An Iterative Approach for Business Process Template Synthesis from Compliance Rules

Ahmed Awad[1], Rajeev Goré[2], James Thomson[2], and Matthias Weidlich[1]

[1] Hasso Plattner Institute, University of Potsdam, Germany
{ahmed.awad,matthias.weidlich}@hpi.uni-potsdam.de
[2] School of Computer Science, The Australian National University, Australia
{Rajeev.Gore,jimmy.thomson}@anu.edu.au

Abstract. Companies have to adhere to compliance requirements. Typically, both, business experts and compliance experts, are involved in compliance analysis of business operations. Hence, these experts need a common understanding of the business processes for effective compliance management. In this paper, we argue that process templates generated out of compliance requirements can be used as a basis for negotiation among business and compliance experts. We introduce a semi automated approach to synthesize process templates out of compliance requirements expressed in Linear Temporal Logic (LTL). As part of that, we show how general constraints related to business process execution are incorporated. Building upon existing work on process mining algorithms, our approach to synthesize process templates considers not only control-flow, but also data-flow dependencies. Finally, we elaborate on the application of the derived process templates and present an implementation of our approach.

Keywords: Process synthesis, Analysis of business process compliance specification, Process mining.

1 Introduction

Recently, there has been a growing interest in compliance checking of business operations. Financial scandals in large companies led to legislative initiatives, such as SOX [1]. The purpose of these initiatives is to enforce controls on the business operations. Such controls relate to the execution order of business activities, the absence of activity execution in a dedicated data context, or restrictions on role resolution to realize separation of duty.

Driven by these trends, numerous approaches have been presented to address compliance management of business processes. In general, we can distinguish two types of approaches. First, compliance rules can guide the design of a business process [12,13]. These approaches ensure *compliance by design* by identifying compliance violations in the course of process model creation. Second, existing process models are *verified* against compliance rules [10,6]. Given compliance requirements and a process model as input, these approaches identify violations on the process model level.

Evidently, addressing compliance during the design of business operations has many advantages. Non-compliant processing is prevented at an early stage of process implementation and costly post-implementation compliance verification along with root

H. Mouratidis and C. Rolland (Eds.): CAiSE 2011, LNCS 6741, pp. 406–421, 2011.

cause analysis of non-compliance is not needed. In most cases, process models that are synthesized from compliance rules cannot be directly used for implementing a business process. Instead, they should be seen as a blueprint that is used as a basis for negotiation between business and compliance experts. Hence, we refer to these process models as *process templates* in order to emphasize that further refinements are needed to actually implement the business process. While this approach has been advocated by other authors, e.g., [12,11,25], existing approaches are limited when it comes to data-dependent compliance requirements.

In this paper, we present an approach to the synthesis of compliant process templates that avoids some of the pitfalls of existing approaches. We start with a set of compliance rules specified in LTL. Hence, we do not require the definition of explicit points in time as in [12,11], but focus on relative execution order dependencies. Further, we also consider data flow dependencies between activity executions, which is neglected in [25]. These rules are then enriched with general constraints related to business process execution to avoid phenomena such as vacuous satisfiability. Subsequently, a process template is generated automatically if the compliance requirements are satisfiable. We also illustrate how generated templates are applied during process design and how the template generation may identify inconsistencies and open questions. Hence, the template guides further refinements of the process model and the compliance requirements. To evaluate the applicability of our approach, we present a prototypical implementation. Our contribution is a complete approach to process design grounded in compliance rules.

Against this background, the remainder of this paper is structured as follows. The next section introduces preliminaries for our work, such as the applied formalism. Section 3 introduces our approach of synthesizing process templates from a given set of compliance rules. We also elaborate on how to use these templates as a basis for process design. A prototypical implementation of our approach is presented in Section 4. Finally, we discuss related work in Section 5 and conclude in Section 6.

2 Preliminaries

This section gives preliminaries for our work. Section 2.1 clarifies our notion of execution semantics. Section 2.2 presents LTL as the logic used in this paper. Section 2.3 summarizes existing work on generating a behavioral model from a given LTL formula.

2.1 Process Runs as Linear Sequences

In this paper, we rely on trace semantics for process models. An execution sequence σ of a process model is referred to as a *process run* or *trace* – a finite linear sequence of states $\sigma : s_0, s_1, \ldots, s_n$ with a start state s_0 and an end state s_n. Evidently, a process model as well as a set of compliance requirements allow for many conforming traces. Each state of a trace is labeled with propositions that refer to *actions* and *results*. Actions are the driving force of a trace and refer to the execution of business activities. This, in turn, may effect or be constrained by results, which relate to data values of the business process. As an example, think of an activity 'risk analysis' (ra) and a data object 'risk'. The action that represents the execution of this activity may have the result of setting the state of the data object to 'high' or 'low'. The execution of another activity, i.e.,

another action, may be allowed to happen solely if a certain result, e.g., the object has been set to 'high', occurred. Both, actions and results, are represented by Boolean propositions at each state. For instance, proposition ra being 'true' at a state s_i means that the action, i.e., execution of activity 'risk analysis', has happened at state s_i. In contrast, proposition ra being 'false' at state s_i means that the action did not happen at state s_i. Given a trace $\sigma : s_0, s_1, \ldots, s_n$, we write $p \in s_i$ to indicate that proposition p is true in state s_i, for $0 \leq i \leq n$ and $p \in \sigma$ if there is a state s_i in σ where $p \in s_i$, for some $0 \leq i \leq n$.

We represent an execution sequence as a linear sequence of states where states are labelled with both actions and results, and (unlabelled) edges between states represent the temporal ordering in the sequence. Hence, we rely on Linear Temporal Logic (LTL) in order to formulate statements about traces.

