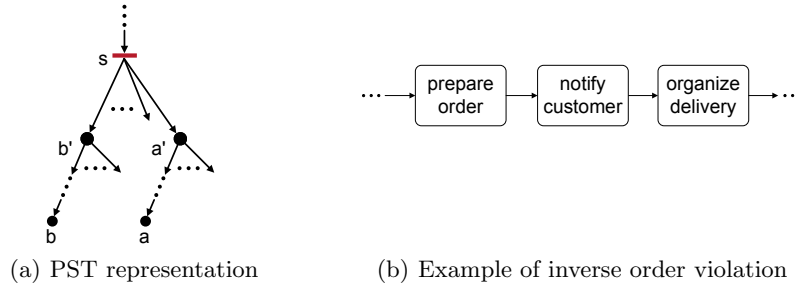
Fig. 3. *Inverse order* violation

to the process end, but not containing B. These two cases are addressed by the violation patterns: each pattern is related to one of them; together the patterns cover all possible violation cases.

The suggested resolution strategies for each violation pattern are given in terms of adding, removing, or moving operations of activities under studies. These operations might be executed recursively on other activities that are related to the activities addressed by the compliance rule as will be shown later in Section 6.

5.1   Inverse Order

Probably the most challenging violation of rule A *leads to* B is the case when activities A and B appear in the inverse order. This means that a process model contains both activities A and B, connected with a path, but this path leads from B to A. Obviously, a compliance rule can not hold.

The inverse order violation can be illustrated by the following example. Assume that a company sends a notification with an order summary to a customer once the order is prepared. Afterwards, the company contacts its logistics partner to arrange a delivery. Fig. 3(b) captures a fragment of the model corresponding to this process. New business conditions require the company to include the delivery information in the notification, i.e. first the delivery should be organized. This requirement is captured in the rule *Organize delivery* `leads to` *Notify customer* rule. In this case an inverse order violation takes place. Fig. 3(a) captures this violation type in terms of PST and the following definition formalizes it.

Definition 10.   Process model has a violation of compliance rule A `leads to` B and this violation is of type *inverse order* if the PST for this model contains nodes a and b corresponding to activities A and B, respectively, and $s = lcp(a, b) \wedge t(s) = \mathtt{seq} \wedge order^*(s, b) < order^*(s, a)$.

In general, inverse order violation can be fixed with one of the following operations:

1. Add B directly after A
2. Remove A from process model
3. Move A to the position directly before B
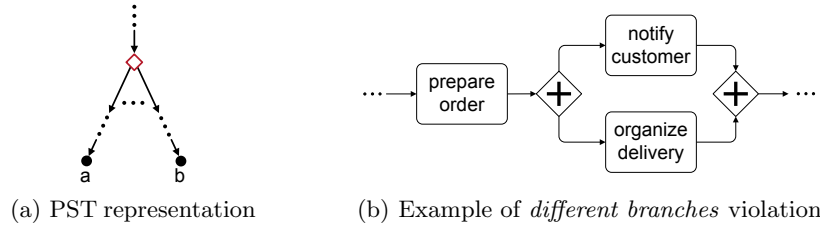4. Move B to the position directly after A

(a) PST representation   (b) Example of *different branches* violation

Fig. 4. *Different branches* violation

5.2 Different Branches

Activities A and B can be located on different branches of a block. Independently of the block type, compliance rule A `leads to` B is violated. From the structural perspective violation of a rule follows from the fact that there is no execution path neither leading from A to B, nor from B to A (see Fig. 4(a)).

The example of *different branches* violation is shown in Fig. 4(b). It presents a variation of the business process discussed in section 5.1. In contrast to the initial example, *Notify customer* and *Organize delivery* activities are executed in parallel. The concurrent activities allow the company to shorten the execution time. However, once the company policy requires to notify a client about delivery details (i.e. *Organize delivery* `precedes` *Notify customer* compliance rule is imposed), the business process becomes incompliant.

