Towards an Abstract Framework for Compliance

Silvano Colombo Tosatto*†, Guido Governatori† and Pierre Kelsen*
*University of Luxembourg, Luxembourg
{silvano.colombotosatto, pierre.kelsen}@uni.lu
†NICTA, Australia
{guido.governatori}@nicta.com.au
†Universitá di Torino, Italy

Abstract—The present paper aims to provide an abstract framework for the regulatory compliance problem. In particular we focus on the problem of deciding whether a structured process is compliant with a single regulation, which is composed of a primary obligation and a chain of compensations.

Keywords—Compliance, Normative Reasoning, Contrary to Duty Obligations.

I. INTRODUCTION

Compliance initiatives are becoming more and more important in enterprises with the increase of the number of regulatory frameworks explicitly requiring businesses to show compliance with them. As a consequence IT support for compliance activities within enterprises is growing. Compliance is the set of initiatives in an organization to ensure that the core business activities and procedures are in agreement with specific normative frameworks. Most compliance solutions are ad hoc solutions and typically, implementation and maintenance are time consuming [1]. Sadiq and Governatori propose in [2] a classification of compliance activities in preventive and detective. Auditing is a typical example of a detective activity, where logs of already executed business activities are examined to discover if there were non compliance issues, and based on the analysis of samples recommendations on measures to prevent recurrences of compliance breaches are proposed. Preventive solutions, on the other hand, consider the activities to be done to achieve business objectives and their interactions with and the impact on them of the obligations and prohibitions imposed on a business by a normative framework or regulation.

The implementation of a particular regulation is conducted as part of a compliance initiative and involves the production of so-called compliance artifacts. These artifacts are used to represent compliance requirements and check them against the enterprise information systems. Usually, regulatory frameworks explicitly require compliance initiatives to provide proof of compliance. [3] advances a compliance-by-design methodology to address the compliance problem. The methodology is based on the use of business process models to describe the activities of an enterprise and to couple them with formal specifications of the regulatory frameworks regulating the business. Business process models describe the task (activities) to be done, and the order in which the task can be executed. [4] argues that this is not enough to ensure that a process is compliant, because, normative frameworks often specify requirements on other aspects (e.g., data, resources, timeframes, relations between different pieces of data and resources). The solution to obviate this problem is to extend business processes with semantic annotations. Tasks in a process are associated with sets of annotations providing information not typically available in a business process model. Several approaches to handle compliance and to formalize normative requirements, based on different logical formalisms have been proposed, see for example ([5], [6], [7], [8], [9]). [10] gives a comparative analysis of a collection of solutions to business process compliance management.

From the above point of view, compliance can be understood as an additional correctness criterion for a business processes. Verification of business process is a very well studied area (see for example the seminal paper by van der Aalst [11]). However, compliance adds complexity to the verification task: the structural correctness of structured processes can be verified in linear time [12], but [13] shows that, even for structured processes checking, whether a process is compliant with a (formalised) legislation, is computationally hard (i.e., checking whether there is at least one possible way to execute the process without violating the norms is an NP-complete problem). Accordingly, there is a need to identify heuristics of classes of tractable problems.

The aim of this paper is not to propose yet another formalism for business process compliance, but to offer an abstract framework capable to verify whether a business process is compliant with a given regulation. A regulation consists of a primary norm to which a chain of compensations, that have to be fulfilled in case the primary norm is not, can be associated. Usually, norms are classified as duty norms or contrary to duty norms. The representation of such type of regulation is the following: if the primary norm is not fulfilled, a chain of compensations should be applied, otherwise the process is compliant.

To achieve this goal we propose an abstract formalism for business process compliance. The advantages of an abstract formalism are that the framework highlights the crucial aspects of compliance without worrying about the details of a specific formalism. An additional benefit of the abstract framework is that it allows scholars the classifications of approaches to compliance based on the features identified by the abstract formalism. In this way we can study general properties of business process compliance.

The paper is structured as follows: Section 2 defines the compliance problem by introducing the definitions of processes and regulations. Section 3 describes the abstract framework for compliance. Section 4 concludes the paper.
II. BUSINESS PROCESS REGULATORY COMPLIANCE

In this section we first introduce the business process regulatory compliance problem and its elements: the business process model and norms.

A. Business Process

We consider a particular class of processes: the structured processes. Such class of processes is limited in its expressivity because it does not allow cycles and its components have to be properly nested. An advantage of using structured processes is that their correctness can be verified in polynomial time as shown in [16] by Kiepuszewski et al. for structured workflow models. We define our processes in a similar way as the workflows defined by Kiepuszewski et al.

Definition 1 (Process Block): A process block $B$ is a directed graph: the nodes are called elements and the edges are called transitions. The set of elements of a process block are identified by the function $V(B)$ and the set of edges by the function $E(B)$. The set of elements is composed of tasks and coordinators. The coordinators are of 4 types: and_split, and_join, xor_split and xor_join. Each process block $B$ has two distinguished nodes called the initial and final element. The initial element has no incoming transition from other elements in $B$, by $b(B)$. Similarly the final element has no outgoing transitions to other elements in $B$, by $f(B)$.

A process block is defined inductively as follows:

- A single task constitutes a process block. The task is both the initial and the final element of the block.
- Let $B_1,\ldots,B_n$ be process blocks:
  - $\text{SEQ}(B_1,\ldots,B_n)$ denotes the process block with node set $\cup V(B_i)$ and edge set $\cup E(B_i) \cup \{(f(B_i), b(B_{i+1})) : 1 \leq i < n\}$.
  - $\text{XOR}(B_1,\ldots,B_n)$ denotes the block with vertex set $\cup V(B_i) \cup \{x\text{split}, x\text{join}\}$ and edge set $\cup E(B_i) \cup \{(x\text{split}, b(B_i)), (f(B_i), x\text{join}) : 1 \leq i \leq n\}$ where $x\text{split}$ and $x\text{join}$ denote a xor_split coordinator and an xor_join coordinator, respectively.
  - $\text{AND}(B_1,\ldots,B_n)$ denotes the block with vertex set $\cup V(B_i) \cup \{a\text{split}, a\text{join}\}$ and edge set $\cup E(B_i) \cup \{(a\text{split}, b(B_i)), (f(B_i), a\text{join}) : 1 \leq i \leq n\}$ where $a\text{split}$ and $a\text{join}$ denote an and_split coordinator and an and_join coordinator, respectively.