2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) [20] is a logic specifically designed for expressing and reasoning about properties of linear sequences of states. The formulae of LTL are built from atomic propositions using the connectives of $\vee$ (or), $\wedge$ (and), $\neg$ (not) and $\Rightarrow$ (implication), and the following temporal connectives: $\mathbf{X}$ (next), $\mathbf{F}$ (eventually), $\mathbf{G}$ (always), $\mathbf{U}$ (until) and $\mathbf{B}$ (before). The latter are interpreted as follows:

$\mathbf{X}\ \varphi$**:** in the neXt state, φ holds
$\mathbf{F}\ \varphi$**:** there is some state either now or in the Future where φ holds
$\mathbf{G}\ \varphi$**:** in every state Globally from now on, φ holds
$\varphi\ \mathbf{U}\ \psi$**:** there is some state, either now or in the future, where ψ holds, and φ holds in every state from now Until that state
$\varphi\ \mathbf{B}\ \psi$**:** Before ψ holds, if it ever does, φ must hold.

We apply LTL to encode compliance requirements. Hence, we obtain a set of formulae Γ expressing the constraints to which compliant traces have to conform.

2.3 Finding All LTL-Models of a Given LTL Formula

Given a collection of compliance requirements expressed as a set Γ of LTL-formulae, we seek to find a behavioral model that captures all *formula-models*, i.e., traces in our setting, which satisfy Γ. That is, such a model describes all linear sequences of states $s_0, s_1, \ldots, s_n$ such that Γ is true at s_0. Since Γ may contain *eventualities*, such as $\mathbf{X}\ \varphi$ or $\psi_1\ \mathbf{U}\ \psi_2$, ensuring that Γ is true at s_0 may require us to ensure that φ is true at s_1 or ψ_2 is true eventually at some state s_i with $0 \leq i \leq n$. In contrast to model checking [7] we are not given a single trace, but construct all traces satisfying the given constraints.

The first step is to determine whether the constraints are satisfiable. If not, the specification is erroneous since no trace can conform to the given constraints. The second step is the creation of the behavioral model that describes *all* traces.

For both steps, we use a tableaux-based method introduced in [24,23]. In essence, this approach works as follows. We start by creating a root node containing Γ and proceed in two phases. First, a finite (cyclic) graph of tableau nodes is created by applying tableau-expansion rules that capture the semantics of LTL and by pruning nodes containing local contradictions [24]. Second, once the graph is complete a reachability

algorithm is used to determine which nodes do not satisfy their eventualities. These nodes are removed and the reachability algorithm is reapplied until no nodes may be removed. The set of formulae Γ is satisfiable, if and only if the root node has not been removed [24]. Further, the graph created by the tableau algorithm, referred to as the *pseudomodel*, describes all possible formula-models, i.e., possible traces [24]. We use this pseudomodel to extract possible traces during our synthesis approach.

3 Synthesis of Process Templates from Compliance Rules

In this section, we describe our approach to the synthesis of process models from a set of compliance rules expressed in temporal logic. First, Section 3.1 gives an overview of the approach and introduces an example set of compliance rules used to illustrate all subsequent steps. Section 3.2 describes the LTL encoding of the compliance rules and additional domain knowledge. Section 3.3 elaborates on the extraction of traces from a behavioral model, while Section 3.4 focuses on consistency of these traces. Synthesis of a process template from these traces is discussed in Section 3.5. Finally, we elaborate on the evaluation of synthesized templates in Section 3.6.

3.1 Overview

The process model in Fig. 1 visualizes the steps to synthesize a process template out of a set of compliance rules. First, a set of compliance rules is collected. In order to identify whether these requirements are consistent and thus a process template can be synthesized, related domain-specific knowledge is identified. In Section 3.2 we give details on the LTL encoding of both compliance rules and domain knowledge.

For the conjunction of these LTL formulae, we verify satisfiability as it has been summarized in Section 2.3. If the set is not satisfiable then no trace can be constructed to satisfy the given LTL formulae so the inconsistency is reported to the user. If the set is satisfiable then the satisfiability checker automatically returns the pseudomodel which is a behavioral model of all traces that obey the given constraints.

As a next step, finite traces are extracted from the pseudomodel by following all choice points and stopping when a trace becomes cyclic. We focus on this step in Section 3.3. Having a finite set of traces that satisfy the compliance rules, we check it for

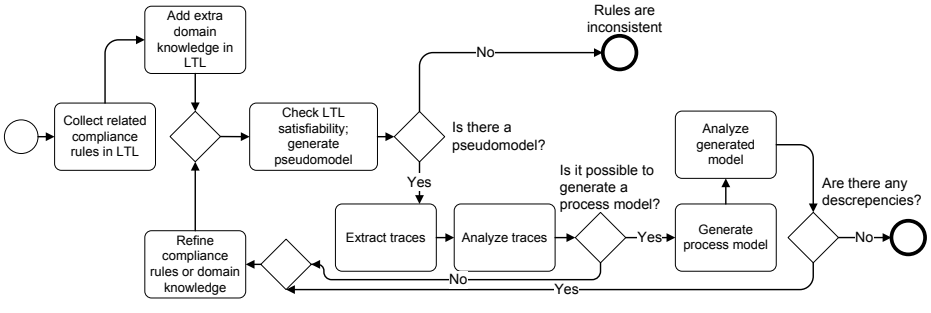

Fig. 1. Process Synthesis Approach

consistency. This check guarantees that a template can be generated. Inconsistent traces hint at issues in the specification, so that a new iteration of the synthesis may be started with refined compliance rules or adapted domain knowledge. We focus on the analysis of traces in Section 3.4. If the traces are consistent, we apply a process synthesis algorithm to extract a process template. Details on this step are given in Section 3.5. The synthesized template is then analyzed to identify discrepancies that stem, e.g., from underspecification. Depending on the result of this analysis, again, a new iteration of the synthesis may be started. We discuss the evaluation of process templates in Section 3.6.