Definition 11. Process model has a violation of compliance rule A `leads to` B and this violation is of type *different branches* if the PST for this model contains nodes a and b corresponding to activities A and B, respectively, and $t(lcp(a, b)) \in \{\texttt{and}, \texttt{or}, \texttt{xor}\}$.

The following operations resolve the violation:

1. Remove A from process model
2. Add B to the branch with A directly after A
3. Add B directly after block
4. Add A directly before block (for **and** block)
5. Move A to the branch with B directly before B
6. Move B to the branch with A directly after A
7. Move B directly after block
8. Move A directly before block (for **and** block)

5.3 Splitting Choice

This violation pattern can be motivated by the following example. Assume we have a business process, containing the fragment presented in Fig. 5. The fragment shows that once a purchase request is received, its budget should be approved; before the approval the request can be optionally analyzed. However, one can require that every received purchase request must be analyzed. This
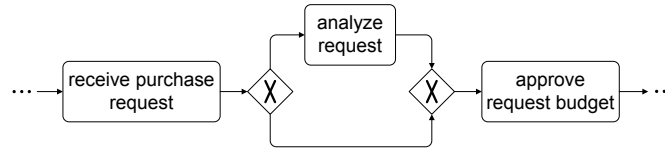
Fig. 5. Example of *splitting choice* violation

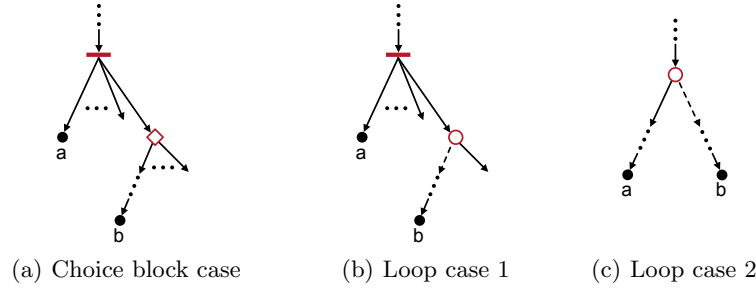(a) Choice block case   (b) Loop case 1   (c) Loop case 2

Fig. 6. Formalization of *Splitting choice* violation in PST

requirement can be formalized in the form of compliance rule *Receive purchase request* `leads to` *Analyze request*. In the presented fragment this rule is violated, since after a purchase request is received, the analysis can be skipped.

We call this type of violation *splitting choice*. `Leads to` compliance rule has a violation of type *splitting choice* if a model contains both activities A and B and, either A and B are connected with a path, but this path contains a split gateway (either `or`, or `xor`), or there is a loop and activity B is on the optional branch, while A is either before the loop or on its mandatory branch (see Fig. 6). Thus, the process model provides an option not to execute B, once A is executed.

Definition 12. Process model has a violation of compliance rule A `leads to` B and this violation is of type *splitting choice* if the PST for this model contains nodes a and b corresponding to activities A and B, such that $s = lcp(a,b) \wedge ((s \in N_{seq} \wedge \exists x \in path(lcp(a,b),b) : x \in N_{or} \cup N_{xor} \cup N_{loop} \wedge exec(x,b) = false) \vee (s \in N_{loop} \wedge exec(s,b) = false))$.

The following operations allow to resolve splitting choice violation:

1. Remove A from process model
2. Remove all the branches which do not contain B
3. Add B to every branch in the choice block
4. Add B in between A and the block, directly after A
5. Add B directly after the block
6. Move A to the branch with B directly before B
7. Move B in between A and the block, directly after A
8. Move B directly after the block

5.4   Lack of Activity

A process model is not compliant with rule A `leads to` B if it has at least one occurrence of A and no occurrence of B. Consider a compliance rule *Receive*

purchase order `leads to` *Close purchase order*. Checking the process model fragment in Fig. 5 against this rule, we see that this rule is violated, since it is missing in the process model.