Definition 2 (Structured Process Model): The coordinators start and end are used to identify the beginning of a structured process model and its ending. A structured process model $P$ is a directed graph composed of a process block $B$ called the main process block, with vertex set $V(B) \cup \{\text{start}; \text{end}\}$ and edge set $E(B) \cup \{(\text{start}, b(B)), (f(B), \text{end})\}$.

Structured process models can be graphically represented using Business Process Model Notation 2.0\footnote{http://www.omg.org/spec/BPMN/2.0}. The symbol $\bigcirc$ is used to represent the start coordinator and $\blacklozenge$ to represent the end coordinator. The and_split and and_join coordinators are represented both by $\blacklozenge$. The and_split is identified by a single incoming transition and multiple outgoing transitions. The opposite is true for the and_join, which is identified by multiple incoming transitions and a single outgoing transition. In the same way, the operator $\blacklozenge$ identifies both xor_split and xor_join coordinators.

In a structured process model XOR blocks and AND blocks have to be properly nested, meaning that if the block $A$ starts inside the block $B$, $A$ has to end within $B$. An example of a structured process model is shown in Fig. 1.

Example 1: Fig. 1 shows a structured process containing four tasks labelled $t_1,\ldots,t_4$. The structured process contains an XOR block delimited by the xor_split and the xor_join. The XOR block contains the tasks $t_1$ and $t_2$. The XOR block is itself nested inside an AND block with the task $t_5$. The AND block is preceded by the start and followed by task $t_4$ which in turn is followed by the end.

![Fig. 1. A graphical representation of a structured process](image1)

Considering the structured process in Fig. 1 as $\text{SEQ}(\text{start}, B, \text{end})$. We can represent the the main block of the structured process using a prefix notation as follows: $B = \text{SEQ}(\text{AND}(\text{XOR}(t_1, t_2), t_3), t_4)$.

Fig. 2 shows two models which are no structured processes. Illustration (a) is not a structured process because of the two badly nested blocks. Illustration (b) is not a structured process either because it contains a cycle. Because these two models not being structured processes, we cannot use a prefix notation to represent their main block as we did for the model in Fig. 1.

![Fig. 2. Examples of non-structured processes](image2)
An execution of a structured process is a sequence of a subset of the tasks belonging to the process. A valid execution identifies a path from the start to the end of the process and follows the semantics of the coordinators and the transitions that are traversed.

**Definition 3 (Process Block Serialization):** Given a process block \( B \), a serialization \( \epsilon = (t_1, \ldots, t_n) \) is a sequence of tasks constructed from \( B \) as follows:

1. If \( B \) is a single task \( t \), then \( t \in \epsilon \) and is unique.
2. If \( B = \text{SEQ}(B_1, \ldots, B_n) \), then each \( B_i \) is serialized and the tasks in \( B_z \) cannot be serialized before the tasks in \( B_k \) if \( z > k \).
3. If \( B = \text{XOR}(B_1, \ldots, B_n) \), then exactly one \( B_i \) is serialized.
4. If \( B = \text{AND}(B_1, \ldots, B_n) \), then each \( B_i \) is serialized.

We identify the set of possible serializations of a process block \( B \) as \( \Sigma(B) \).

**Definition 4 (Process Execution):** Given a structured process \( P \) whose main process block is \( B \), \( \Sigma(P) = \Sigma(B) \). Each \( \epsilon \in \Sigma(P) \) is called execution of \( P \).

If a structured process is conformed with Definition 2, then a task belonging to such structured process appears in at least one of its executions. This means that each task contained in a process has the possibility to be executed as stated in the following lemma.

**Lemma 1 (Process Block Serialization):** Given a process block \( B \), \( \forall t \in B \) such that \( t \) is a task, \( \exists \epsilon \in \Sigma(B) \) such that \( t \in \epsilon \). This means that each task belonging to process block has to appear in at least one of its possible serializations. In other words a process block contains only tasks which can be serialized.

**Block Serialization:** We prove this by contradiction. Assume that: \( \forall \epsilon \in \Sigma(B), t \notin \epsilon \).

From the assumption and Definition 3 follows that \( B \) itself or any of the process blocks contained in \( B \), cannot contain a process block \( B' \) and \( B' = t \).

From Definition 1, because \( \neg \exists B' \) contained in \( B \), follows that \( t \notin B \) which contradicts the premises.

**Example 2:** Considering the structured process shown in Fig. 1 as \( P \). We have that \( \Sigma(P) = \{ \epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4 \} \) where \( \epsilon_1 = (t_1, t_3, t_4), \epsilon_2 = (t_2, t_3, t_4), \epsilon_3 = (t_3, t_1, t_4) \) and \( \epsilon_4 = (t_3, t_2, t_4) \). \( \Sigma(E) \) contains the four possible executions of the process \( P \). An execution not contained in \( \Sigma(E) \), like \( \epsilon_5 = (t_3, t_4, t_1) \), is not a valid execution of \( P \). In this case \( \epsilon_5 \) is not a valid execution of \( P \) because \( t_1 \) is serialized after task \( t_4 \). The serialization in \( \epsilon_5 \) is not correct because it violates the second requirement of Definition 4.

The state of a process can evolve during the execution of the process. An annotated process is a process whose tasks are associated with consistent sets of literals. These set of literals are called *annotations* [17] and determine how the state of the process changes when a task is executed.

**Definition 5 (Consistent literal set):** A set of literals \( L \) is consistent if and only if it does not contain both \( l \) and its complement \( \bar{l} \) for each literal \( l \in L \). Where the \( \bar{l} \) denotes the negation of literal \( l \).

**Definition 6 (Annotated process block):** Let \( B \) be a process block and let \( T \) be the set of tasks contained in \( B \). An annotated process block is a pair: \((B, \text{ann})\), where \text{ann} is a function associating to each task in \( T \) a consistent set of literals, \text{ann} : T \rightarrow 2^L.

**Definition 7 (Annotated process):** Let \( P \) be a structured process and let \( T \) be the set of tasks contained in \( P \). An annotated process is a pair: \((P, \text{ann})\), where \text{ann} is a function associating to each task in \( T \) a consistent set of literals, \text{ann} : T \rightarrow 2^L.

