Compliance by design for artifact-centric business processes[☆]

Niels Lohmann

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany

Abstract

Compliance to legal regulations, internal policies, or best practices is becoming a more and more important aspect in business processes management. Compliance requirements are usually formulated in a set of rules that can be checked during or after the execution of the business process, called *compliance by detection*. If noncompliant behavior is detected, the business process needs to be redesigned. Alternatively, the rules can be already taken into account while modeling the business process to result in a business process that is *compliant by design*. This technique has the advantage that a subsequent verification of compliance is not required.

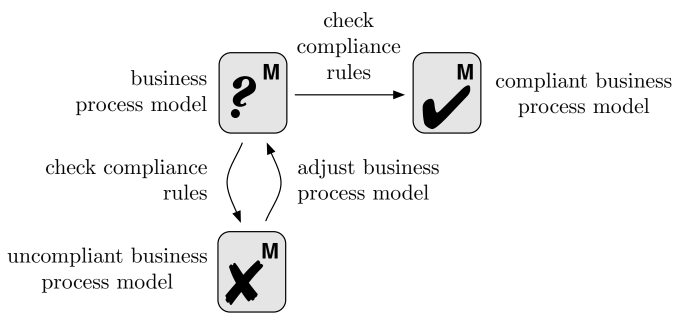
This paper focuses on compliance by design and employs an *artifact-centric* approach. In this school of thought, business processes are not described as a sequence of tasks to be performed (i. e., imperatively), but from the point of view of the artifacts that are manipulated during the process (i. e., declaratively). We extend the artifact-centric approach to model compliance rules and show how compliant business processes can be synthesized automatically.

1. Introduction

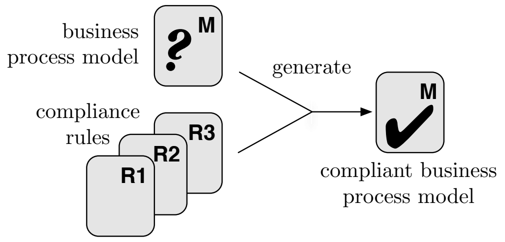
Business processes are the main asset of companies as they describe their value chain and fundamentally define the "way, businesses are done". Beside fundamental correctness criteria such as soundness (i. e., every started case is eventually finished successfully), also nonfunctional requirements have to be met. Such requirements are often collected under the umbrella term *compliance*. They include legal regulations such as the often cited Sarbanes-Oxley Act to fight accounting frauds, internal policies to streamline the in-house processes, or industrial best practices to reduce complexity and costs as well as to facilitate collaborations. Finally, compliance can be seen as a means to validate business processes [2].

[☆]Extended abstract appeared as [1]. Extensions are discussed in Sect. 7.

Email address: `niels.lohmann@uni-rostock.de` (Niels Lohmann)


check compliance rules
business process model
M
compliant business process model
check compliance rules
adjust business process model
uncompliant business process model

(a) compliance by detection



(b) compliance by design

Figure 1: Approaches to achieve compliance

Compliance requirements are usually defined without a concrete business process in mind, for instance in legal texts such as *"The Commission shall [...] certify [...] that the signing officers have designed such internal controls to ensure that material information [...] is made known [...] during the period in which the periodic reports are being prepared"*.[1] Such informal descriptions then must be translated by domain experts into precise rules that unambiguously capture the essence of the requirement in a shape that it can be checked in concrete business process. The example above could yield a rule such as *"Information on financial reports must be sent to the press team by the signing officers at most two weeks after signing"*. The rules clarify who has to take which actions and when. That is, they specifically deal with the execution order of actions of the business process or the reachability of data values. This formalization of domain-specific knowledge is far from trivial and out of scope of this paper. In the remainder, we assume — similar to other approaches [3] — that such rules are already present.

Compliance rules are often *declarative* and describe *what* should be achieved rather than *how* to achieve it. Temporal logics such as CTL [4], LTL [5] or PLTL [6] are common ways to formalize such declarative rules. To make these logics approachable for nonexperts, also graphical notations have been proposed [7, 8].

[1]Excerpt of Title 15 of the United States Code, § 7241(a)(4)(B).

Given such rules, compliance of a business process model can be verified using model checking techniques [9]. These checks can be classified as *compliance by detection*, also called after the fact or retrospective checking [10]. Their main goal is to provide a rigorous proof of compliance. In case of noncompliance, diagnosis information may help to fix the business process toward compliance. This step can be very complicated, because the rules may affect various parts and agents of the business process (e. g., financial staff and the press team). Furthermore, the declarative nature of the rules does not provide recipes on how to fix the business process. To meet the previous example rule, an action "send information to press team" needs to be added to the process and must be executed at most two weeks after the execution of an action "sign financial report". Compliance can be eventually reached after iteratively adjusting the business process model, cf. Fig. 1(a). The main advantage of this approach is the fact that it can be applied to already running business processes.

An alternative approach focuses on the early design phases and takes a business process model and the compliance rules as input and automatically generates a business process model that is *compliant by design* [10], cf. Fig. 1(b). This has several advantages: First, a subsequent proof and potential corrections are not required. This may speed up the modeling process. Second, the approach is flexible as the generation can be repeated when rules are added, removed, or changed. Third, the approach is complete in the sense that an unsuccessful model generation can be interpreted as "the business process cannot be *made* compliant" rather than "the current model is not compliant". Fourth, compliance is not only detected, but actually enforced. That is, noncompliant behavior becomes technically impossible.

This paper investigates the latter compliance-by-design approach. We employ a recent framework for *artifact-centric* business processes [11]. In this framework, a business process is specified by a description of the life cycles of its data objects (artifacts). From this declarative specification, which also specifies agents and locations of artifacts, a sound, operational, and interorganizational business process can be automatically generated.

Contribution. This paper makes three contributions: First, we extend the artifact-centric framework [11] to model a large family of compliance rules. Second, we use existing tools and techniques to achieve not only soundness, but also compliance by design. We also sketch the diagnosis of noncompliant models. Third, we show how a role-based access control can be added to the artifact-centric framework and how additional compliance requirements such as separation of duties can be modeled and enforced.

Organization. The next section introduces a small example we use throughout the paper to exemplify our approach and later extensions. Section 3 sets the stage for our later contributions. There, we introduce artifact-centric business processes and correctness by design. In Sect. 4, we demonstrate how a large family of compliance rules can be expressed with in our approach. Section 5 presents how compliance by design can be achieved. We also discuss the diagnosis of unrealizable compliance rules. In Sect. 6, we introduce a role-based access control. We show how additional compliance aspects of a business process can be expressed and present an extension to our formal model. Section 7 brings our approach in the context of related work, before Sect. 8 concludes the paper.

2. Running example: insurance claim handling

We use a simple insurance claim handling process (based on [12]) as running example for this paper. In this process, a customer submits a claim to an insurer who then prepares a fraud detection check offered by an external service. Based on the result of this check, the claim is either (1) assessed and the settlement estimated, (2) detected fraudulent and reported, or (3) deemed incomplete. In the last case, further information are requested from the customer before the claim is resubmitted to the fraud detection service. In this situation, the customer can alternatively decide to withdraw the claim. On successful assessment, a settlement case is processed by a financial clerk. The claim is settlement paid in several rates or all at once. A single complete payment further requires an authorization of the controlling officer. When the settlement is finally paid, the claim is archived.

One way to model this process is to explicitly order the actions to be taken and to give an operational business process model. An alternative to this *verb-centric* approach offers an artifact-centric framework which starts by identifying what is acted on (*noun-centric*) and to derive a business process from the life cycles of the involved artifacts. We shall discuss artifact-centric business processes in the next section.

3. Artifact-centric business processes

In this section, we shall introduce artifact-centric business processes [11]. We first give an informal overview of all concepts involved. Then we present a Petri net formalization and discuss it in on the basis of the running example. Admittedly, this section takes a large part of this paper, but is required to discuss the contributions to compliance.

3.1. Informal overview

Artifact-centric modeling promotes the data objects of a business process (called *artifacts*) and their life cycles to first-class citizens. In the running example, we consider two artifacts: an insurance claim file and a settlement case. Each life cycle describes how the state of an artifact may evolve over time. The actions that change states are executed by *agents*, for instance the customer, the insurer, the financial clerk, and the controlling officer. As multiple agents can participate in a business process, the artifact-centric approach is particularly suited to model interorganizational business processes.