Definition 13. Process model has a violation of compliance rule A `leads to` B and this violation is of type *lack of activity* if the PST for this model contains node a corresponding to activity A and for a there is no corresponding occurrence of activity B.

The following operations allow to resolve lack of activity violation:

1. Remove A from process model
2. Add B directly after A
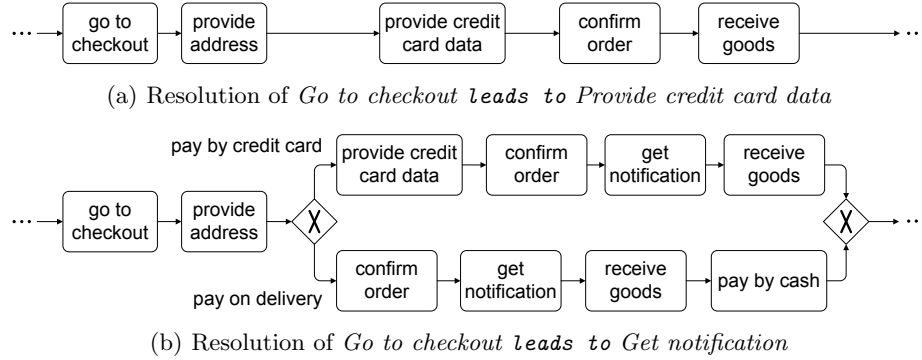
6   Resolution Context

In the previous section we have discussed alternative violation resolutions. Meanwhile, we have not considered how to choose the appropriate operation in a particular case. This choice is driven by the resolution context. In this section we aim at formalizing the context notion and illustrating its influence.

Definition 14. The *resolution context* C is a 7-tuple $(N_{act}, A, T, type, con_{t \in T}, pre_{t \in T}, post_{t \in T})$, where:
  ○ N_{act} is the set activities
  ○ A is the set of objects, which define model aspects
  ○ T is the set of aspect types
  ○ $type : A \to T$ is the function relating each object to a particular aspect
  ○ $con_{t \in T} : \{a : \forall a \in A, type(a) = t\} \times \{a : \forall a \in A, type(a) = t\}$ is the relation between two objects, indicating their contradiction
  ○ $pre_{t \in T} \subseteq N_{act} \times \{a : \forall a \in A, type(a) = t\}$ is the relation defining the prerequisites of activity execution in terms of aspects
  ○ $post_{t \in T} \subseteq N_{act} \times \{a : \forall a \in A, type(a) = t\}$ is the relation defining the result of activity execution in terms of aspects.
One *aspect* of a context is a tuple $(N_{act}, A_t, t, type, con_t, pre_t, post_t)$, where $t \in T \wedge A_t = \{a : a \in A, type(a) = t\}$.

Definition 14 captures aspects with three elements: sets A and T and function *type*. Set A is the set of objects, describing the business environment from a certain perspective, e.g. dependencies of activities on data objects, activities execution constraints, or their semantic annotations. Set T consists of object types; an example is $T = \{controlFlow, dataFlow, semanticAnnotation\}$. Function *type* specifies a type (element of set T) for an element of A. Typification of objects allows distinguishing aspects, e.g. distinguishing a data flow from a semantic description of a process. Relation *con* shows which objects from A contradict each other. A context example with one aspect for a process model in Fig. 1 is the following: $N_{act} = \{$*Confirm order, Get notification, Go to check out, Pay by cash, Provide address, Provide credit card data, Receive goods*$\}$, $A = N_{act}$,

(a) Resolution of *Go to checkout* `leads to` *Provide credit card data*

(b) Resolution of *Go to checkout* `leads to` *Get notification*

Fig. 7. Resolution strategies for violations of different compliance rules

$T = \{CF\}$, $\forall n \in N_{act} type(n) = CF$, $con_{CF} = \{(Provide\ credit\ card\ data,\ Pay\ by\ cash)\}$, $pre_{CF} = \{(Provide\ address,\ Go\ to\ check\ out),\ (Confirm\ order,\ Go\ to\ check\ out),\ (Receive\ goods,\ Confirm\ order),\ (Receive\ notification,\ Confirm\ order)\}$, $post_{CF} = \emptyset$.