**Example 3:** Fig. 3 shows an annotated process with the same topology of the structured process previously shown in Fig. 1. The annotation of each task of the process is shown as a set above the task itself. The annotations indicate what has to hold after a task is executed. If \( t_1 \) is executed, then the literal \( a \) has to hold in the state of the process.

![Fig. 3. An annotated process](image)

If we represent an annotated process using the prefix notation, then writing the annotations above each task is not a viable option. In this case the prefix representation of the structured process is integrated with a list describing the annotation of each task. For this example the list is the following:

- \( \text{ann}(t_1) = \{a\} \)
- \( \text{ann}(t_2) = \{b, c\} \)
- \( \text{ann}(t_3) = \{c, d\} \)
- \( \text{ann}(t_4) = \{\neg a\} \)

**Definition 8 (Process State):** The state of a process is represented by a pair \((t, L)\) where \( L \) is the set of literals holding after the execution of the task \( t \).

We define an update process (inspired by AGM belief revision [18]).

**Definition 9 (Literal set update):** Given two consistent sets of literals \( L_1 \) and \( L_2 \), the update of \( L_1 \) with \( L_2 \), denoted by \( L_1 \oplus L_2 \), is a set of literals defined as follows:

\[ L_1 \oplus L_2 = L_1 \setminus \{ \bar{t} \mid t \in L_2 \} \cup L_2 \]

\(^2\)If \( t = a \), then \( \bar{t} = \neg a \). If \( t = \neg a \), then \( \bar{t} = a \).
Process Compliance Logic (PCL) [19].

A trace represents a sequence of snapshots of the process state during one of its executions. It is composed by a sequence of pairs, where each pair contains a task belonging to the execution, represented by the trace, and a process state holding after the execution of such task. We distinguish two types of traces; a process trace identifies the sequence of states from an execution of a process and a block trace identifies the sequence of states from a serialization of a process block.

Definition 10 (Block Trace): Given a serialization $\epsilon = (t_1, L_1), \ldots, (t_n, L_n))$. Each $L_i$ is a set of literals such that:

1) $L_i \supseteq \emptyset \oplus \text{ann}(t_i)$
2) $L_{i+1} = L_i \oplus \text{ann}(t_{i+1})$, for $1 \leq i < n$.

$\Theta(B, \text{ann})$ represents the set of traces of an annotated process block.

Example 4: Table 4 shows the traces of the annotated process block $(B, \text{ann})$. In this example we use the main block $B$ and $\text{ann}$ defined in Example 3. The first column contains the possible serializations of $B$. The second column contains the corresponding traces.

Definition 11 (Process Trace): Given an execution $\epsilon = (t_1, \ldots, t_n)$, a process trace $\theta$ is a finite sequence of states $((\text{start}, L_0)(t_1, L_1), \ldots, (t_n, L_n), (\text{end}, L_{n+1}))$, capturing the state of the process holding after the execution of a task. Each $L_i$ is a set of literals such that:

1) $L_0 = \emptyset$
2) $L_{i+1} = L_i \oplus \text{ann}(t_{i+1})$, for $1 \leq i < n$.
3) $L_{n+1} = L_n$.

$(\text{start}, L_0)$ represents the state holding before $\epsilon$ and $(\text{end}, L_{n+1})$ the state holding after $\epsilon$. $\Theta(F, \text{ann})$ represents the set of traces of an annotated process.

Example 5: Table 5 shows the traces of the annotated process $(P, \text{ann})$ illustrated in Fig. 3. The first column contains the possible executions of $P$. The second column contains the corresponding traces.

B. Regulations

A regulation $R$ is composed of a primary norm followed by a chain of compensations $R = \mathcal{O} \otimes \zeta$, where $\mathcal{O}$ represent the primary norm and $\zeta$ the chain of compensations. A chain of compensations consists on a sequence of secondary norms $\zeta = \Omega_1 \otimes \cdots \otimes \Omega_n$. If the primary norm is not fulfilled, then the first secondary norm of the chain is enforced and has to be fulfilled. The same applies when a secondary norm of the chain is enforced and not fulfilled, the difference is that in such case the following secondary norm of the chain is enforced.

We specify the primary and secondary norms using part of Process Compliance Logic (PCL) [19].

Each norm has a lifeline and a deadline. These elements define the activation period of the norm. Once triggered by its lifeline, a norm become active. If a norm is already active, further triggers of its lifeline have no effect. Similarly when its deadline is triggered, a norm is deactivated. The last state of a trace deactivates every norm.

When active, a norm enforces an obligation. We identify two types of obligations: achievement and maintenance. An achievement obligation has to verify a condition in at least a state within the activation period of the norm containing it. A maintenance obligation has to verify a condition in each state within the activation period of the norm containing it.

Definition 12 (Primary Norm): Let $l_c, l_b$ and $l_d$ be literals. A primary norm $\mathcal{O}$ is a triple $\mathcal{O} = \langle \zeta, l_b, l_d \rangle$ where $l_b$ is the lifeline condition, $l_d$ is the deadline condition and $\zeta$ is one of the following types of obligations where $l_c$ is their fulfillment condition:

$$\mathcal{O} := \begin{cases} O^a(l_c) & \text{achievement} \\ O^m(l_c) & \text{maintenance} \end{cases}$$

Literals are satisfied in a state if and only if such state contains them. Lifelines and deadlines activate or deactivate a norm in the in states that satisfy them. The activation period of a norm is identified by all the states between the state where the lifeline is satisfied and the state where the deadline is satisfied, including it.

Achievement obligations prematurely terminate the activation period of the norm they belong to in the state where they are fulfilled. Differently, for maintenance obligations, their activation period is prematurely terminated when a state do not fulfill the condition of the obligation.

Definition 13 (Obligation Fulfillment): Let $L \models l$ iff $l \in L$. We also define the following exceptions: each deadline $l_d$ is always $L_{\text{end}} \models l_d$ and each lifeline $l_c$ is always $L_{\text{end}} \not\models l_c$.