Each artifact has at least one *final state* which models a successful processing of the artifact, for instance "claim archived", or "settlement paid". Artifact-centric business processes are inherently *declarative*: the control flow of the business process is not explicitly modeled, but follows from the life cycles of the artifacts. That is, any execution of actions that brings all artifacts to a final state can be seen as sound. However, not every sound execution makes sense. For instance, semantically ordered actions of different and independently modeled artifacts (e. g., "assess claim" and "create settlement") may be executed in any order. In addition, not every combination of final states may be desirable, for instance "claim withdrawn" in combination with "settlement paid". Therefore, the executions have to be constrained using *policies* and *goal states*. A policy is a way of expressing constraints between artifacts. For instance, a policy may constrain the order of state changes in different artifacts (e. g., always executing "assess claim" before "pay settlement"). Finally, goal states restrict final states by reducing those combinations of artifacts' final states that should be considered successful.

In recent work [11], we presented an approach that takes artifacts, policies, and goal states as input and automatically synthesizes an interorganizational business process. This business process has two important properties: First, it is *operational*. It explicitly models which agent may perform which action in which state. In addition to the control flow, we can also derive the data flow and even the message flow, because artifacts may be sent between agents. Operational models can be translated into languages such as BPMN [13] or WS-BPEL [14] and can be easily refined toward execution [15]. Second, the business process is *weakly terminating*. Weak termination is a correctness criterion that ensures that a goal state is always reachable from every reachable state. It is very similar to the *soundness property* [16], but without the requirement that every modeled action can actually be executed. This relaxation is reasonable, because artifacts can be used in different contexts and the exclusion of certain behavior in a concrete business process is not necessarily a design flaw. To ensure weak termination, any actions that would lead to deadlocks or livelocks are removed. This means that the approach is *correct by design*. To summarize, the artifact-centric approach allows to model artifacts and to restrict their manipulation by additional domain knowledge such as policies and goal states. From this declarative model we can then automatically generate a weakly terminating operational model. Figure 2 illustrates the overall approach.

3.2. Formalization

We model artifact-centric business processes with Petri nets [17]. Petri nets combine a simple graphical representation with a rigorous mathematical foundation. They can naturally express locality of actions in distributed systems. This allows us to model the life cycles of several artifacts independently, yielding a more compact model compared to explicit state machines.

Definition 1 (Petri net). A *Petri net* $N = [P, T, F, m_0]$ consists of two finite and disjoint sets P of *places* and T of *transitions*, a *flow relation* $F \subseteq (P \times T) \cup (T \times P)$, and an *initial marking* m_0. A marking $m : P \to \mathbb{N}$ represents a state of the Petri net and is visualized as a distribution of tokens on the places. Transition t is enabled in marking m iff, for all $[p, t] \in F$, $m(p) > 0$.

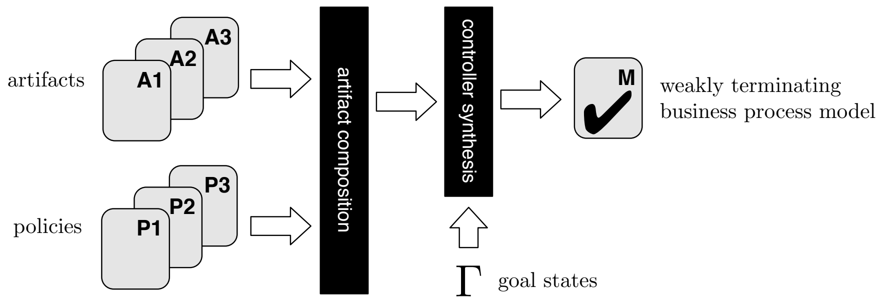


Figure 2: Artifact-centric business processes in a nutshell

An enabled transition t can fire, transforming m into the new state m' with $m'(p) = m(p) - W([p,t]) + W([t,p])$ (for all places $p \in P$) where $W([x,y]) = 1$ if $[x,y] \in F$, and $W([x,y]) = 0$, otherwise.

A Petri net shall describe the life cycle of an artifact. For our purposes, we have to extend this model with several concepts: Each transition is associated with an action from a fixed set $\mathcal{L} = \mathcal{L}_c \cup \mathcal{L}_u$ of *action labels*. This set is partitioned into a set $\mathcal{L}_c$ of *controllable actions* that are executed by agents and a set $\mathcal{L}_u$ of *uncontrollable actions* that are not controllable by any agent, but are under the influence of the environment. Such uncontrollable actions are suitable to model choices that are external to the business process model, such as the outcome of a service call (e.g., to a fraud detection agency) or just choices whose decision process is not explicitly modeled at this level of abstraction.

Definition 2 (Artifact [11]). An *artifact* $A = [N, \ell, \Omega]$ consists of (1) a Petri net $N = [P, T, F, m_0]$, (2) a transition labeling $\ell : T \to \mathcal{L}$ associating actions with Petri net transitions, and (3) a set Ω of *final markings* of N representing endpoints in the life cycle of the artifact.

Running example (cont.). Figure 3 depicts the claim and the settlement artifacts. Each transition is labeled by the agent that executes it (*ins*urer, *cus*tomer, *con*troller, and *fin*ancial cleark) or is shaded gray in case of uncontrollable actions. Two additional places ("@insurer" and "@controller") model the *locaction* of the insurance claim file. The artifact-centric approach allows to distinguish physical objects (e. g., documents or goods) that need to be transferred between agents to execute certain actions and virtual objects (e. g., data bases or electronic documents). By taking the shape of the artifacts and their location into account, we can later derive explicit message transfer among the agents. In our example, we assume that after submitting the claim, a physical file is created by the insurer which can be sent to the controlling officer by executing the respective action "send to controller". We further assume that the settlement case is a data base entry that can be remotely accessed by the insurer, the financial clerk, and the controlling officer.

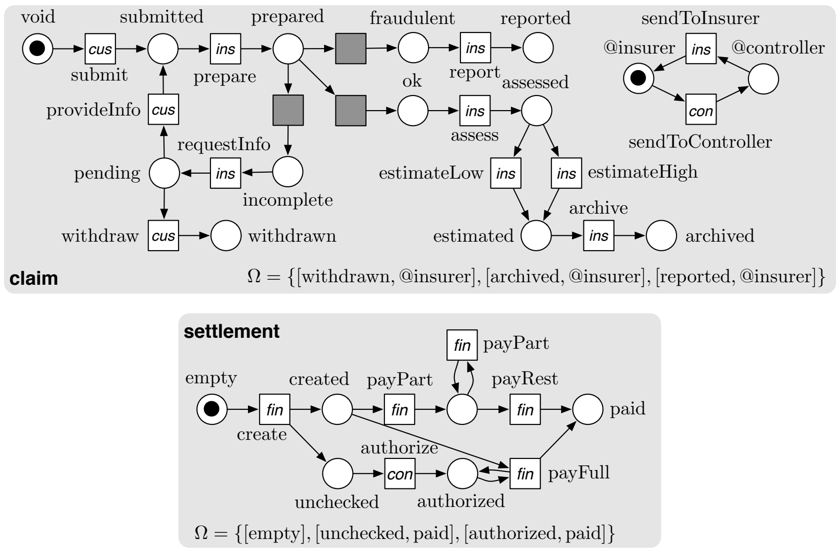


Figure 3: Artifacts of the running example

We can now model each artifact of our business process independently with a Petri net model. Together, the artifacts implicitly specify a process of actions that may be performed by the agents or the environment according to the life cycles of the artifacts. This global model is formalized as the union of the artifact models, which is again an artifact.