Example. We illustrate our approach with an example from the financial domain. Anti money laundering guidelines [8] address financial institutes, e.g., banks, and define a set of checks to prevent money transfers with the purpose of financing criminal actions. We focus on the following guidelines for opening new bank accounts:

$R1$**:** A risk assessment has to be conducted for each 'open account' request.
$R2$**:** A due diligence evaluation has to be conducted for each 'open account' request.
$R3$**:** Before opening an account the risk associated with that account must be low. Otherwise, the account is not opened.
$R4$**:** If due diligence evaluation fails, the client has to be added to the bank's black list.

3.2 LTL Encoding

Once the compliance rules have been collected, a behavioral model that represents all traces conforming to these rules is created. In order to arrive at such a model, we need to collect extra domain-specific rules. Much of the domain-specific rules can be generated automatically from a higher level description. Such a description needs to be defined by a human expert in the first place and comprises the following information.

Actions and Goals. The set of all *actions* is denoted by A. The set of *goal* actions $G \subset A$ comprises activities that indicate the completion of a trace. Moreover, we capture contradicting actions that are not allowed to occur together in one trace in a relation $CA : A \times 2^A$.

Results and Initial Values. The set of all *results* is denoted by R. The initial values of data objects are defined by a set $IV \subset R$. Further, we define the set of negated results as $\overline{R} = \{\neg r | r \in R\}$. Similar to contradicting actions, we capture contradicting results in a relation $CR : R \times 2^R$.

Relation between Actions and Results. The mapping from actions to sets of results is given as a relation $AM : A \times 2^{R \cup \overline{R}}$. Mutually exclusive sets of results are captured in a relation $RE = \{S : \exists a \in A.(a, S) \in AM \wedge S \neq \emptyset\}$.

Based on this information and two additional actions $start$ and end that represent the initial and final states of a trace (independent of any goal states), we derive LTL rules to represent the domain knowledge according to Table 1. Common process description languages, e.g., BPMN or EPCs, assume interleaving semantics, which is enforced by formula $interleaving$ and $progress$. The information on exclusiveness constraints and on contradicting actions and results yields the formulae $mutex$ and $contra$. The formula $causality$ guarantees correct implementation of dependencies between actions and results. Finally, the formulae $once$, $final$, $goals$, and $initial$ ensure

Table 1. The formulae making up the domain knowledge

Constraint Description	Formalization
To realize interleaving semantics, the formula $interleave$ ensures that at most one action can be true, i.e., one activity can be executed, at any state.	$interleave(a) = a \Rightarrow (\bigwedge_{b \in A \setminus \{a\}} \neg b))$ $interleave = \bigwedge_{a \in A} interleave(a)$
The formula $progress$ guarantees that at least one action occurs at each state.	$progress = \bigvee_{a \in A} a$
The mutual exclusion constraints given in RE are enforced by the formula $mutex$, i.e., exclusive results cannot be true at the same time.	$mutex(S) = \bigwedge_{a,b \in S,\ a \neq b} \neg(a \wedge b)$ $mutex = \bigwedge_{S \in RE} mutex(S)$
Knowledge on contradicting actions or results is taken into account by the formulae, con and $conRes$.	$con(a) = a \Rightarrow \mathbf{G} \bigwedge_{b \in CA(a)} \neg b$ $conRes(r) = a \Rightarrow \mathbf{G} \bigwedge_{s \in CR(r)} \neg s$ $contra = \bigwedge_{a \in A \cup R} con(a) \wedge conRes(a)$
To implement the relation between actions and results, formula cau_1 states that for every entry $(a, S) \in AM$ the action a must cause at least one of the results in S. Formula cau_2 states that for every result r, that result can only be changed by one of the actions which can cause it.	$cau_1(a, S) = a \Rightarrow \bigvee_{r \in S}\ r$ $cau_2(r) =$ $r \Rightarrow (\mathbf{X} \bigvee_{(a,S) \in AM,\ \{r, \neg r\} \cap S \neq \emptyset} a)\ \mathbf{B}\ \neg r$ $causality =$ $\bigwedge_{(a,S) \in AM} cau_1(a, S) \wedge \bigwedge_{r \in R \cup \overline{R}} cau_2(r)$
The formula $once$ enforces that all actions other than end occur at most once, in order to avoid infinite behavior. The formula $final$ enforces that end persists forever to represent the process end.	$once(a) = a \Rightarrow \mathbf{X}\ \mathbf{G}\ \neg a$ $once = \bigwedge_{a \in A \setminus \{end\}} once(a)$ $final = end \Rightarrow \mathbf{G}\ end$
The formula $goals$ is used to require that eventually the outcome of the process is determined, while $inital$ ensures correct initial values for all objects.	$goals = \bigvee_{g \in G} g$ $initial = start \Rightarrow \bigwedge_{v \in IV} v$

correct initialization and successful termination of any trace. The combination of all these formulae yields the formula $domain$, which represents the domain knowledge.

$$domain = start \wedge \mathbf{G}\ initial \wedge \mathbf{F}\ goals \wedge \mathbf{F}\ end \wedge \mathbf{G}\ interleave \wedge \mathbf{G}\ progress \wedge \mathbf{G}\ mutex \wedge \mathbf{G}\ causality \wedge \mathbf{G}\ once \wedge \mathbf{G}\ contra \wedge \mathbf{G}\ final$$

Example. For our example, an expert first identifies the following actions and results.

$Actions = \{ra, edd, og, od, bl\}$

ra: conduct a risk assessment
edd: evaluate due-diligence
og: grant a request to open an account
od: deny a request to open an account
bl: blacklist a client.

$Results = \{ri, rh, rl, ei, ef, ep\}$

ri: risk assessment is initial
rh: risk was assessed as high
rl: risk was assessed as low
ei: due-diligence evaluation is initial
ef: due-diligence evaluation failed
ep: due-diligence evaluation passed.

Note that the results are all descriptive statements, while the actions refer to activities. Moreover, we introduce *positive* representations for the states 'high' and 'low' of the risk object, even though both states are opposed. That is due to the three possible states of the risk object: high, low, or initial. The same holds true for the the due-diligence object.