Given a norm $\mathcal{O} = \langle \zeta, l_b, l_d \rangle$ and a trace $\theta$, $\theta$ fulfills $\mathcal{O}$, written $\theta \vdash \mathcal{O}$, iff:

$$\mathcal{O} = \begin{cases} O^a(l_c) & \theta \vdash (O^a(l_c), l_b, l_d) \iff \forall (t_i, L) \in \theta \text{ such that } L_i \models l_b \text{ then } \exists(t_j, L_j) \in \theta \text{ such that } L_j \models l_c \text{ and } j > i \text{ and } \exists(t_h, L_h) \in \theta \text{ such that } L_h \models l_d \text{ and } i < h < j. \\ O^m(l_c) & \theta \vdash (O^m(l_c), l_b, l_d) \iff \forall (t_i, L_i) \in \theta \text{ such that } L_i \models l_b \text{ then } \exists(t_j, L_j) \in \theta \text{ such that } L_j \models l_c \text{ and } \forall (t_i, L_i) \in \theta \text{ such that } L_i \models l_d \text{ and } \forall (t_j, L_j) \in \theta \text{ such that } L_j \models l_c \text{ and } i < j \leq h. \end{cases}$$

Otherwise $\theta$ does not fulfills $\mathcal{O}$, written $\theta \not\vdash \mathcal{O}$.

Example 6 (Achievement): In a game of chess, moving a piece is an achievement obligation triggered by the opponent move. The deadline of such obligation can be considered as the time allowed for the player to make his move. Thus the player has to make his move before the allotted time expires.

Example 7 (Maintenance): While accessing secure data there is the obligation to have the proper credentials for the whole access period. The lifeline is when the access starts and the deadline when the access ends. For the whole time period of the access the credential must be retained.
C. Primary Norm Compliance

Given a primary norm, an annotated process can be fully compliant, partially compliant or not compliant with such primary norm. An annotated process is fully compliant if each trace fulfills the norm. It is partially compliant, if at least one trace fulfills the norm. If none of the traces fulfill the norm, then the annotated process is not compliant.

Definition 14 (Process Primary Norm Compliance):
Given an annotated process \((P, \text{ann})\) and a norm \(\mathcal{O}\).

- **Full Compliance**: \((P, \text{ann}) \models^P \mathcal{O}\) if \(\forall \theta \in \Theta(P, \text{ann}), \theta \vdash \mathcal{O}\).
- **Partial Compliance**: \((P, \text{ann}) \models^P \mathcal{O}\) if \(\exists \theta \in \Theta(P, \text{ann}), \theta \vdash \mathcal{O}\).
- **Not Compliant**: \((P, \text{ann}) \not\models^P \mathcal{O}\) if \(\exists \theta \in \Theta(P, \text{ann}), \theta \not\vdash \mathcal{O}\).

Full compliance implies partial compliance. This means that if a process is fully compliant with a norm, then such process is also partially compliant with the same norm.

D. Compensation Chains

Norms are often described as soft constraints due to the possibility of that they can be violated. Thus, as happens in the real world, it is useful to consider what should be done in case a norm is violated.

Compensations are used to define the behavior that should be adopted in the case a norm is violated. A compensation is associated to a norm and it defines a secondary norm that is activated when the one preceding it is violated. Secondary norms are a particular type of norms which lifeline is the violation of the norm they try to compensate.

Definition 15 (Violation): Given a norm \(\mathcal{O} = \langle \Omega, l_0, l_1, \ldots, l_k, \text{end}, \text{end} \rangle\) and a trace \(\theta = ((\text{start}, L_0), (t_1, L_1), \ldots, (t_k, L_k), (\text{end}, \text{end}))\). If \(\theta \not\vdash \mathcal{O}\), then \(\exists L_i \in \theta\) where \(\mathcal{O}\) is violated. Each state in \(\theta\) where \(\mathcal{O}\) is violated is labeled as \(V_\mathcal{O}\).

If \(\mathcal{O} = O^c(l_c)\), then \(V_\mathcal{O} = \forall L_i\) such that \(L_i \models l_b, \exists L_h\) such that \(L_h \models l_d\) and \(\exists L_j\) such that \(L_j \models l_c\) and \(i < j \leq h\).

If \(\mathcal{O} = O^m(l_c)\), then \(V_\mathcal{O} = \forall L_i\) such that \(L_i \models l_b, \exists L_h\) such that \(L_h \models l_d\) and \(\exists L_j\) such that \(L_j \models l_c\) and \(i < h \leq j\).

Because compensations are also norms, it means that they can be also violated. It is possible then to associate a compensation to another compensation. This way of assigning compensations can create a sequence of norms, where a norm is activated when the previous is violated. We call such sequence of secondary norms a chain of compensations.

Definition 16 (Compensation Chain): Let \(l_c\) be a literal. Given a primary norm \(\mathcal{O}\), \(\tau\) is the chain of compensations for \(\mathcal{O} \otimes \tau\).

\[\tau = \Omega\]
\[\tau = \Omega \otimes \tau\]

\(\Omega = \langle \Omega, l_d \rangle\) where \(\mathcal{O}\) can be either \(O^c(l_c)\) or \(O^m(l_c)\), and \(l_d\) represents the deadline condition. The activation period of a secondary obligation \(\Omega\) is considered to be the sequence of states starting from the one violating the previous norm till the one fulfilling the deadline condition. The state violating the previous norm is included in the activation period and the state fulfilling the deadline condition is not. Thus the lifeline for a secondary obligation \(\Omega\) can be implicitly considered to be triggered by the state preceding the state violating the previous norm.

Definition 17 (Compensation Lifeline): Given a norm with a chain of compensations associated \(\mathcal{O} \otimes \tau\), where \(\tau\) is a sequence of one or more secondary norms \(\Omega_1 \otimes \cdots \otimes \Omega_n\). Let \(V_\tau\) be a special literal where \(y\) refers to the previous norm in the chain.

The first compensation of \(\tau\) is \(\Omega_1 = \langle \Omega, l_{V_\tau}, l_d \rangle\) and each other compensation in the chain is \(\Omega_m = \langle \Omega, l_{V_{\tau m-1}}, l_d \rangle\).

Given a state \(L_i \in \theta, L_i \models l_{V_\tau}\) iff \(L_{i+1}\) has been labeled as \(V_\tau\) according to Definition 15.