Definition 3 (Artifact union [11]). Let $A_1, \dots, A_n$ be artifacts with pairwise disjoint Petri nets $N_1, \dots, N_n$. Define the *artifact union* $\bigcup_{i=1}^{n} A_i = [N, \ell, \Omega]$ to be the artifact consisting of (1) $N = [\bigcup_{i=1}^{n} P_i, \bigcup_{i=1}^{n} T_i, \bigcup_{i=1}^{n} F_i, m_{0_1} \oplus \dots \oplus m_{0_n}]$, (2) $\ell(t) = \ell_i(t)$ iff $t \in T_i$ $(i \in \{1, \dots, n\})$, and (3) $\Omega = \{m_1 \oplus \dots \oplus m_n \mid m_i \in \Omega_i \wedge 1 \leq i \leq n\}$. Thereby, $\oplus$ denotes the composition of markings: $(m_1 \oplus \dots \oplus m_n)(p) = m_i(p)$ iff $p \in P_i$. This composition can be canonically extended to sets of markings: $M_1 \oplus M_2 = \{m_1 \oplus m_2 \mid m_1 \in M_1 \wedge m_2 \in M_2\}$.

The previous definition is of rather technical nature. Disjointness of places and transitions an be achieved by renaming, for instance using prefixes. The only noteworthy property is that the set of final markings of the union consists of all combinations of final markings of the respective artifacts. Conceptually, the union of the artifacts has several downsides as we discussed in Sect. 3.1. First, it may contain sequences of actions that reach deadlocks or livelocks. That is, the model is not necessarily weakly terminating. Second, the artifacts may evolve independently which may result in implausible execution orders or undesired final states. As stated earlier, the latter problems can be ruled out by defining *policies*, which restrict interartifact behavior, and *goal states*, which restrict the final states of the union. Before discussing this, we shall first cope with the first problem.

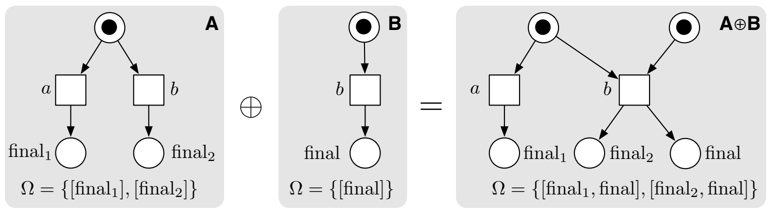

A
a
b
final$_1$
final$_2$
$\Omega = \{[\text{final}_1], [\text{final}_2]\}$
$\oplus$
B
b
final
$\Omega = \{[\text{final}]\}$
=
A⊕B
a
b
final$_1$
final$_2$
final
$\Omega = \{[\text{final}_1, \text{final}], [\text{final}_2, \text{final}]\}$

Figure 4: Composition of two artifacts

The transitions of the artifacts are labeled with actions that can be executed by agents. Hence, the agents have control about the evolution of the overall business process. To avoid undesired situations such as deadlocks or livelocks, their behavior needs to be coordinated. That is, it must be constrained such that every execution can be continued to a final state. This coordination can be seen as a *controller synthesis* problem [18]: given an artifact A, we are interested in a controller C (in fact, also modeled as an artifact) such that their interplay is weakly terminating. The interplay of two artifacts is formalized by their *composition*, cf. Fig. 4.

Definition 4 (Artifact composition [11]). Let A_1 and A_2 be artifacts. Define their *shared labels* as $S = \{l \mid \exists t_1 \in T_1, \exists t_2 \in T_2 : \ell(t_1) = \ell(t_2) = l\}$. The *composition* of A_1 and A_2 is the artifact $A_1 \oplus A_2 = [N, \ell, \Omega]$ consisting of:

- $N = [P, T, F, m_{0_1} \oplus m_{0_2}]$ with
  - $P = P_1 \cup P_2$,
  - $T = \big(T_1 \cup T_2 \cup \{[t_1, t_2] \in T_1 \times T_2 \mid \ell(t_1) = \ell(t_2)\}\big) \setminus \big(\{t \in T_1 \mid \ell_1(t) \in S\} \cup \{t \in T_2 \mid \ell_2(t) \in S\}\big)$,
  - $F = ((F_1 \cup F_2) \cap ((P \times T) \cup (T \times P))) \cup \{[[t_1, t_2], p] \mid [t_1, p] \in F_1 \vee [t_2, p] \in F_2\} \cup \{[p, [t_1, t_2]] \mid [p, t_1] \in F_1 \vee [p, t_2] \in F_2\}$,
- for all $t \in T \cap T_1$: $\ell(t) = \ell_1(t)$, for all $t \in T \cap T_2$: $\ell(t) = \ell_2(t)$, and for all $[t_1, t_2] \in T \cap (T_1 \times T_2)$: $\ell([t_1, t_2]) = \ell_1(t_1)$, and
- $\Omega = \Omega_1 \oplus \Omega_2$.

The composition $A_1 \oplus A_2$ is *complete* if for all $t \in T_i$ holds: if $\ell_i(t) \notin S$, then $\ell_i(t) \in \mathcal{L}_u$ ($i \in \{1, 2\}$).

Given an artifact A, we call another artifact C a *controller* for A iff (1) their composition $A \oplus C$ is complete and (2) for each reachable markings of the composition, a final marking $m \oplus m'$ of $A \oplus C$ is reachable. The existence of controllers (also called *controllability* [19]) is a fundamental correctness criterion for communicating systems such as services. It can be decided constructively [19]: If a controller for an artifact exists, it can be constructed automatically [20]. Note that the requirement of a complete composition makes sure that the controller does *not* constrain the execution of uncontrollable actions.

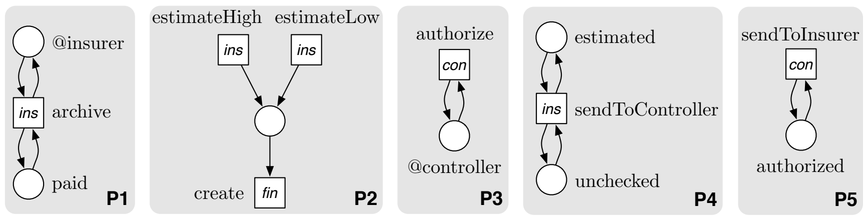


Figure 5: Policies for the running example

Finally, we can define goal states and policies to constrain the interartifact behavior. *Goal states* are a set of markings of the artifact union and are used as final markings during controller synthesis. To model interdependencies between artifacts, we employ *policies*. We also model policies with artifacts (similar to behavioral constraints [21]); that is, labeled Petri nets with a set of final markings. These artifacts have no counterpart in reality and are only used to model dependencies between actions of different artifacts. The application of policies then boils down to the composition of the artifacts with these policies.

Running example (cont.). To rule out implausible behavior, we further define the following policies to constrain interartifact behavior and the location's impact on actions:

P1 *The claim may be archived only if it resides at the insurer and the settlement is paid.*

P2 *A settlement may only be created after the claim has been estimated.*

P3 *To authorize the complete payment of the settlement, the claim artifact must be at hand to the controlling officer.*

P4 *The claim artifact may only be sent to the controller if it has been estimated and the settlement has not been checked.*

P5 *The claim artifact may only be sent back to the insurer if the settlement has been authorized.*

The policies address different aspects of the artifacts such as location (P1 and P3), execution order (P2), or data constraints (P4 and P5). The modeling of the policies as artifacts is straightforward and depicted in Fig. 5. Note that in policy P2, we use an unlabeled place to express the causality between the transitions. This place has no counterpart in any artifact and is added to the composition.