Let us look at the examples, illustrating the influence of a context on the resolution. In the example we refer to the fragment in Fig. 1. The fragment is evaluated against two compliance rules: *Go to checkout* `leads to` *Provide credit card data* and *Go to checkout* `leads to` *Get notification*. A thorough inspection reveals that both compliance rules are violated. Furthermore, both violations are of the same type *splitting choice*.

The original business process violates *Go to checkout* `leads to` *Provide credit card data* rule, providing the customer the alternative form of payment (i.e. paying by cash on delivery). At first blush the violation can be fixed with any of the strategies proposed by *splitting choice* pattern. For instance, the activities on the upper branch can be moved to the position before the block. According to the resolution context, the resulting resolution makes the process *inconsistent* because the two contradicting activities *Provide credit card data* and *Pay by cash* appear in sequence. Thus, the violation requires a resolution preserving exactly one required payment option and the corresponding process completion. The most suitable strategy is to remove all the branches alternative to the credit card payment. This makes the business process compliant with the rule and rational at the same time. Fig. 7(a) presents the result of this transformation.

Let us refer to the violation of the second rule. The business process in the example is not compliant with the rule, since the customer receives a notification only if "pay on delivery" method is selected. Although this violation has the same type as in the first example, the resolution strategy employed in the previous case is inappropriate here: a removal of one branch means that only one payment method is accepted. This effect is undesired as this resolution restricts the business process logic too much. Both alternative process evolutions have to be preserved, while a notification delivery should be included in both. Thus, in this case an occurrence of activity *Get notification* is added to the upper branch

after activity *confirm order*, so that *pre* relation holds. The result of a violation resolution is presented in Fig. 7(b).

It is obvious how the availability of the resolution context affects the choice of the resolution strategy. With several resolution strategies applicable, one can choose the strategy with the least change in the process structure, yet consistent with the context.

7   Conclusions and Future Work

In this paper we attempted to address the problem of resolving compliance violations systematically. The problem is extremely complex and, thus, we started with the problem definition. We scoped our work, narrowing the type of addressed rules to execution order compliance rules and setting up the number of important assumptions. We assumed that we can learn a pair of activities which causes a violation and that violations can be resolved independently. Another important assumption is that process models are structured workflows.

The main contribution of this paper are *a)* the catalog of violation patterns, capturing all possible violation types and proposing resolution operations and *b)* the notion of a resolution context, which drives the choice of a suitable operation.

The violation patterns and the resolution context are perceived by us as the basis for the future work: they formalize the knowledge a human expert employs during violation resolution. Therefore, in the next steps we aim at developing algorithms enabling the choice of resolution strategies basing on the context. Another interesting question is to what extent the automatic resolution is possible, i.e. what kinds of violations can be automatically fixed and which can not.

References

1. A. Awad, G. Decker, and M. Weske. Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In *BPM*, volume 5240 of *LNCS*, pages 326–341. Springer, 2008.
2. A. Awad, A. Polyvyanyy, and M. Weske. Semantic querying of business process models. In *EDOC*, pages 85–94. IEEE Computer Society, 2008.
3. A. Awad and M. Weske. Visualization of Compliance Violation Using Anti-patterns. Technical Report 2, Business Process Technology Group at Hasso Plattner Institute, 2009.
4. M. El Kharbili, S. Stein, I. Markovic, and E. Pulvermüller. Towards a Framework for Semantic Business Process Compliance Management. In *GRCIS*, pages 1–15. CEUR-WS.org, June 2008.
5. A. Förster, G. Engels, T. Schattkowsky, and R. Van Der Straeten. Verification of Business Process Quality Constraints Based on Visual Process Patterns. In *TASE*, pages 197–208. IEEE Computer Society, 2007.
6. A. Ghose and G. Koliadis. Auditing Business Process Compliance. In *ICSOC*, volume 4749 of *LNCS*, pages 169–180. Springer, 2007.
7. S. Goedertier and J. Vanthienen. Compliant and Flexible Business Processes with Business Rules. In *BPMDS*, volume 236 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

8. S. Goedertier and J. Vanthienen. Designing Compliant Business Processes from Obligations and Permissions. In *BPD*, volume 4103 of *LNCS*, pages 5–14. Springer Verlag, 2006.