Based on these actions and results, the compliance rules are encoded in LTL. As a process to open a bank account is considered, the process is assumed to start by receiving such a request. Therefore, rules 1 and 2 are interpreted as "A risk assessment has to be conducted" and "A due diligence evaluation has to be conducted", respectively. The third rule is interpreted to mean that the risk associated with opening an account must be low *at the time the request is granted*, rather than at some point in the past. Similarly is the case when denying the open request, the risk has to be high.

$R1$**:** A risk assessment has to be conducted.
$\mathbf{F}\ ra$ "Eventually ra must hold"

$R2$**:** A due diligence evaluation has to be conducted.
$\mathbf{F}\ edd$ "Eventually edd must hold"

$R3$**:** The risk associated with opening an account must be low when the request is granted.
$\mathbf{G}\ (og \Rightarrow rl) \wedge \mathbf{G}\ (od \Rightarrow rh)$ "Always, og only if rl, and always, od only if rh"

$R4$**:** If due diligence evaluation fails, the client has to be added to the bank's black list.
$\mathbf{G}\ (edd \wedge ef \Rightarrow \mathbf{F}\ bl)$ "Always, edd and ef imply eventually bl"

As a next step, the domain knowledge is defined in more detail. For instance, the action mapping defines $ra \mapsto \{rh, rl\}$ and $ra \mapsto \{\neg ri\}$. The former says that action ra causes the risk object to take a concrete value of 'high' or 'low'. The latter means that ra causes the risk to stop being 'initial' by forcing ri to not hold. Excluding results are defined, e.g., $\{ri, rl, rh\}$ states that at most one of the propositions ri, rh, rl can hold at a time. The goal of the process is defined as $\{og, od\}$ and the set of initial values $\{ri, ei\}$ signifies that initially, both risk and due-diligence objects, are put to an initial, unknown, value. There are also contradicting actions, $\{og \mapsto \{od\}, od \mapsto \{og\}\}$, ensuring that we cannot grant and deny a request within the same trace. Based on Table 1, this specification is converted into LTL. For example, this yields the formula $progress = ra \vee edd \vee og \vee od \vee bl \vee start \vee end$. The final set of LTL formulae is the union of the $domain$ formula with all four formulae representing the compliance rules.

3.3 Extracting Traces

Given a set of LTL formulae, we apply the technique summarized in Section 2.3 to determine whether the constraints are satisfiable. If so, we obtain a pseudomodel that describes all traces that conform to the set of formulae. To create a process template, these traces are extracted. Any sequence $\sigma = s_0, \ldots, s_n$ of states, starting at the root node of the pseudomodel can be extended into a trace. As we are modeling finite sequences with an end state, we consider a trace to be complete if $end \in s_n$. Because of the $once$ constraint introduced in the previous section, there will be no loops in the pseudomodel between the start and the end. Hence, the finite set of paths in the pseudomodel between the root state and a state labeled with end is the set of correct traces.

Table 2. Excerpt of the extracted traces

$\sigma_1 : start \wedge ei \wedge ri, edd \wedge ep \wedge ri, ra \wedge ep \wedge rh, bl \wedge ep \wedge rh, od \wedge ep \wedge rh, end \wedge ep \wedge rh$
$\sigma_2 : start \wedge ei \wedge ri, edd \wedge ep \wedge ri, ra \wedge ep \wedge rh, od \wedge ep \wedge rh, end \wedge ep \wedge rh$
$\dots$
$\sigma_{37} : start \wedge ei \wedge ri, bl \wedge ei \wedge ri, edd \wedge ep \wedge ri, ra \wedge ep \wedge rh, od \wedge ep \wedge rh, end \wedge ep \wedge rh$
$\dots$
$\sigma_{42} : start \wedge ei \wedge ri, bl \wedge ei \wedge ri, ra \wedge rl \wedge ei, og \wedge rl \wedge ei, edd \wedge ep \wedge rl, end \wedge ep \wedge rl$

Note that it is possible to extract traces that take repetition of activities into account by omitting the *once* constraint in the domain knowledge. Still, for our purpose, this does not seem to be appropriate. Compliance rules rarely forbid the repetition of activity execution, so that modeling all potential loops blurs up the structure of a generated process template. As this hinders discussions between business and compliance experts, we neglect potential repetition for our synthesis approach.

Example. Some of the traces extracted from the pseudomodel of our running example are illustrated in Table 2. Here, the states of a trace are characterized by the conjunction of propositions that hold true in the respective state.

3.4 Analysis of Extracted Traces

As stated earlier, the goal of synthesizing a process template out of compliance rules is to support experts in getting a better understanding of the compliance aspects and to discover missing or under-specified requirements. However, it is possible to detect such under-specification by analysis of extracted traces before proceeding to synthesizing a process template. Yet, not every semantical error in the specification can be detected, so that a human expert has to validate the synthesized process template. We address the issue of under-specified LTL specification by correctness criteria for the extracted traces.

Let $\mathcal{P}$ be a set of traces derived from a pseudomodel, cf., Section 3.3. We leverage the information whether an action $a \in A$ is optional for completing the process.

Definition 1 (Optional Actions). *Given a set of actions A and a set of traces $\mathcal{P}$, the set A_O of optional actions is defined as $A_O = \{a \in A : \exists\, \sigma \in \mathcal{P}.a \notin \sigma\}$.*

We argue that correctness of a specification where some activity is optional implies the existence of a specific data condition under which the optional activity is executed. For the traces in Table 2, for instance, *og* and *od* are optional activities. The condition under which *og* executes is $(rl \wedge ef) \vee (rl \wedge ep) \vee (rl \wedge ei)$, i.e., the risk object assumes the state 'low'. Action *og* is executed independently from the state of the due diligence evaluation object. For action *od* the condition is $(rh \wedge ef) \vee (rh \wedge ep) \vee (rh \wedge ei)$, i.e., the risk is 'high'. In contrast, action *bl* is executed under the condition $(ei \wedge ri) \vee (ei \wedge rh) \vee (ei \wedge rl) \vee (ef \wedge ri) \vee (ef \wedge rh) \vee (ef \wedge rl) \vee (ep \wedge rh) \vee (ep \wedge rl) \vee (ep \wedge ri)$. Hence, none of the objects influences the decision of executing *bl*, since *bl* appears with *all* combinations of data values. Yet, *bl* is optional. This indicates an under-specified LTL specification as conditions for executing optional activities are not stated explicitly.