As goal states, we specify the set

$$\begin{aligned}\Gamma = \{&[\text{withdrawn}, @\text{insurer}, \text{empty}], [\text{reported}, @\text{insurer}, \text{empty}],\\ &[\text{archived}, @\text{insurer}, \text{paid}, \text{unchecked}], [\text{archived}, @\text{insurer}, \text{paid}, \text{authorized}]\}\end{aligned}$$

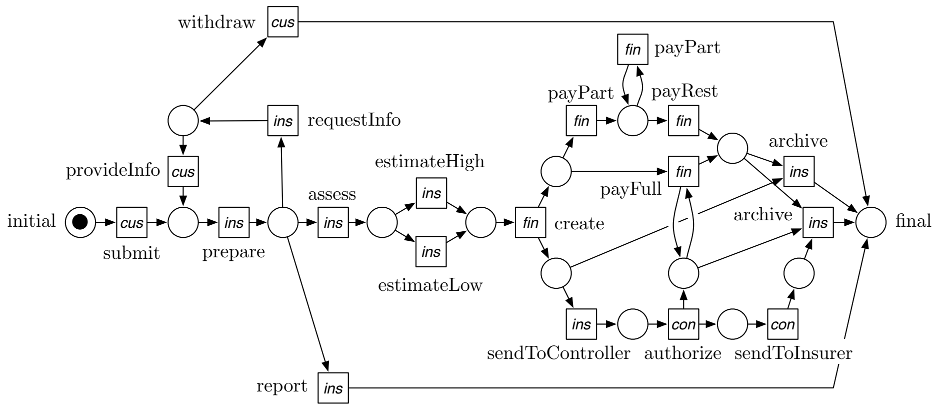


Figure 6: The running example synthesized as weakly terminating operational business process

which models four cases: withdrawal of the claim, detection of a fraud, and settlement with or without authorization. Taking the artifacts, policies, and goal states as input, we can *automatically* synthesize the weakly terminating and operational business process depicted in Fig. 6 using existing tools [20, 22].

4. Modeling compliance rules

This section investigates to what extend compliance rules can be integrated into the artifact-centric approach. Before we present different shapes of compliance rules and their formalization with Petri nets, we first discuss the difference between a policy and a compliance rule.

4.1. Enforcing policies vs. monitoring compliance rules

As described in the previous section, we use policies to express interdependencies between artifacts and explicitly restrict behavior by making the firing of transitions impossible. Policies thereby express domain knowledge about the business process and its artifacts and are suitable to *inhibit* implausible or undesired behavior. This finally affects the subsequent controller synthesis.

In contrast, a compliance rule specifies behavior that is not under the direct control of the business process designer. Consequently, a compliance rule *must not restrict the behavior of the process, but only monitor it to detect noncompliance*. For instance, a compliance rule must not disable external choices within the business process as they cannot be controlled by any agent. If such a choice would be disabled to achieve compliance, the resulting business process model would be spurious as the respective choice could not be disabled in reality. Therefore, compliance rules must not restrict the behavior of the artifacts, but only restrict the final states of the model. This may classify behavior as undesired (viz. noncompliant), but this behavior remains reachable. Only if this behavior can be circumvented by the controller synthesis, we faithfully found a compliant business process which can be actually implemented. We formalize this nonrestricting nature as *monitor property* [19, 21]. Intuitively, this property requires that in every reachable marking of an artifact, it holds that for each action label of that artifact a transition with that label is activated. This rules out situations in which the firing of a transition in a composition is inhibited by a compliance rule.

4.2. Expressiveness of compliance rules

Conceptually, we model compliance rules by artifacts with the monitor property. Again, adding a compliance rule to an artifact-centric model boils down to composition, cf. Def 4. The monitor property ensures that the compliance rule's transitions are synchronized with the other artifacts, but without restricting (i. e., disabling) actions. That is, the life cycle of a compliance rule model evolves together with the artifacts' life cycles, but may only affect the final states of the composed model.

In a finite-state composition of artifacts, the set of runs reaching a final state forms a regular language. The terminating runs of a compliance rule (i. e., sequences of transitions that reach a final marking) describe compliant runs. This set again forms a regular language. In the composition of the artifacts and the compliance rules, these regular languages are synchronized — viz. intersected — yielding a subset of terminating runs. Regular languages allow to express a variety of relevant scenarios. In fact, we can express all patterns listed by Dwyer et al. [23], including:

- enforcement and existence of actions (e. g., *"Every compliant run must contain an action 'archive claim'."*),
- absence/exclusion of actions (e. g., *"The action 'withdraw claim' must not be executed."*),
- ordering (precedence and response) of actions (e. g., *"The action 'create settlement' must be executed after 'submit claim', but before 'archive claim'."*), and
- numbering constraints/bounded existence of actions (e. g., *"The action "partially pay settlement" must not be executed more than three times".*).

The explicit model of data states of the artifacts further allows to express rules concerning data flow, such as:

- enforcement/exclusion of data states (e. g., *"The claim's state 'fraud reported' and the settlement's state 'paid' must never coincide."*), or
- data and control flow concurrence (e. g., "*The action 'publish review' may only be executed if the review artifact is in state 'reviewers blinded'.*").

Additionally, the explicit modeling of the location of the artifacts allows to express spacial constraints:

- enforcement/exclusion of actions at specific locations (e. g., *"The task 'sign' may only be executed if the contract artifact is at the human resources department"*), or
- enforcement/exclusion of the transport/transfer of an artifact in a certain state (e. g., *"iPhone prototypes must not leave the company premises after the operating system is installed."*).

On top of that, any combinations are possible, allowing to express complex compliance rules.

4.3. Limitations

Apart from the conceptual richness, the presented approach has some theoretical limits. First, it is not applicable to nonregular languages. For instance, a rule requiring that a compliant run must have an arbitrary large, but equal number of a and b actions or that a and b actions must be properly balanced (Dyck languages) cannot be expressed with a finite-state models. Second, rules that affect infinite runs (e. g., certain LTL formulae [5]) cannot be expressed. Infinite runs are predominantly used to reason about reactive systems. A business process, however, is usually designed to eventually reach a final state — this basically is the essence of the soundness property. Therefore, we shall focus on an interpretation of LTL which only considers finite runs, similar to a semantics described by Havelund and Roşu [24]. Third, just like Awad et al. [25], we also do not consider the $\mathbf{X}$ (next state) operator of CTL*, because we typically discuss distributed systems in which states are not partially ordered. Forth, we do not use timed Petri nets and hence can make no statements on temporal properties of business processes. However, we can abstract the variation of time by events such as "time passes" or data states such as "expired" as in [3, 26].

Conceptually, an extension toward more expressive constraints would be possible. For instance, pushdown automata could be used to express Dyck languages, a yielding context-free language as product. However, this extension would need further theoretical consideration as, for instance, controllability is undecidable in case of infinite state systems [27].

4.4. Example formalizations of compliance rules

As mentioned earlier, we again use artifacts (i. e., Petri nets with final markings and action labels) that satisfy the monitor property to model compliance rules. As an example, we consider the following compliance rules for our example insurance claim process:

R1 *All insurance claims with an estimated high settlement must be authorized.*

R2 *Customers must not be allowed to withdraw insurance claims.*

R3 *Settlements should be paid in at most three parts.*

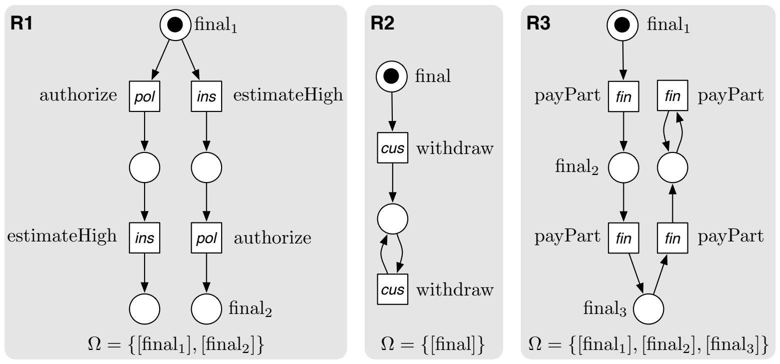


Figure 7: Compliance rules modeled as Petri nets

Figure 7 shows the Petri net formalizations of these compliance rules. In rule R1, we exploited the fact that the actions "authorize" and "estimateHigh" are executed at most once. In rule R2 and R3, the monitor property is achieved by allowing "withdraw" and "payPart" to fire in any reachable state. Without restriction of the behavior, the final markings classify executions as compliant or not. For instance, executing "estimateHigh" in rule R1 without eventually executing "authorize" does not reach the final marking [final$_2$]. Other examples can be formalized similarly.

4.5. Discussion

We conclude this section by a discussion of the implications of using Petri nets to formalize compliance rules.

- *Single formalism.* We can model artifacts, policies, and compliance rules with the same formalism. Though we do not claim that Petri nets should be used by domain experts to model compliance regulations, using a single formalism still facilitates the modeling and verification process. Furthermore, each rule implicitly models compliant behavior which can be simulated. This is not possible if, for instance, arbitrary LTL formulae are considered.