Given a norm with a compensation associated \(\mathcal{O} \otimes \tau\), knowing that \(\Omega = \langle \Omega, l_d \rangle\), we can handle the secondary norm \(\Omega\) like a primary norm as described in Definition 13 by including the violations of \(\mathcal{O}\) as the lifelines of \(\Omega\) as follows \(\Omega = \langle \Omega, l_{V_\tau}, l_d \rangle\). In this way the fulfillment of a secondary norm can be handled as a primary norm. This works in the same way when we have a secondary norm followed by another \(\mathcal{O} \otimes \Omega_1 \otimes \Omega_2\), in this case the lifeline of \(\Omega_2\) is considered to be \(l_{V_{\tau 1}}\).

Definition 18 (Compensation Fulfillment): Given a trace \(\theta\) and a norm to which is associated a chain of compensations \(\mathcal{O} \otimes \tau\), where \(\tau = \Omega_1 \otimes \cdots \otimes \Omega_n\).

- **Norm Compliance**: \(\theta \models^P \mathcal{O} \otimes \tau\) iff \(\exists \Omega_i \in \tau\) such that \(\theta \vdash \mathcal{O}\).
- **Compensation Compliance**: \(\theta \models^P \mathcal{O} \otimes \tau\) iff \(\exists \Omega_i \in \tau\) such that \(\theta \vdash \mathcal{O}\).
- **Not Compliant**: \(\theta \not\models^P \mathcal{O} \otimes \tau\) iff \(\exists \Omega_i \in \tau\) such that \(\theta \not\vdash \mathcal{O}\).

From Definition 13 we know that a norm whose lifeline is never fulfilled is considered to be fulfilled. Thus from this observation we can state that norm compliant implies compensation compliant, because if the primary obligation is fulfilled, which is the case for norm compliance, then none of the secondary obligations are activated and each of them is considered to be fulfilled.

Definition 19 (Process Compensation Compliance):
Given an annotated process \((P, \text{ann})\) and a primary norm with a chain of compensations associated \(\mathcal{O} \otimes \tau\).
Full Norm Compliance: \((P, \Omega) \models_{\text{FN}} \Theta \otimes \sharp\) if \(\forall \theta \in \Theta(P, \Omega), \theta \not\vdash \Theta \otimes \sharp\).

Full Compensation Compliance: \((P, \Omega) \models_{\text{FC}} \Theta \otimes \sharp\) if \(\forall \theta \in \Theta(P, \Omega), \theta \not\vdash \Theta \otimes \sharp\).

Partial Norm Compliance: \((P, \Omega) \models_{\text{PN}} \Theta \otimes \sharp\) if \(\exists \theta \in \Theta(P, \Omega), \theta \vdash \Theta \otimes \sharp\).

Partial Compensation Compliance: \((P, \Omega) \models_{\text{PC}} \Theta \otimes \sharp\) if \(\exists \theta \in \Theta(P, \Omega), \theta \vdash \Theta \otimes \sharp\).

Not Compliant: \((P, \Omega) \not\models \Theta \otimes \sharp\) if \(\not\exists \theta \in \Theta(P, \Omega), \theta \vdash \Theta \otimes \sharp\).

We can see from Definition 19 that full norm compliance implies both full compensation compliance and partial norm compliance. Both full compensation compliance and partial norm compliance imply partial compensation compliance.

When a structured process is partially norm compliant with a regulation, it can be also be the case that it is fully compensation compliant. However this is not always the case, otherwise Partial Norm Compliance would have implied Full Compensation Compliance. Because of this, the abstract framework proposed in the following section explicitly distinguish whether the structured process is only partially norm compliant or also fully compensation compliant.

III. Abstract Framework

In this section we present the abstract framework. Given a structured process and a regulation, the abstract framework determines whether the structured process is compliant with the given regulation.

The abstract framework uses both algorithm and procedures. We use as an algorithm a more detailed sequence of instructions and it is used when the details of the computation are relevant. Differently the procedures used in the framework are defined as interfaces, where only the properties of the input and the output are described in details.

Framework 1: \(F(P, \mathcal{R})\)

The input of the abstract framework is a structured process \(P = \text{SEQ}((\text{start}, B, \text{end}))\) and a regulation \(\mathcal{R} = \Theta \otimes \sharp\). The regulation is composed by a primary norm with a chain of compensations associated \(\Theta \otimes \sharp\). The primary norm is expressed as \(\Theta = \langle \Omega, l_0, l_4 \rangle\) and the chain of compensations \(\sharp = \Omega_0 \otimes \cdots \otimes \Omega_n\). Each compensation in the chain is expressed as \(\Omega = \langle \Omega, l_4 \rangle\). We handle \(\sharp\) as a vector, meaning that \(\sharp_i\) returns the \(i\)-th element in the chain. In case the index given to \(\sharp\) is greater than the length of the chain or in case the chain is empty, \(\text{null} \) is returned.

\[
pool = \text{Primary}(B, \Theta)
\]

The abstract framework uses the algorithm Primary, which has the task to verify for which subsets of the possible traces the primary obligation is fulfilled. When a subset of traces does not fulfill the primary obligation, the algorithm takes care to pass such subset to the algorithm in charge to verify if the eventual compensations are fulfilled in such subset.

The results produced by the algorithm Primary and some of its sub procedures are collected in the set pool. The set pool is then analyzed by the algorithm Compliance which based on the content of such set decides whether \(P\) is compliant with \(\mathcal{R}\) and to which extent.