Definition 2 (Optional Action Execution Condition). *Let A_O be the set of optional actions, $\mathcal{P}$ a set of traces, and RE the set of mutually exclusive results. For an action $a \in A_O$, the execution condition is defined as:*
$cond_a = \{\{x_1, \ldots, x_n\} : \exists\, \sigma \in \mathcal{P}.\exists\, s \in \sigma.a \in s \wedge x_1 \in s \wedge x_1 \in S_1 \wedge S_1 \in RE \wedge \cdots \wedge x_n \in s \wedge x_n \in S_n \wedge S_n \in RE \wedge n = |RE|\}$.

This definition describes the conditions under which an action executes by investigating for each observation of the action a the data effects that are true in the same state as a. If an optional activity a has an execution condition, which is a proper subset of the combination of non-exclusive results, then this indicates a well specified set of compliance rules. We formalize this trace correctness criterion as follows.

Definition 3 (Proper Execution of Optional Actions). *Let A_O be the set of optional actions with respect to a set of traces $\mathcal{P}$ and RE the set of mutually exclusive results. We define the set of all possible results interactions as $RI = \{\{x_1, \ldots, x_n\} : x_1 \in S_1 \wedge S_1 \in RE \wedge \cdots \wedge x_n \in S_n \wedge S_n \in RE \wedge n = |RE|\}$. An action $a \in A_O$ has a proper execution iff $cond_a \subset RI$.*

The *proper execution of actions* is the first correctness criterion to be investigated on traces before synthesizing a template. Referring to the set of traces in Table 2, we find that this criterion is not met for activity bl. This problem is reported to the user so that the compliance rules are refined and a new set of traces is extracted.

Another correctness criterion for a set of traces is *data-completeness*. A set of traces $\mathcal{P}$ is data-complete if for every possible combination of results resulting from the *mandatory* activities, there is a trace in which this combination occurs.

Definition 4 (Traces Data-Completeness). *Let $\mathcal{P}$ be a set of traces, AM be the set of action mappings, $A_M = A \setminus A_O$ be the set of mandatory actions and $RE_M = \{S : \exists a \in A_M.(a, S) \in AM \wedge S \neq \emptyset\}$ be the set of mutually exclusive results of mandatory actions. We define the set $CO = \{\{x_1, \ldots, x_n\} : x_1 \in S_1 \wedge S_1 \in RE_M \wedge \cdots \wedge x_n \in S_n \wedge S_n \in RE_M \wedge n = |RE_M|\}$. The set of traces $\mathcal{P}$ is data-complete iff $\forall\, C \in CO\ \exists\, \sigma \in \mathcal{P}\ \exists\, s_i \in \sigma : \forall\, x \in C\ x \in s_i$ where $i > 0$.*

Even if data incompleteness is detected for a set of traces, a process template may be generated. Nevertheless, the template would suffer from deadlocks as for some combinations of results, continuation of processing is not defined. Therefore, we proceed solely in case the set of traces shows data completeness.

Example. For our running example, we find that the set of traces lacks the proper execution condition for activity bl. To address this issue, a compliance expert might add an explicit condition to black list a client only if the evaluation fails. This is represented by an additional constraint $\mathbf{G}\ (bl \Rightarrow ef)$. Repeating all steps from satisfiability checking to extracting traces yields a set of traces that satisfies the two correctness criteria above.

3.5 Generating Process Templates

Given a set of traces within which activities have proper execution conditions, *process mining* [4] is applied to generate a process template. Most mining algorithms neglect the difference between control flow dependencies and data flow dependencies when

generating a process model. Therefore, we cannot apply an existing algorithm directly. Instead, we use the α-algorithm [4] and incorporate the respective data aspects.

Order of actions. As a first step, we extract the precedence of actions. To this end, we employ an adapted version of the order relations known from the α-algorithm [4].

Definition 5 (Order Relations). *Let $\mathcal{P}$ be a set of traces and A the sets of actions. We define the following* order relations *for two actions $a_1, a_2 \in A$ with $R_{a_2} = \bigcup_{(a_1,S)\in AM} S$ as the set of results of a_2.*

$a_1 > a_2$**:** *iff either*
- $R_{a_2} = \emptyset$ *(i.e. a_2 has no results), and there is a trace $\sigma : s_0, \ldots, s_n \in \mathcal{P}$, such that $a_1 \in s_i \wedge a_2 \in s_{i+1}$ for some $0 \leq i < n$, or*
- $R_{a_2} \neq \emptyset$ *(i.e. a_2 has results), and $\forall\, r \in R_{a_2}$ there is a trace $\sigma : s_0, \ldots, s_n \in \mathcal{P}$, such that $a_1 \in s_i \wedge a_2 \in s_{i+1} \wedge r \in s_{i+1}$ for some $0 \leq i < n$.*

$a_1 \gg a_2$**:** *iff $a_1 > a_2$ and $a_2 \not> a_1$*

$a_1 \rightarrow a_2$**:** *iff $a_1 \gg a_2$ and $\nexists\, a_3 \in A : a_1 \gg^+ a_3 \wedge a_3 \gg a_2$ with $\gg^+$ as the transitive closure of $\gg$*

For two actions ordered by $>$, we know that the first action appears immediately before the second action. This notion of order is stronger than the one originally used in the α-algorithm [4]. If $a_1 > a_2$ and $a_2 \not> a_1$, then we conclude that a_1 precedes a_2, i.e., $a_1 \gg a_2$. However, it might be the case that $a_1 \gg a_2$, $a_2 \gg a_3$ and $a_1 \gg a_3$. In this case, we drop the dependency $a_1 \gg a_3$ as it unnecessarily complicates the template synthesis. Thus, we use the more strict precedence relation $\rightarrow$.