- *Level of abstraction.* Rules can be expressed using minimal overhead. Each rule contains only those places and transitions that are affected by the rule and plus some additional places to model further causalities. In particular, no placeholder elements (e. g., anonymous activities in BPMN-Q [8]) are required. These placeholder elements must not be confused with "wildcard" dependencies, for instance requiring input from artifact A, B, or C. To formalize such dependencies with our artifact model, they need to be unfolded explicitly. Of course, syntactic extensions may be introduced to modeling languages to compact the models.

- *Independent design.* The rules can be formulated independently of the artifact and policy models. That is, the modeler does not need to be confronted with the composite model. This modular approach is more likely to scale, because the rules can also be validated independently of the other rules.

- *Reusability.* The composition is defined in terms of action labels. Therefore, rules may be reused in different business process models as long as the labels match. This can be enforced using standard naming schemes or ontologies.

- *Runtime monitoring.* The monitor property ensures that the detection of noncompliant behavior is transparent to the process as no behavior is restricted. Therefore, the models of the compliance rules can be also used to check compliance during or after runtime, for instance by inspecting execution logs.

- *Rule generation.* Finally, the structure of the Petri nets modeling compliance rules is very generic. Therefore, it should be possible to automatically generate Petri nets for standard scenarios or to provide templates to which only the names of the constrained actions need to be filled. Also, the monitor property can be automatically enforced.

5. Compliance by design

This section presents the second contribution of this paper: the construction of business process models that are compliant by design. Beside the construction, we also discuss the diagnosis of noncompliant business process models.

5.1. Constructing compliant models

None of the compliance rules discussed in the previous section hold in the example process depicted in Fig. 6. This noncompliance can be detected by standard model checking tools. They usually provide a counterexample which describes how a noncompliant situation can be reached. For instance, the action sequence "1. submit, 2. prepare, 3. requestInfo, 4. withdraw" is a witness that the process does not comply with rule R2 from Sect. 4.4. To satisfy this requirement, the transition "withdraw" can be simply removed. However, implementing the other rules is more complicated, and each modification would require another compliance check.

We propose to *synthesize* a compliant model instead of verifying compliance. By composing the Petri net models of the artifacts (cf. Fig. 3), the policies (cf. Fig. 5), and the compliance rules (cf. Fig. 7) and by taking the goal states into account, we derive a Petri net that models the artifacts' life cycles that are restricted by the policies and whose final states are constrained by the goal states and the compliance rules. *Compliant behavior is now reduced to weak termination*, and we can apply the same algorithm [19] and tool [20] to synthesize

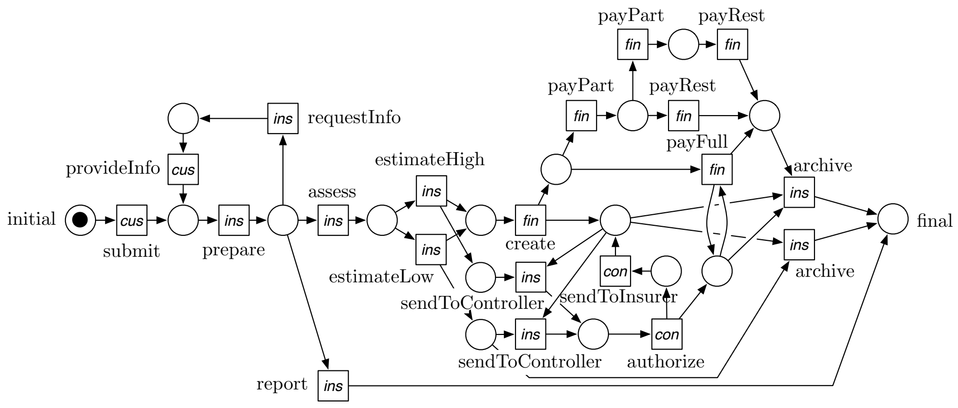


Figure 8: Operational business process satisfying the compliance rules R1–R3

a controller. If such a controller exists, it provides an operational model that specifies the order in which the agents need to perform their actions. This model is *compliant by design* — a subsequent verification is not required. Beside weak termination (and hence, compliance), the synthesis algorithm further guarantees the resulting model is *most permissive* [19]. That is, exactly that behavior has been removed that would violate weak termination. Another important aspect of the approach is its flexibility to add further compliance rules. That is, we do not need to edit the existing model, but we can simply repeat the synthesis for the new rule set.

Running example (cont.). Figure 8 depicts the resulting business process model. It obviously contains no transition labeled with "withdraw", but the implementation of the other rules yielded a whole different structure of the part modeling the settlement processing. *It is important to stress that the depicted business process model has been synthesized completely automatically* using the partner synthesis tool Wendy [20] and the Petri net synthesis tool Petrify [28]. Admittedly, it is a rather complicated model, but any valid implementation of the compliance rules would yield the same behavior or a subset. Though our running example is clearly a toy example, experimental results [20] show that controller synthesis can be effectively applied to models with millions of states.

5.2. Diagnosing noncompliant models

So far, we only considered the case that the business process can be constructed such that it satisfies the compliance rules. Then, we can find a controller that guarantees weak termination. In case a compliance rule is not met by the business process, no such controller exists. That is, the business process cannot be *made* compliant. Intuitively, the intersection between the behavior of the business process and the compliance rule is empty. Just as a controller would be a witness for compliant behavior, such witness does not exist in case of noncompliance.

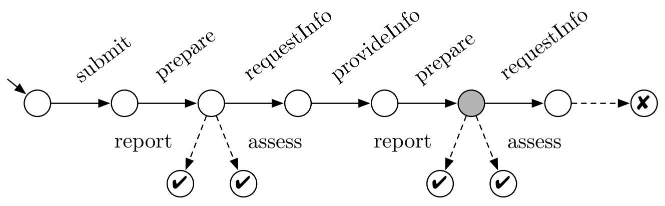


Figure 9: Counterexample for an unrealizable compliance rule

In previous work [29], we studied uncontrollable models and presented an algorithm that generates diagnosis information. This diagnosis information is presented as a graph that overapproximates the behavior of any controller. As no controller exists that can avoid states that violate weak termination, this graph contains paths to such deadlocks or livelocking states. These paths and the reason why they cannot be avoided by a controller serve as a counterexample similar to those of standard model checking. Note that such a counterexample not only describes noncompliance of a concrete business process model, but for a whole family of operational models that can be derived from a set of artifacts.

Running example (cont.). To exemplify this diagnosis information, consider the following compliance rule:

R4 *The customer should not be asked for further information more than once.*

This rule is not realizable in the business process, because the outcome of the fraud detection service is not under the control of any agent. Furthermore, rule R2 excluded the possibility for the customer to withdraw the claim. Hence, we cannot exclude a noncompliant run in which the transition “requestInfo” is fired more than once. The diagnosis algorithm (which is also implemented in the tool Wendy [20]) generates a graph similar to that depicted in Fig. 9. From this graph, we pruned those paths that eventually reach a final state (represented as check marks) and replaced them by dashed arrows. As we can see, a final state cannot be reached after the second “requestInfo” action is executed. However, we cannot avoid this situation, because in the gray state, the external service decides whether or not the claim is fraudulent or whether further information are required. For the business process, this choice is external and uncontrollable and hence it must be correct for any outcome.

6. Role-based access control

Until now, the definition of an artifact only describes *which* actions can be executed and policies, goal states, and compliance rules constrained *when* the execution may occur — the link to the agent *who* may actually execute the action at runtime was only implicitly given. For instance, we assumed that the "submit" action of the claim artifact is to be executed by an agent "customer", but this mapping was not part of the artifact definition, cf. Def. 2. In this section, we shall make this assignment to agents explicit and establish a *role-based access control* (RBAC). That is, we pool agents to roles and annotate each action with a role to express who may choose to execute this action. The execution of the business process then yields an execution schedule; that is, a mapping from actions to executing agents.

By extending our artifact model, we can reason about the execution privileges and are able to express another class of compliance rules specifying *separation of duties* and *restricted access* constraints. Such execution constraints limit the choice of agents who may execute an action at runtime. A typical example for the separation of duties is the well-known *four-eyes principle* which demands that certain transactions need to be approved by both the CEO and the CFO of a company rather than a single chief officer.