Algorithm 1 (Primary): \(pool = \text{Primary}(B, \Theta)\)

The algorithm Primary takes as input a process block \(B\) and a primary norm \(\Theta\). The output of the algorithm is a set of elements, where each of the elements describe whether a subset of the possible traces in \(P\) are compliant with \(\Theta\).

\[
\{B_1\} = \text{Life}(B, l)
\]

\[
\forall B : \{B\}, \{B'\} = \text{State}(B', l_0, l_4)
\]

\[
\forall B'' : \{B_1\}, \{B_2\} = \text{State}(B'', l_0, l_4)
\]

\[
\forall B''' : \{B_3\}, \{B_4\} = \text{State}(B''', l_0, l_4)
\]

If \(\mathcal{O} = \Omega^o(l_c)\)

\[
\forall B_1 : \{B_1^o\}, \{B_1^e\} = \text{M}(B_1, l_c, l_4, \text{true, true})
\]

\[
\forall B_2 : \{B_2^o\}, \{B_2^e\} = \text{M}(B_2, l_c, l_4, \text{true, false})
\]

\[
\forall B_3 : \{B_3^o\}, \{B_3^e\} = \text{M}(B_3, l_c, l_4, \text{false, true})
\]

\[
\forall B_4 : \{B_4^o\}, \{B_4^e\} = \text{M}(B_4, l_c, l_4, \text{false, false})
\]

\[
B^c = B_1^o \cup B_2^o \cup B_3^o \cup B_4^o
\]

\[
B'^c = B_1^o \cup B_2^o \cup B_3^o \cup B_4^o
\]

If \(B''^c \not= \emptyset\)

\[
\text{add to the pool of results } o^c
\]

If \(B^c \not= \emptyset\)

\[
\text{add to the pool of results } o^e
\]

\[
\forall B^c : \text{Primary}(B^c, \Theta)
\]

\[
\forall B''^c : \text{Violation}(B''^c, [1])
\]

Procedure 1 (Lifeline): \(\{B\} = \text{Lifeline}(B, l)\)

The procedure Lifeline takes as input a process block \(B\), which structure is consistent with Definition 1, and a literal \(l\). The process block \(B\) can contain a task labeled \(x\). The input of such procedure is a set of process blocks \(\{B\}\) where each \(B\) is a sub-process block of \(B\), which means that \(\Sigma(B') \subseteq \Sigma(B)\).

Each \(B\) contains a task \(t_i\) labeled \(l_0\) such that \(l \in \text{ann}(t_i)\). Each serialization of \(B\) must contain \(t_i\) and none of the tasks following \(t_i\) in the serialization can be labeled \(x\).

\[
\forall \epsilon \in \Sigma(B'), t_i \in \epsilon \text{ and } \not\exists t_x \in \epsilon | t_x \text{ is labeled as } x \text{ and } x \geq i
\]

It can be noticed that, according to Lemma 1, independently from the task in \(B\) chosen to be labeled \(l_0\), it exists at least a serialization of \(B\) containing such task. Thus it always exists a sub-process block \(B\) containing the serializations of such task. In the worst case, which is when only one serialization of \(B\) contains such task, the sub-process block resulting consists of a sequence block containing the tasks belonging to the serialization in the same order.
However this does not mean that a sub-process block of $B$ containing the task labeled $life$ always exists if $B$ contains a task labeled $x$.

The last constraint of each $B'$ contained in the output set of the algorithm, is that they cannot allow serializations containing a task whose annotation contains $l$ and such task precedes the task labeled as $life$ in the serialization.

$$\forall \epsilon \in \Sigma(B') \text{ where } t_i \in \epsilon \text{ is labeled as } life, \neg \exists \in \epsilon \text{ and } j < i$$

**Procedure 2 (Starting State):** $\{B^i_{life}\}, \{B'^i_{life}\} = State(B_{life}, l)$

The procedure Starting State takes as input a process block $B_{life}$, containing a single task labeled $life$, and a literal $l$. Each serialization of $B_{life}$ has to contain the task labeled $life$.

The output of this procedure is a pair of sets of sub-process blocks of $B_{life}$, namely $\{B^i_{life}\}$ and $\{B'^i_{life}\}$. The union of the serializations of all $B_{life}$ and all $B_{life}$ has to contain the same serializations of $B_{life}$.

$$\forall B^i_{life}, B'^i_{life} \forall \epsilon \in \Sigma(B^i_{life}) \text{ or } \epsilon \in \Sigma(B'^i_{life}), \exists \epsilon' \in \Sigma(B_{life}) \epsilon = \epsilon'$$

Each $B^i_{life}$ ensures that each of its possible traces execute the task labeled $life$ and the resulting state contains $l$.

$$\forall \theta \in \Theta(B^i_{life}, ann), \exists (t, L) \in \theta | t \text{ is labeled } life \text{ and } l \in L$$

Each $B'^i_{life}$ ensures that each of its possible traces execute the task labeled $life$ and the resulting state contains $l$.

$$\forall \theta \in \Theta(B'^i_{life}, ann), \exists (t, L) \in \theta | t \text{ is labeled } life \text{ and } l \notin L$$

**Procedure 3 (Achievement):** $\{B^o\}, \{B'^o\} = A(B_{life}, l, l_d, b_1, b_2)$

The procedure Achievement takes as input a process block $B_{life}$ which contains a single task labeled $life$, two literals $l$, $l_d$ and two booleans $b_1, b_2$. Each trace of $B_{life}$ has to execute the task labeled $life$ and the boolean $b_1$ describes whether $l$ holds in the process state after the task labeled life has been executed. The same is described by $b_2$ regarding $l_d$.

The output of this procedure consists of a pair of sets of sub-process blocks $\{B^o\}, \{B'^o\}$. Each possible serialization of $B_{life}$ has to be contained either in one of the sub-process block belonging to $B^o$ or belonging to $B'^o$. As additional constraint, if a serialization is possible for an element of one of the two sets, then it cannot be a possible serialization of an element belonging to the other set.

$$\forall B^o, B'^o \forall \epsilon \in \Sigma(B^o), \neg \exists \epsilon' \in \Sigma(B'^o) | \epsilon = \epsilon'$$

Each element of $\{B^o\}$ allows only traces containing at least a state where $l$ holds and occurs after the state holding after the execution of the task labeled $life$. Additionally such traces must not contain a state where $l_d$ holds between it and the state holding after the execution of the task labeled $life$.

$$\forall B^o, \forall \theta \in \Theta(B^o, ann), \exists (t_i, L_i) \in \theta | t_i \text{ is labeled } life, \exists (t_j, L_j) \in \theta | t_j \text{ is labeled } life, \exists (t, L) \in \theta \text{ and } j > i, \neg \exists (t_i, L_i) \in \theta | l \in L_k \text{ and } i < k < j$$

Each element of $\{B^o\}$ has a task labeled $x$ and such task must belong to all its serializations. For each possible trace of each element of $\{B^o\}$, the task labeled $x$ is never executed before the task labeled $life$, the state holding after the execution of the task labeled $x$ has to contain $l$, and there must not be another state between that and the one holding after the execution of the task labeled $life$ where $l_c$ holds.