In contrast to common order relations known in process mining, the precedence dependencies in our approach may be guarded by conditions, captured as follows.

Definition 6 (Precedence Condition). *Let $\mathcal{P}$ be a set of traces, A the sets of actions, AM the mapping from actions to results, and $a_1, a_2 \in A$ two actions in precedence, $a_1 \rightarrow a_2$. Let $E = \{r | r \in \bigcup_{(a_1,S)\in AM} S \wedge (\exists\, \sigma = s_0, \ldots, s_i, s_{i+1}, \ldots, s_n \in \mathcal{P} : a_1 \in s_i \wedge r \in s_i \wedge a_2 \in s_{i+1})\}$ be the set of results of a_1 under which a_2 is observed. Then, we define the* precedence condition $cond(a_1, a_2)$ *as follows.*

$$cond(a_1, a_2) = \begin{cases} \bigvee_{r\in E} & \textit{iff} \quad E \subset \bigcup_{(a_1,S)\in AM} S \ \wedge \ E \neq \emptyset. \\ true & \textit{otherwise}. \end{cases}$$

According to this definition, we distinguish two types of precedence conditions. First, precedence holds for a proper subset of the results of the first action. Then, the precedence condition is the disjunction of results that can be caused by the first action. The second case captures unconditioned precedence, i.e., the precedence holds independent of any results. For our running example, we observe $ra \rightarrow og$. This precedence is guarded, as we observe this dependency solely in case of the result rl. In other words, only if action ra yields the result rl, we observe the action og subsequently.

Synthesis of process model. Based on the precedence among activities, the precedence conditions, along with the knowledge on optionality of activities, we proceed to build a process template. First, the overall structure of a process model is derived from the

precedence relation. This step yields a graph with all nodes representing actions, while the precedence relation defines directed edges between them. Second, control nodes (split and join nodes) that realize the behavior routing in the process model have to be introduced whenever a node has more than one predecessor or successor.

Starting with split nodes, our approach inserts nodes that implement either AND-, XOR-, or OR-logic. The routing semantics depends on the precedence conditions for the edges to succeeding nodes. If all precedences originating at an action are unconditioned, an AND-split node is inserted. If all precedences are conditioned and those conditions do not overlap, an XOR-split node is inserted and each outgoing edge inherits the respective precedence condition. Similarly, an OR-split is applied if the conditions are overlapping.

The case of join nodes, nodes with multiple predecessors, is not straightforward. We distinguish the following cases.

- All precedences of an action a are conditioned, we use an AND-join to synchronize these conditions.
- Only a proper subset of precedences of an action a is conditioned, we use an OR-join to synchronize any subset of these conditions.
- All precedences of an action a are unconditioned. If a is mandatory, we apply either an OR-join or an AND-join. The former is used if at least one of the preceding actions is optional; the latter is used in all other cases. If a is optional, we proceed as follows. An AND-join is applied to synchronize all precedences. Moreover, for all combinations of results of preceding actions of a, we check for a state in which the execution of a is observed as well. In other words, we identify all combinations of results of preceding actions under which a can occur. The disjunction of these result combinations is then used as a precedence condition for the edge between the AND-join and action a.

Applying these steps yields a process template. Still, our approach to model synthesis is rather naive and may create OR-joins for which the synchronization behavior could be implemented using solely AND- and XOR-joins. However, existing methods for restructuring a process model are used to replace these OR-joins with a semantically corresponding structure of AND- and XOR-joins, see [22].

Example. After we adapted the set of constraints for our running example as discussed above, we derive the basic graph structure for the template based on the precedence relation. Fig. 2 visualizes this structure in a BPMN-like notation. Here, the start and end actions are represented by start and end events. Activities depicted with a dashed border are optional. After inserting control nodes (aka gateways in BPMN) into the graph, the complete process template is derived. The first version of the generated process template is shown in Fig. 3a. Application of the restructuring according to [22] yields the process template shown in Fig. 3b.

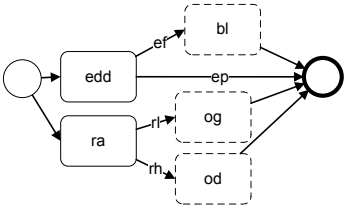

Fig. 2. Precedence among actions

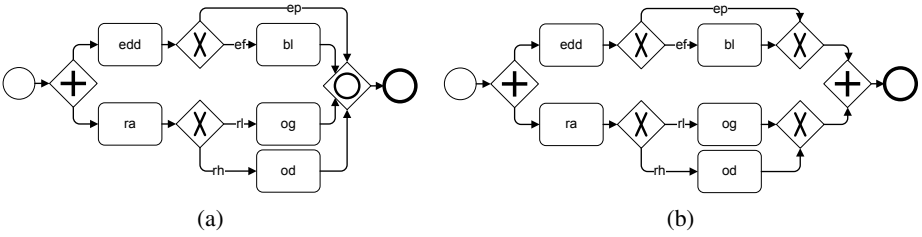

Fig. 3. (a) The process template for our example. (b) The restructured process template.