6.1. Modeling role-based access control

We begin by adding agents and roles to our formal model. Let $\mathcal{A}$ be a set of agents (i. e., individuals that may participate in the execution of the business process) and $\mathcal{R}$ be a set of role names. A *role assignment* (denoted with ra) assigns a set of agents to each role name: $ra : \mathcal{R} \to 2^{\mathcal{A}}$. Note that roles may overlap; that is, one agent may be member of several roles. Consequently, hierarchy can be expressed indirectly (e. g., explicitly defining a role "managers" and "employers" where each manager is also an employer) rather than to add relations between roles. The rationale behind this is to work with a simple formal model — a proper proper high-level modeling language would provide syntactical means to express hierarchy in a natural way.

Next, we formalize the role-based access control which links actions with roles.

Definition 5 (Role-based access control, schedule). A *role-based access control* assigns to each controllable action a role name; $rbac : \mathcal{L}_c \to \mathcal{R}$. A *schedule* assigns to each controllable action an executing agent; $s : \mathcal{L}_c \to \mathcal{A}$. A schedule s *conforms to* a role-based access control $rbac$ iff $s(\ell) \in ra(rbac(\ell))$, for all $\ell \in \mathcal{L}_c$. Denote with $\mathcal{S}_{rbac}$ all conforming schedules for $rbac$.

Note that roles are only assigned to controllable actions $\mathcal{L}_c$, because the execution of uncontrollable actions $\mathcal{L}_u$ is out of scope of the business process and hence cannot be constrained. The role-based access control constrains which agent may choose to execute which action at runtime.

Running example (cont.). We refine the running example by specifying six agents ($\mathcal{A} = \{\mathit{ins}_1, \mathit{ins}_2, \mathit{customer}, \mathit{controller}, \mathit{fin}_1, \mathit{fin}_2\}$) and four role names ($\mathcal{R} = \{\mathit{ins}, \mathit{cus}, \mathit{con}, \mathit{fin}\}$) with the role assignment ra:

$$\begin{aligned} ra(\mathit{ins}) &= \{\mathit{ins}_1, \mathit{ins}_2\} \\ ra(\mathit{cus}) &= \{\mathit{customer}\} \\ ra(\mathit{con}) &= \{\mathit{controller}\} \\ ra(\mathit{fin}) &= \{\mathit{fin}_1, \mathit{fin}_2\} \end{aligned}$$

We set the role-based access control according to Fig. 3; that is, $\mathit{rbac}(\text{submit}) = \mathit{cus}$, $\mathit{rbac}(\text{prepare}) = \mathit{ins}$, etc.

After all choices are made, the business process can be executed and, for each controllable action, a single executing agent is scheduled. Formally, such a *schedule* is a mapping similar to the role-based access control, but having agents as codomain rather than role names. Conceptually, a schedule implements the role-based access control by choosing at runtime, for each action, which agent from the assigned role actually executes it. As motivated earlier, not every schedule that conforms the role-based access control may be desirable, be it for legal regulations or for efficiency reasons. To this end, *execution constraints* further confine the choice which agent may execute which action. Formally, an execution constraint is a subset of the conforming schedules $\mathcal{S}_{rbac}$.

Running example (cont.). To exemplify the shape of execution constraints, consider again the insurance claim handling. There, we might want to express constraints to rule out certain schedules that would violate legal requirements or internal regulations.

EC1 *To avoid frauds without excessive involvement of financial controllers, the actions "create" and "payPart" must not be performed by the same financial officer.*

EC2 *The actions "prepare" and "requestInfo" must be performed by the same insurer to streamline the process and to ensure information economy.*

The execution constraints bind certain agents to actions. For instance, constraint EC2 states that after deciding that the insurance employer ins_1 shall execute the "prepare" action, it is required that he must also execute the "requestInfo" action if required, whereas ins_2 is excluded from this action. Formally, the constraints exclude noncompliant schedules such as

- $s(\text{create}) = \mathit{fin}_1$ and $s(\text{payPart}) = \mathit{fin}_1$,
- $s(\text{create}) = \mathit{fin}_2$ and $s(\text{payPart}) = \mathit{fin}_2$,

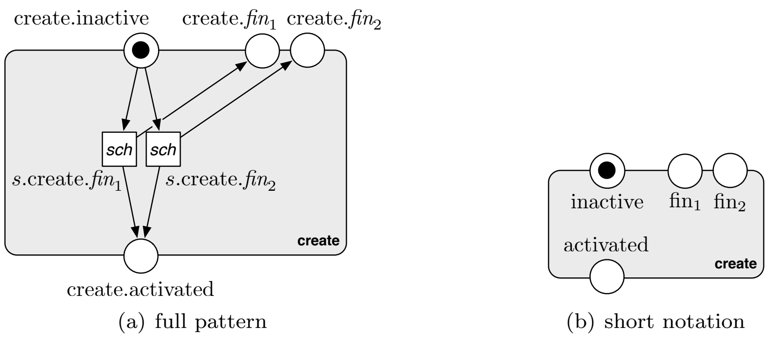


Figure 10: Constraint patterns with $\Omega = \{[\text{create}.\mathit{fin}_1], [\text{create}.\mathit{fin}_2]\}$

- $s(\text{prepare}) = \mathit{ins}_1$ and $s(\text{requestInfo}) = \mathit{ins}_2$, and
- $s(\text{prepare}) = \mathit{ins}_2$ and $s(\text{requestInfo}) = \mathit{ins}_1$.

In the remainder of this section, we shall extend our framework to not only synthesize correct and compliant behavior, but also to generate all compliant schedules; that is, the result of the synthesis is a business process together with a plan which agent may execute which action.

6.2. Integration of execution constraints into the process synthesis

To integrate execution constraints into the process synthesis, we need to extend our Petri net model such that only compliant schedules yield weak termination. We model the role-based access control by a small Petri net pattern that is added to the respective artifacts. Such a *schedule pattern* then only enables the action after an agent from the assigned role is scheduled to execute the action. Additionally, the schedule can be evaluated with respect to the execution constraints.

An example for such a schedule pattern is depicted in Fig. 10(a). This pattern models the scheduling for the label “create”: The such labeled actions may be executed by any agent of the role *fin* (i. e., fin_1 or fin_2). The initially marked place “create.inactive” models a state in which no schedule has been made for this label yet. Such an schedule can be made by the transitions labeled “s.create.fin_1” and “s.create.fin_2”. Firing one of these transition makes the choice explicit by marking one of the places “create.fin_1” or “create.fin_2” modeling a schedule s with $s(\text{create}) = \mathit{fin}_1$ or $s(\text{create}) = \mathit{fin}_2$, respectively. Furthermore, the place “create.activated” is marked to enable the label “create”. Note that the execution of the scheduling transitions is not part of the actual business process; that is, no agent can be scheduled for the execution. Instead, we assume a virtual agent “*sch*” (for “scheduler”) who is responsible for the compliant scheduling. We further assume that any marking that marks either “create.fin_1” or “$\mathit{create}.\mathit{fin}_2$” is a final marking of the pattern. This ensures that only conforming schedules are taken into account. This pattern can be

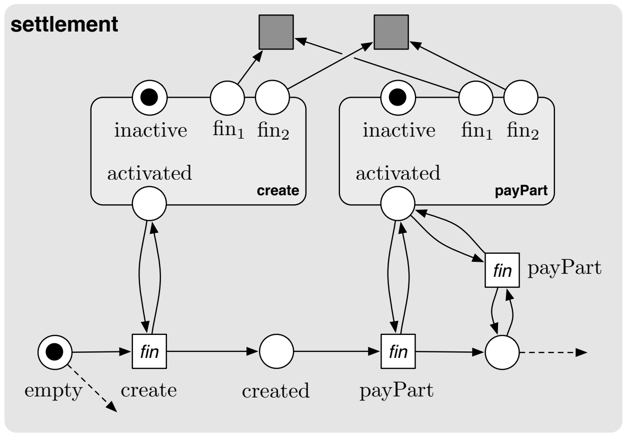


Figure 11: Extending the settlement artifact to implement the execution constraint EC1

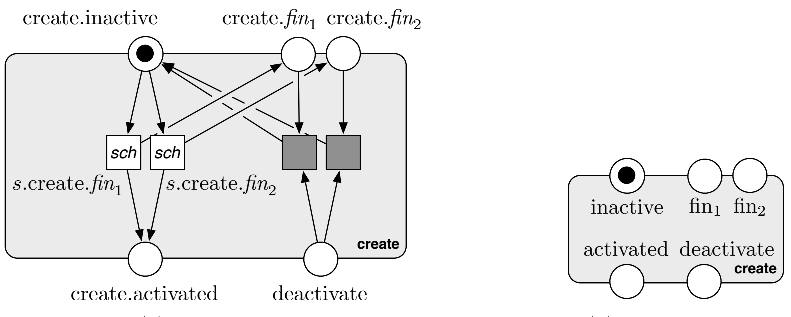
canonically extended for any number of agents. To make the following graphics more readable, we further use a shorthand notation (cf. Fig. 10(b)) which only shows the places of the pattern and does not use the label prefix as it is clear from the inscription of the constraint pattern.