$$\forall B^o, \forall \theta \in \Theta(B^o, ann), \exists (t_i, L_i) \in \theta | t_i \text{ is labeled } life, \exists (t_j, L_j) \in \theta | t_j \text{ is labeled } x \text{ and } j > i, \neg \exists (t_k, L_k) \in \theta | l_c \text{ in } L_k \text{ and } i < k < j$$

Each element of $\{B^o\}$ allows only traces containing a state where $l$ holds and occurs after the state holding after the execution of the task labeled $life$, unless such traces contain a state where $l_d$ holds between it and the state holding after the execution of the task labeled $life$.

$$\forall B^o, \forall \theta \in \Theta(B^o, ann), \exists (t_i, L_i) \in \theta | t_i \text{ is labeled } life, \neg \exists (t_j, L_j) \in \theta | c \in L_j \text{ and } j > i, \exists (t_k, L_k) \in \theta | l_d \in L_k \text{ and } i < k < j$$

Each element of $\{B^o\}$ has a task labeled $v$ and such task must belong to all its serializations. For each possible trace of each element of $\{B^o\}$, the task labeled $v$ is never executed before the task labeled $life$, the state holding after the execution of the task labeled $v$ has to either contain $l$ or be the last task in all serializations of $B^o$, and in both cases there must not be another state between that and the one holding after the execution of the task labeled $life$ where $l_d$ holds.

$$\forall B^o, \forall \theta \in \Theta(B^o, ann), \exists (t_i, L_i) \in \theta | t_i \text{ is labeled } life, \exists (t_j, L_j) \in \theta | t_j \text{ is labeled } v \text{ and } (either \exists (t, L) \in L \text{ and } \neg \exists (t_i, L_i) \in \theta | k > j \text{ and } j > i, \neg \exists (t_k, L_k) \in \theta | l_d \text{ and } i < k < j)$$

Each element of $\{B^o\}$ allows only traces containing states following the one which holds after the execution of the task labeled $life$ where $l_c$ holds. These states have to contain $l_c$ until a state contains $l_d$ or for the whole trace.

$$\forall B^o, \forall \theta \in \Theta(B^o, ann), \exists (t_i, L_i) \in \theta | t_i \text{ is labeled } life, \exists (t_j, L_j) \in \theta | l \in L_j \text{ and } j > i, \forall (L) \in \theta | k < k \text{ and } l_c \in L_k \text{ or } (L) \in \theta | k < k \text{ and } l_c \in L_k)$$

Each element of $\{B^o\}$ has a task labeled $x$ and such task must belong to all its serializations. For each possible trace of each element of $\{B^o\}$, the task labeled $x$ is never executed before the task labeled $life$, the state holding after the execution of the task labeled $x$ has to either contain $l$ or be the last task in all serializations of $B^o$, and in both cases there must not be another state between that and the one holding after the execution of the task labeled $life$ where $l_d$ holds.

$$\forall B^o, \forall \theta \in \Theta(B^o, ann), \exists (t_i, L_i) \in \theta | t_i \text{ is labeled } life, \exists (t_j, L_j) \in \theta | t_j \text{ is labeled } x \text{ and } (either \exists (L) \in L \text{ and } \neg \exists (t_i, L_i) \in \theta | h > j \text{ and } j > i, \neg \exists (t_k, L_k) \in \theta | l_d \text{ and } i < k < j)$$

Each element of $\{B^o\}$ allows only traces which contains at least a state where $l_c$ does not hold and follows the state
holding after the execution of the task labeled \textit{life}. Moreover each of those traces must not contain a state where \( l_c \) holds between the one \( l_c \) does not hold and the one holding after the execution of the task labeled \textit{life}.

\[ \forall B^c, \forall \theta \in \Theta(B^c, \text{ann}), \exists (t_i, L_i) \in \theta | t_i \text{ is labeled life, } \exists (t_j, L_j) \in \theta | | c \notin L_j \text{ and } j > i, \neg \exists (t_k, L_k) \in \theta | | d \in L_k \text{ and } i < k < j. \]

Each element of \( \{ B^v \} \) has a task labeled \( v \) and such task must belong to all its serializations. For each possible trace of each element of \( \{ B^v \} \), the task labeled \( v \) is never executed before the task labeled \textit{life}, the state holding after the execution of the task labeled \( v \) must not contain \( l_c \) and there must not be another state between that and the one holding after the execution of the task labeled \textit{life} where \( l_c \) does not hold.

\[ \forall B^v, \forall \theta \in \Theta(B^v, \text{ann}), \exists (t_i, L_i) \in \theta | t_i \text{ is labeled life, } \exists (t_j, L_j) \in \theta | t_j \text{ is labeled } v, l_c \notin L_j \text{ and } j > i, \neg \exists (t_k, L_k) \in \theta | l_d \notin L_k \text{ and } i < k < j. \]

Algorithm 2 (Violation): \( \text{pool} = V(B_v, \hat{\gamma}[i]) \)

The algorithm \textit{Violation} takes as input a process block \( B_v \) and a secondary norm \( \Omega_i \). The output of the algorithm is a set of elements, where each of the elements describe whether a subset of the possible traces in \( P \) are compliant with \( \Omega_i \).

\[
\text{if } \hat{\gamma}[i] = \text{null } \\
\text{add to the pool of results } c^- \\
\text{else} \\
\{ B'_v \} = V \text{life}(B_c) \\
\forall B': \{ B'^v \}, \{ B''^v \} = \\
\text{State}(B', \text{liv}, l_c) \\
\forall B': \{ B_1 \}, \{ B_2 \} = \\
\text{State}(B''_v, \text{liv}, l_d) \\
\forall B''_v: \{ B_3 \}, \{ B_4 \} = \\
\text{if } O = O^\gamma(l_c) \\
\forall B_1: \{ B'_1 \}, \{ B_1 \} = \\
A(B_1, l_c, l_d, \text{true, true}) \\
\forall B_2: \{ B_2 \}, \{ B_2' \} = \\
A(B_2, l_c, l_d, \text{true, false}) \\
\forall B_3: \{ B_3 \}, \{ B_3' \} = \\
A(B_3, l_c, l_d, \text{false, true}) \\
\forall B_4: \{ B_4 \}, \{ B_4' \} = \\
A(B_4, l_c, l_d, \text{false, false}) \\
\text{else} \\
\forall B_1: \{ B'_1 \}, \{ B_1 \} = \\
M(B_1, l_c, l_d, \text{true, true}) \\
\forall B_2: \{ B_2 \}, \{ B_2' \} = \\
M(B_2, l_c, l_d, \text{true, false}) \\
\forall B_3: \{ B_3 \}, \{ B_3' \} = \\
M(B_3, l_c, l_d, \text{false, true}) \\
\forall B_4: \{ B_4 \}, \{ B_4' \} = \\
M(B_4, l_c, l_d, \text{false, false}) \\
B^c = B_1 \cup B_2 \cup B_3 \cup B_4 \\
B^v = B_1' \cup B_2' \cup B_3' \cup B_4' \\
\text{if } B^c \neq \emptyset \\
\text{add to the pool of results } c^+ \\
\forall B^v: \text{Primary}(B^v, \emptyset) \]