3.6 Evaluation of the Synthesized Process Template

Process templates aim at supporting experts in getting a better understanding of the compliance aspects and to discover missing or under-specified requirements. Such under-specification is manifested in the process template in terms of semantical problems. Those problems can only be detected by human experts. In this section, we will further elaborate on the running example to illustrate such problems. Using the process template in Fig. 3a as a basis of the discussion between compliance expert and business expert, they identify that the template allows for executing both black listing the client and granting to open the account in the same instance. This is an example of the aforementioned semantical problems caused by under-specified compliance rules. The compliance expert refines the set of constraints by indicating that black listing and granting open the account are contradicting, cf. the CA relation in Section 3.2, formalized as $\mathbf{G}\ (og \Rightarrow \mathbf{G}\ (\neg bl))$ and $\mathbf{G}\ (bl \Rightarrow \mathbf{G}\ (\neg og))$. Repeating the steps of our approach reveals that the adapted set of compliance rules yields a set of traces that is data incomplete. This is explained based on the two added constraints as follows. By forcing bl and og to be exclusive, we implicitly require bl to be executed only with the condition $ef \wedge rh$, while og is executed only with the condition $ep \wedge rl$. Other combinations of results are not considered. There is no trace that addresses the situation where $ef \wedge rl$ holds in some state. This contradicts with our requirement to execute either og or od in each run. Since the condition $ef \wedge rl$ enables neither of them, it is not observed in any of the generated traces.

As a consequence, another adaptation of our set of compliance requirements is needed. The missing interaction $ef \wedge rl$ has to be handled. One solution is to update the conditions under which og and od are executed, i.e., $\mathbf{G}\ (og \Rightarrow ep \wedge rl)$ and $\mathbf{G}\ (od \Rightarrow (ef \vee rh))$. With these updated constraints, another iteration of behavior synthesis is started. This time, the generated set of traces shows data completeness. The final generated process template is visualized in Fig. 4.

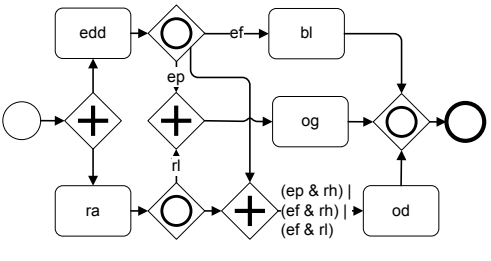

Fig. 4. A compliant process template where bl and og are exclusive and conditions adjusted

4 Implementation

We created a prototypical implementation to validate our approach. Fig. 5 shows a snapshot of it. It relies on a specification of domain knowledge, such as activity results and contradicting activities, which has to be defined once by a human expert. Given a set of compliance rules, our implementation adds extra rules to control the behavior synthesis and to enforce domain knowledge automatically. The satisfiability checking is done by an implementation of Wolper's method for checking LTL satisfiability [24] developed by the authors at the School of Computer Science of the Australian National University[1]. If the rules are satisfiable, the checker generates the pseudomodel of all possible traces. Next, our implementation extracts finite traces, analyzes them and synthesizes the process template, if the extracted traces pass quality tests, cf. Section 3.4. At that point, the resulting template is visualized using GraphViz [9]. In case that traces do not pass checks, the found problems are reported on the "Analysis result" tab.

There is the potential for a state space explosion, especially since the additional constraints of the process are unrestricted logical formulae. Even without pathological constraints, if there is a lot of freedom or non-local conditions then the satisfiability checking phase can take a considerable amount of time. The *once* constraint helps limit this, and too much freedom can often be a sign that other conditions have been omitted. We aim evaluating these issues in further case studies.

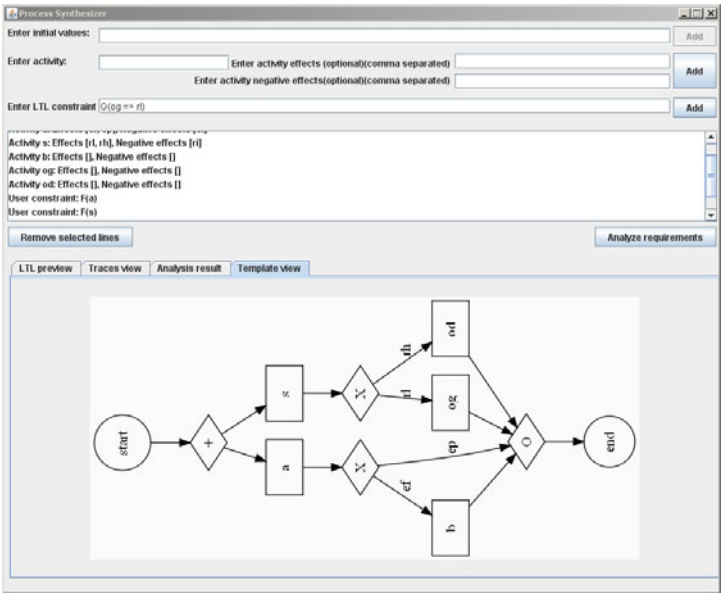

Fig. 5. A snapshot of the process synthesis tool

[1] Source code available at
`http://users.cecs.anu.edu.au/~rpg/PLTLProvers/pltlmultipass.tar`

5 Related Work

Compliance checking of business process models with a focus on execution order constraints has been approached from two angles: namely compliance by design and compliance checking of existing models. The latter has been tackled using model checking techniques [6,10,14]. Our work follows a compliance by design approach that has also been advocated in [11,12,13,15,16,25]. Close to our work, the authors of [12,11] employ temporal deontic assignments to specify what can or must be done at a certain point in time and synthesize a process template from these assignments. In contrast to our work, however, the approach is limited to temporal dependencies between activity executions and the underlying logic requires an encoding of these dependencies via explicit points in time. Another approach to synthesize compliant processes was introduced in [25]. The authors employ a set of compliance patterns expressed in Linear Temporal Logic (LTL). For each pattern a finite state automaton (FSA) is defined. To synthesize a process, the FSAs of the involved patterns are composed. Next, the user is required to select for each composition an execution path in order to synthesize the process. That approach is able to generate processes with sequence and choice only. Moreover, it does not consider data flow aspects in the synthesized process.