9. G. Governatori and Z. Milosevic. Dealing with Contract Violations: Formalism and Domain Specific Language. In *EDOC*, pages 46–57. IEEE Computer Society, 2005.

10. G. Governatori, Z. Milosevic, and S. Sadiq. Compliance Checking between Business Processes and Business Contracts. In *EDOC*, pages 221–232, Washington, DC, USA, 2006. IEEE Computer Society.

11. T. Hartman. *The Cost of Being Public in the Era of Sarbanes-Oxley*. [Chicago, Ill.] : Foley & Lardner, June 2006.

12. R. Johnson, D. Pearson, and K. Pingali. The Program Structure Tree: Computing Control Regions in Linear Time. *SIGPLAN Not.*, 29(6):171–185, 1994.

13. R. Liu and A. Kumar. An Analysis and Taxonomy of Unstructured Workflows. In *BPM*, pages 268–284, 2005.

14. R. Lu, S. Sadiq, and G. Governatori. Compliance Aware Business Process Design. In *BPM Workshops*, volume 4928 of *LNCS*, pages 120–131. Springer, 2007.

15. R. Lu, S. Sadiq, and G. Governatori. Measurement of Compliance Distance in Business Processes. *Inf. Sys. Manag.*, 25(4):344–355, 2008.

16. Y. Lui, S. Müller, and K. Xu. A Static Compliance-checking Framework for Business Process Models. *IBM SYSTEMS JOURNAL*, 46(2):335–362, 2007.

17. L. T. Ly, K. Göser, S. Rinderle-Ma, and P. Dadam. Compliance of Semantic Constraints—A Requirements Analysis for Process Management Systems. In *GRCIS*, pages 16–30. CEUR-WS.org, June 2008.

18. Z. Milosevic, S. Sadiq, and M. Orlowska. Translating Business Contract into Compliant Business Processes. In *EDOC*, pages 211–220. IEEE Computer Society, 2006.

19. K. Namiri and N. Stojanovic. Pattern-Based Design and Validation of Business Process Compliance. In *OTM Conferences*, volume 4803 of *LNCS*, pages 59–76. Springer, 2007.

20. S. Rinderle, M. Reichert, and P. Dadam. Evaluation of Correctness Criteria for Dynamic Workflow Changes. In *BPM*, volume 2678 of *LNCS*, pages 41–57. Springer, 2003.

21. S. Sadiq, G. Governatori, and K. Namiri. Modeling Control Objectives for Business Process Compliance. In *BPM*, volume 4714 of *LNCS*, pages 149–164. Springer, 2007.

22. W. M. P. van der Aalst and T. Basten. Inheritance of Workflows: an Approach to Tackling Problems Related to Change. *Theor. Comput. Sci.*, 270(1-2):125–203, 2002.

23. J. Vanhatalo, H. Völzer, and F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In *ICSOC*, volume 4749 of *LNCS*, pages 43–55. Springer, 2007.

24. B. Weber, M. Reichert, and S. Rinderle-Ma. Change Patterns and Change Support Features - Enhancing Flexibility in Process-aware Information Systems. *Data Knowl. Eng.*, 66(3):438–466, 2008.

25. J. Yu, T. P. Manh, J. Han, Y. Jin, H. Y., and J. Wang. Pattern Based Property Specification and Verification for Service Composition. In *WISE*, volume 4255 of *LNCS*, pages 156–168. Springer, 2006.