Figure 11 shows how two constraint patterns are integrated into the settlement artifact to implement the execution constraint EC1 of the running example. Beside making the "activated" places a prerequisite for the "create" and the "payPart" labeled transitions, also two gray (viz. uncontrollable) transitions are added which are enabled in case of illegal agent assignments, namely $[\mathit{fin}_1, \mathit{fin}_1]$ and $[\mathit{fin}_2, \mathit{fin}_2]$. If either of this combination is reached due to an illegal assignment, firing one of the transitions makes a final marking of the schedule patterns unreachable. As a result, weak termination can only be guaranteed by avoiding illegal assignments, viz. a synthesis result guarantees that all execution constraints are met.

6.3. Extension to repetitive behavior

The patterns of Fig. 10 have the property that once a schedule is made, it is fixed for the whole business process execution. That is, if an action is executed several times (e. g., the "payPart" action in the settlement artifact of Fig. 11), it is always executed by the same agent. To add more flexibility, it might be desirable to *scope* the constraints to smaller portions of the process and to reschedule the executing agent of an action during the execution of a process. To be able to reason about repeated execution of an action, we assume structured loops (e. g., a while or repeat-until loop) where a unique "enter loop" and "leave loop" action can be identified.

In case of such structured loops, we extend our constraint pattern of Fig. 10(a) with additional deactivating transitions, cf. Fig. 12(a). These uncontrollable transitions are enabled by marking place "deactivate" and ensure that the schedule is eventually reset and the action is deactivated. As a result, the



(a) full pattern (b) short notation

Figure 12: Constraint patterns for repetitive behavior

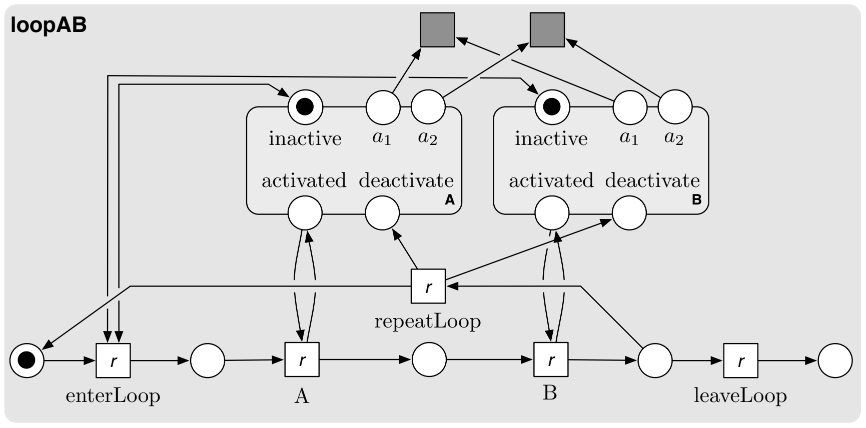


Figure 13: Example for a looped four-eyes principle

constraint pattern eventually reaches its initial state and can be rescheduled. Again, we also provide a shorthand notation, cf. Fig. 12(b).

As an example, consider the artifact depicted in Fig. 13. It models a loop in which a four-eyes principle is demanded for each loop execution. The loop is entered by firing transition "enterLoop" which requires all schedule patterns to be in the state "inactive". Then, the agents for action "A" and "B" need to be scheduled. After executing "B", the loop can be left by firing "leaveLoop" or repeated by firing "repeatLoop". Firing the latter transition marks all "deactivate" places of the enclosed schedule patterns. As a consequence, the schedule of label "A" and "B" is reset and the loop can be reentered. Now, the choice which agent executes which action can be made independently in each iteration.

6.4. Discussion

By extending our artifact model with a role-based access control, we can express a new class of constraints that focus on which agent may execute which action. Classical examples of such constraints are the four-eyes principle or other separation of duties or restricted access constraints. Whereas the distribution of agents to actions can also be expressed as a constraint problem that can be solved using SAT solvers, our model also allows to link the access control with concrete states and locations of artifacts. Such constraints (e. g., "Prototypes may only be demonstrated by engineers at the company site, whereas managers are allowed to demonstrate them off-site.") mix schedules with artifact states and cannot be expressed in terms of a constraint problem. Consequently, our approach offer more dynamic constraints.

The process synthesis (cf. Sect. 5) of a model with role-based access control consists of two parts: (1) an operational business process that follows all goal states, policies, and compliance rules and (2) a description of all compliant schedules. The latter is expressed in terms of sequences of scheduling transitions such as "$s.\mathrm{create}.\mathit{fin}_1$" (cf. Fig. 10(a)). In the raw synthesis result, however, these actions are intertwined. To separate the operational business process from the schedules, undesired action labels can be projected.

Finally, we would like to note that the presented Petri net constraint patterns are only a formalization of the presented concepts rather than a modeling language. Similar to the compliance rules of Sect. 4, a concrete modeling language should offer a list of frequently used scenarios for which concrete translations are provided.

7. Related work

Compliance has received a lot of attention in the business process management community. Wheres classical approaches on artifact-centric models are often formalized using infinite state systems (cf. [30, 31]), our approach heavily relies synthesis and hence on finite state models. We hence classify contributions related to our approach as follows.

Compliance by detection. Awad et al. [25] investigate a pattern-based compliance check based on BPMN-Q [8]. They also cover the compliance rule classes defined by Dwyer et al. [23] and give a CTL formalization as well as an antipattern for each rule. These antipatterns are used to highlight the compliance violations in a BPMN model. Such a visualization is very valuable for the process designer and it would be interesting to see whether such antipatterns are also applicable to the artifact-centric approach. Sadiq et al. [32] use a declarative specification of compliance rules from which they derive compliance checks. These checks are then annotated to a business process and monitored during its execution. These checks are similar to the nonblocking compliance rule models that only monitor behavior rather than constraining it. Lu et al. [3] compare business processes with compliance rules and derive a compliance degree. This is an interesting approach, because it replaces yes/no answers by numeric values which could help to easier diagnose noncompliance. Knuplesch et al. [33] analyze data aspects of operational business process models. Similar to the artifact-centric approach, data values are abstracted into compact life cycles.

Compliance by design. Goedertier and Vanthienen [26] introduce the declarative language PENELOPE to specify compliance rules. From these rules, a state space and a BPMN model is generated which is compliant by design. This approach is limited to acyclic process models. Furthermore, the purpose of the generated model is rather the validation of the specified rules than the execution. Küster et al. [34] study the interplay between control flow models and object life cycles. The authors present an algorithm to automatically derive a sound process model from given object life cycles. The framework is, however, not designed to express dependencies between life cycles and therefore cannot specify complex policies or compliance rules.

To the best of knowledge, this paper presents the first approach that generates compliant and operational business process models from declarative specifications of artifact life cycles, policies, and compliance rules. Note this paper is an extension of an earlier extended abstract [1]. Compared to that abstract, this paper introduced role-based access control, cf. Sect. 6.