Related to our approach to process model synthesis is work on process mining, which aims at automatic construction of a process model from a set of logs [5,4,3]. We adapted the α-algorithm [4], a standard mining approach, for our purposes. Besides the commonalities, there are some important differences between process mining and process template synthesis. We consider control flow routing based on data values. This aspect is often neglect in process mining algorithms. Only recently, time information and data context have been considered when predicting the continuation of a trace based on its current state [21,2]. Further, process mining approaches have to be robust against incorrect data (log noise). As we derive a model from artificially generated traces, this is not an issue for our approach.

Work on declarative business process modeling is also related to our work. The authors of [17,19] propose to model processes by specifying a set of execution ordering constraints on a set of activities. These constraints are mapped onto LTL formulas; which are used to generate an automaton that is used to both guide the execution and monitor it. That is similar to our approach of generating a pseudomodel. Recently, the authors also showed how finite traces that respect interleaving semantics can be extracted from a set of LTL constraints [18]. The major difference from our work is that [18] does not model data constraints as we do. They also change the semantics of LTL rather than by using standard LTL as we do. Finally, we initially tried the approach of extracting Büchi automata from our LTL specifications for our example, but found that the automata approach required hours to return the automata whereas our LTL satisfiability checker returns a pseudomodel in less than a second.

6 Conclusion

In this paper, we introduced an approach to synthesize business process templates out of a set of compliance rules expressed in LTL. We also showed that extra domain-specific

knowledge is required to decide about consistency of such requirements and introduced an LTL encoding for compliance rules and domain knowledge. This was used to generated traces, which are analyzed for inconsistencies. Finally, we proposed an approach to the synthesis of process templates that goes beyond existing work on process mining by focusing on data dependencies of activity execution. We also discussed the analysis of generated templates with respect to semantical errors.

In our approach, we addressed control- and data-flow aspects of compliance rules, in contrast to similar approaches that focus on control-flow aspects only. The consideration of data-flow aspects comes with new challenges which we addressed in this paper by introducing correctness criteria for the set of generated traces. We illustrate that data dependencies may show rather interactions that are hard to handle at the first place. As a consequence, our approach is iterative – the required knowledge is built incrementally each time constraints are under-specified. In future work, we want to consider constraints on role resolution for generating process templates.

References

1. Sarbanes-Oxley Act of 2002. US Public Law 107–204 (2002)
2. van der Aalst, W.M.P., Pesic, M., Song, M.: Beyond process mining: From the past to present and future. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 38–52. Springer, Heidelberg (2010)
3. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., de Medeiros, A.K.A., Song, M., Verbeek, H.M.W.E.: Business process mining: An industrial application. Inf. Syst. 32(5), 713–732 (2007)
4. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. 16(9), 1128–1142 (2004)
5. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 469–483. Springer, Heidelberg (1998)
6. Awad, A., Weidlich, M., Weske, M.: Visually specifying compliance rules and explaining their violations for business processes. J. Vis. Lang. Comput. 22(1), 30–55 (2011)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
8. Commission, F.S.: Guidelines on anti-money laundering & counter-financing of terrorism (2007)
9. Ellson, J., Gansner, E.R., Koutsofios, E., North, S.C., Woodhull, G.: Graphviz - open source graph drawing tools. In: Graph Drawing, pp. 483–484 (2001)
10. Förster, A., Engels, G., Schattkowsky, T., Van Der Straeten, R.: Verification of Business Process Quality Constraints Based on VisualProcess Patterns. In: TASE, pp. 197–208. IEEE Computer Society Press, Los Alamitos (2007)
11. Goedertier, S., Vanthienen, J.: Compliant and Flexible Business Processes with Business Rules. In: BPMDS. CEUR Workshop Proceedings. CEUR-WS.org, vol. 236 (2006)
12. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes with Obligations and Permissions. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 5–14. Springer, Heidelberg (2006)
13. Lu, R., Sadiq, S.K., Governatori, G.: Compliance Aware Business Process Design. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 120–131. Springer, Heidelberg (2008)

14. Lui, Y., Müller, S., Xu, K.: A Static Compliance-checking Framework for Business Process Models. IBM Systems Journal 46(2), 335–362 (2007)
15. Milosevic, Z., Sadiq, S., Orlowska, M.: Translating Business Contract into Compliant Business Processes. In: EDOC, pp. 211–220. IEEE Computer Society, Los Alamitos (2006)
16. Namiri, K., Stojanovic, N.: Pattern-Based Design and Validation of Business Process Compliance. In: Chung, S. (ed.) OTM 2007, Part I. LNCS, vol. 4803, pp. 59–76. Springer, Heidelberg (2007)
17. Pesic, M., van der Aalst, W.M.P.: A Declarative Approach for Flexible Business Processes Management. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006)
18. Pesic, M., Bosnacki, D., van der Aalst, W.M.P.: Enacting declarative languages using ltl: Avoiding errors and improving performance. In: SPIN 2010. LNCS, vol. 6349, pp. 146–161. Springer (2010)
19. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: Full Support for Loosely-Structured Processes. In: EDOC, pp. 287–300. IEEE Computer Society, Los Alamitos (2007)
20. Pnueli, A.: The temporal logic of programs. In: SFCS, pp. 46–57. IEEE Computer Society, Washington, DC, USA (1977)
21. Schonenberg, H., Jian, J., Sidorova, N., van der Aalst, W.M.P.: Business trend analysis by simulation. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 515–529. Springer, Heidelberg (2010)
22. Vanhatalo, J., Völzer, H., Leymann, F., Moser, S.: Automatic workflow graph refactoring and completion. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 100–115. Springer, Heidelberg (2008)
23. Wolper, P.: Temporal logic can be more expressive. Information and Control 56, 72–99 (1983)
24. Wolper, P.: The tableau method for temporal logic: an overview. Logique et Analyse 110-111, 119–136 (1985)
25. Yu, J., Han, Y., Han, J., Jin, Y., Falcarin, P., Morisio, M.: Synthesizing service composition models on the basis of temporal business rules. J. Comput. Sci. Technol. 23(6), 885–894 (2008)