Procedure 5 (Violation Lifeline): \( \{ B_{life} \} = V \text{life}(B_v) \)

The procedure \textit{Violation Lifeline} takes as input a process block \( B_v \) where one of its tasks is labeled \( v \).

The output of the procedure is a set of process blocks \( \{ B_{life} \} \) where each of its element is a sub-process blocks of \( B_v \). Each element of \( \{ B_{life} \} \) has a task labeled as \textit{life} and for each of its possible serializations, the task labeled \textit{life} is serialized immediately before the task labeled as \( v \).

\[ \forall B_{life}, \forall \epsilon \in \Sigma(B_{life}), \exists t_i \in \epsilon | t_i \text{ is labeled } v \text{ and } \exists t_j \in \epsilon | t_j \text{ is labeled life and } j = i - 1. \]

The label \( v \) is removed from each element of \( \{ B_{life} \} \).

Algorithm 3 (Compliance): \( \text{Compliance(pool)} \)

The algorithm \textit{Compliance} takes as input a set of compliance results \( \text{pool} \). Each element of \( \text{pool} \) represent wether a subset of the traces belonging to the structured process are compliant with the primary norm: \( o^+ \), not compliant with the primary norm: \( o^- \), compliant with one of the compensations in the chain \( c^" \) or no compliant with each compensation in the chain \( c^- \).

The algorithm returns a compliance result for the structured process with respect to the norm given as input to the framework as follows:

\[
\text{if } \forall e \in \text{pool}, e = o^+ \\
(P, \text{ann}) \vdash_{\text{EN}} \emptyset \otimes \gamma \\
\text{else if } \neg \exists e \in \text{pool} | e = o^+ \\
\text{if } \neg \exists e \in \text{pool} | e = c^- \\
(P, \text{ann}) \vdash_{\text{EN}} \emptyset \otimes \gamma \\
\text{else if } \neg \exists e \in \text{pool} | e = c^+ \\
(P, \text{ann}) \not\vdash_{\text{EN}} \emptyset \otimes \gamma \\
\text{else} \\
(P, \text{ann}) \vdash_{\text{EN}} \emptyset \otimes \gamma \\
\]

A. Complexity

Because a structured process can represent an exponential number of traces, our abstract framework avoids to analyze the the problem trace by trace, in other words using a \textit{brute force} approach. Instead the abstract framework tries to decide whether a structured is compliant with a regulation by analyzing the topology of the structured process directly, avoiding in this way to generate all the possible traces.

However we do not claim that the abstract framework always avoids to deal with the single traces, in fact it can be the case that when a sub-process block is generated by one of the procedures used, such sub-process block allows exactly one serialization, hence a single trace. If we consider the worst case, where the abstract framework is not able to
group together different serializations into a single sub-process block, then the result is that the abstract framework is applying a brute force approach to solve the problem.

Additionally, the detailed way in which Algorithms 2 and 3 are defined, allows to notice the existence of various branching points inside the abstract framework. Namely those branching points occur when different traces need a different type of analysis.

One of the branching points of the framework is identified by the procedure State, which splits a set of traces in two sets depending in a given literal holds in a given state of the traces. This procedure is used to determine whether the fulfillment condition and the deadline condition hold before at the beginning of the activation period of a norm. Because we provide just an interface for the procedure to verify whether an obligation is fulfilled, these details may seem superfluous. However by pointing out these differences, we can define the procedure in a similar way as the ones described in [20] which have the advantage to be computable in polynomial time.

IV. Conclusion and Related Work

Business process compliance received increased attention in the field of business process modeling in the past few years. The majority of approaches propose some logics for compliance (e.g., deontic logic [5], linear temporal logic [21], clause based logic/logic programming [8], [17], extensions of BPMN languages [22]). However, to the best of our knowledge, this is a first systematic investigation on the complexity of business process compliance. [19] provides a linear time algorithm to check whether a single trace is compliant, and [23] gives approximate solutions in linear time. [13] shows that the problem of checking whether a process is compliant or not is computationally infeasible.

Linear Temporal Logic and model checking are very powerful techniques for the verification of different type of systems, and it can be used for the verification of business processes and some aspects of compliance [21]. However, the complexity of linear temporal logic is NP-complete for the language including the F (sometimes in the future) operator and PSPACE-complete for the extensions with F, X (next), U(until) operators. In addition, while it is tempting to represent the deontic operators (obligations) in temporal logic, temporal logic is not fully appropriate for the task [24], since it is not able to capture in a natural way the different nuances required by normative reasoning, for example, how representation on norms that can be violated but for which compensation are possible.

This paper is a first approach to propose an abstract framework to verify whether a structured process is compliant with a regulation. We do not argue about the complexity of the solution proposed, apart from a brief discussion at the end of Section 3, however the problem tackled in this paper intersects the one which is proven to be NP-complete [13] and extends it with the concept of compensations.

As a matter of fact, as future work we plan to extend the current framework, allowing it to decide whether a structured process is compliant with a set of regulations instead of a single one as discussed in the present paper.

In addition the abstract framework proposed in the present paper can be used to compare existing solutions for compliance checking problems.

ACKNOWLEDGMENT

• NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

• Silvano Colombo Tosatto is supported by the National Research Fund, Luxembourg.

REFERENCES