8. Conclusion

Summary. We presented an approach to automatically construct business process models that are compliant by design. The approach follows an artifact-centric modeling style in which business processes are specified from the point of view of the involved data objects. We showed how artifact-centric business processes can be canonically extended to also take compliance rules into account. These rules can express constraints on the execution of actions, but can also take data and location information into account. By composing compliance rules to the artifact-centric model, we could reduce the check for compliant behavior to the reachability of final states. This reduction could also be used to model a role-based access control to constrain which agent is allowed to execute which action, allowing more involved constrains such as the four-eyes principle. Consequently, we could use existing synthesis algorithms and tools to automatically generate compliant business process models. In case of noncompliance, we further sketched diagnosis information that can be used to visualize the reasons that make compliance rules unrealizable.

Lessons learnt. With Petri nets, we can use a single formalism to model artifact life cycles, interartifact dependencies, compliance rules, and a role-based access control. Only this unified way of modeling enabled us to approach the compliance-by-design approach with only a few concepts (namely artifacts, composition, and partner synthesis). In addition, it is notable that the compliance rule models can also be used to check operational business process models for compliance: By composing compliance rules to existing Petri net models, we reduced the compliance check by a check for weak termination and allows to use standard verification tools [35].

Future work. We see numerous directions to continue the work in the area of compliance by design. The main practical limitation is the lack of a proper modeling language, because the presented Petri net formalization is only a conceptual modeling language. We recently developed an extension [36] for BPMN [13] to provide a graphical notation that is more accessible for domain experts to model artifacts, policies, and compliance rules. A canonic next step would then be the integration of the approach into a modeling tool and an empirical evaluation thereof. Furthermore, we concentrated on the early design of a business process and did not consider its execution. For business processes that are already in execution, our approach is currently not applicable as the translations from conceptual models to industrial languages are still very immature [15].

Another aspect that needs to be addressed is the expressiveness of the artifacts and compliance rules. Of great interest are *instances*. To model more involved scenarios, it is crucial distinguish several instances of an artifact. We currently assume that for each artifact only a single instance exists and that the number of agents for each role is fixed. Interesting research questions would deal with the minimal number of agents that need to be distinguished to realize a compliant process execution.

References

[1] N. Lohmann, Compliance by design for artifact-centric business processes, in: BPM 2011, LNCS 6896, Springer, 2011, pp. 99–115.

[2] J. C. Cannon, M. Byers, Compliance deconstructed, ACM Queue 4 (7) (2006) 30–37.

[3] R. Lu, S. W. Sadiq, G. Governatori, Compliance aware business process design, in: BPM 2007 Workshops, LNCS 4928, Springer, 2007, pp. 120–131.

[4] M. Ben-Ari, Z. Manna, A. Pnueli, The temporal logic of branching time, in: POPL '81, ACM, 1981, pp. 164–176.

[5] A. Pnueli, The temporal logic of programs, in: FOCS 1977, IEEE, 1977, pp. 46–57.

[6] A. Pnueli, In transition from global to modular temporal reasoning about programs, in: Logics and models of concurrent systems, volume F-13 of NATO Advanced Summer Institutes, Springer, 1985, pp. 123–144.

[7] W. M. P. v. d. Aalst, M. Pesic, DecSerFlow: Towards a truly declarative service flow language, in: WS-FM 2006, LNCS 4184, Springer, 2006, pp. 1–23.

[8] A. Awad, BPMN-Q: a language to query business processes, in: EMISA 2007, LNI P-119, GI, 2007, pp. 115–128.

[9] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, MIT Press, 1999.

[10] S. Sackmann, M. Kähmer, M. Gilliot, L. Lowis, A classification model for automating compliance, in: CEC/EEE 2008, IEEE, 2008, pp. 79–86.

[11] N. Lohmann, K. Wolf, Artifact-centric choreographies, in: ICSOC 2010, LNCS 6470, Springer, 2010, pp. 32–46.

[12] K. Ryndina, J. M. Küster, H. Gall, Consistency of business process models and object life cycles, in: MoDELS Workshops, LNCS 4364, Springer, 2006, pp. 80–90.

[13] OMG, Business Process Model and Notation (BPMN), Version 2.0, Object Management Group (2011).
URL `http://www.omg.org/spec/BPMN/2.0`

[14] A. Alves, et al., Web Services Business Process Execution Language Version 2.0, OASIS Standard, OASIS (Apr. 2007).
URL `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`

[15] N. Lohmann, J. Kleine, Fully-automatic translation of open workflow net models into simple abstract BPEL processes, in: Modellierung 2008, Vol. P-127 of Lecture Notes in Informatics (LNI), GI, 2008, pp. 57–72.

[16] W. M. P. v. d. Aalst, The application of Petri nets to workflow management, Journal of Circuits, Systems and Computers 8 (1) (1998) 21–66.

[17] W. Reisig, Petri Nets, EATCS Monographs on Theoretical Computer Science Edition, Springer, 1985.

[18] P. Ramadge, W. Wonham, Supervisory control of a class of discrete event processes, SIAM J. Control Optim. 25 (1) (1987) 206–230.

[19] K. Wolf, Does my service have partners?, LNCS ToPNoC 5460 (II) (2009) 152–171.

[20] N. Lohmann, D. Weinberg, Wendy: A tool to synthesize partners for services, in: PETRI NETS 2010, LNCS 6128, Springer, 2010, pp. 297–307, tool available at `http://service-technology.org/wendy`.

[21] N. Lohmann, P. Massuthe, K. Wolf, Behavioral constraints for services, in: BPM 2007, LNCS 4714, Springer, 2007, pp. 271–287.

[22] N. Lohmann, S. Mennicke, C. Sura, The Petri Net API: A collection of Petri net-related functions, in: APWN 2010, CEUR Workshop Proceedings 643, CEUR-WS.org, 2010, pp. 148–155.

[23] M. B. Dwyer, G. S. Avrunin, J. C. Corbett, Patterns in property specifications for finite-state verification, in: ICSE 1999, IEEE, 1999, pp. 411–420.

[24] K. Havelund, G. Roşu, Testing linear temporal logic formulae on finite execution traces, Technical Report 01.08, RIACS (2001).

[25] A. Awad, M. Weidlich, M. Weske, Visually specifying compliance rules and explaining their violations for business processes, J. Vis. Lang. Comput. 22 (1) (2011) 30–55.

[26] S. Goedertier, J. Vanthienen, Designing compliant business processes with obligations and permissions, in: BPM Workshops 2006, LNCS 4103, Springer, 2006, pp. 5–14.

[27] P. Massuthe, A. Serebrenik, N. Sidorova, K. Wolf, Can I find a partner? Undecidablity of partner existence for open nets, Inf. Process. Lett. 108 (6) (2008) 374–378.

[28] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers, Trans. Inf. and Syst. E80-D (3) (1997) 315–325.

[29] N. Lohmann, Why does my service have no partners?, in: WS-FM 2008, LNCS 5387, Springer, 2009, pp. 191–206.

[30] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, R. Vaculín, Introducing the guard-stage-milestone approach for specifying business entity lifecycles, in: WS-FM 2010, LNCS 6551, Springer-Verlag, 2011, pp. 1–24.

[31] B. B. Hariri, D. Calvanese, G. D. Giacomo, R. D. Masellis, P. Felli, Foundations of relational artifacts verification, in: BPM 2011, LNCS 6896, Springer, 2011, pp. 379–395.

[32] S. W. Sadiq, G. Governatori, K. Namiri, Modeling control objectives for business process compliance, in: BPM 2007, LNCS 4714, Springer, 2007, pp. 149–164.

[33] D. Knuplesch, L. T. Ly, S. Rinderle-Ma, H. Pfeifer, P. Dadam, On enabling data-aware compliance checking of business process models, in: ER 2010, LNCS 6412, Springer, 2010, pp. 332–346.

[34] J. M. Küster, K. Ryndina, H. Gall, Generation of business process models for object life cycle compliance, in: BPM 2007, LNCS 4714, Springer, 2007, pp. 165–181.

[35] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, K. Wolf, Instantaneous soundness checking of industrial business process models, in: BPM 2009, LNCS 5701, Springer, 2009, pp. 278–293.

[36] N. Lohmann, M. Nyolt, Artifact-centric modeling using BPMN, in: ICSOC 2011 Workshops, Vol. 7221 of LNCS, Springer, 2012, pp. 54–65.