Design and Verification of Instantiable Compliance Rule Graphs in Process-Aware Information Systems[⋆]

Linh Thao Ly[1], Stefanie Rinderle-Ma[2], and Peter Dadam[1]

[1] Institute of Databases and Information Systems
Ulm University, Germany
[2] Faculty of Computer Science
University of Vienna, Austria
{thao.ly,peter.dadam}@uni-ulm.de, stefanie.rinderle-ma@univie.ac.at

Abstract. For enterprises it has become crucial to check compliance of their business processes with certain rules such as medical guidelines or financial regulations. When automating compliance checks on process models, existing approaches have mainly addressed *process-specific* compliance rules so far, i.e., rules that correspond to a particular process model. However, in practice, we will rather find *process-independent* compliance rules that are nevertheless to be checked over process models. Thus, in this paper, we present an approach that enables the *instantiation* and verification of process-independent compliance rules over process models using domain models. For this, we provide an intuitive visualization of compliance rules and compliance rule instances at user level and show how rules and instances can be formalized and verified at system level. The overall approach is validated by a pattern-based comparison to existing approaches and by means of a prototypical implementation.

1 Introduction

In many application domains, business processes are subject to compliance rules and policies that stem from domain specific requirements such as standardization or legal regulations [1]. Ensuring compliance of their business processes is crucial for enterprises nowadays, particularly since auditing and certification has become a competitive edge in many domains. Examples include certified family-friendly enterprises being more attractive to prospective employees or clinics proving a certain standard of their audited treatments to patients. Since process models are the common way to represent business processes, business process compliance can be ensured by verifying process models to be installed in process-aware information systems against imposed compliance rules. Tab. 1 summarizes quality compliance rules imposed on the software development process depicted in Fig. 1.

[⋆] This work was done within the SeaFlows project, which is funded by the German Research Foundation (DFG).

Table 1. Examples of compliance rules for software development

c_1	Goals have to be defined before starting the development.
c_2	Each test activity has to be documented.
c_3	After the development freeze, no further development activities shall take place.
c_4	Before carrying out a second usability test, a necessity check after the first usability test is necessary.
c_5	The testing has to be followed by an approval and the integration. Additionally, no changes shall take place between the approval and the integration.

So far, existing approaches mainly focus on checking *process-specific* compliance rules that correspond to certain process models [1–3]. One example is rule c_1 (cf. Tab. 1) over process model P (cf. Fig. 1) referring to process activities `define goals` and `start of development phase`. However, in practice, compliance rules are often specified in a more general manner and not specifically in correspondence with a particular process model. Rule c_2, for example, refers to a general `test` activity and not to a `functional test` as present in the software development process depicted in Fig. 1. Nevertheless, for quality assurance, it could be desired to verify c_2 over the development process. Thus, in order to support a broad spectrum of realistic compliance rules, we must enable the verification of both, process-specific and process-independent compliance rules over process models.

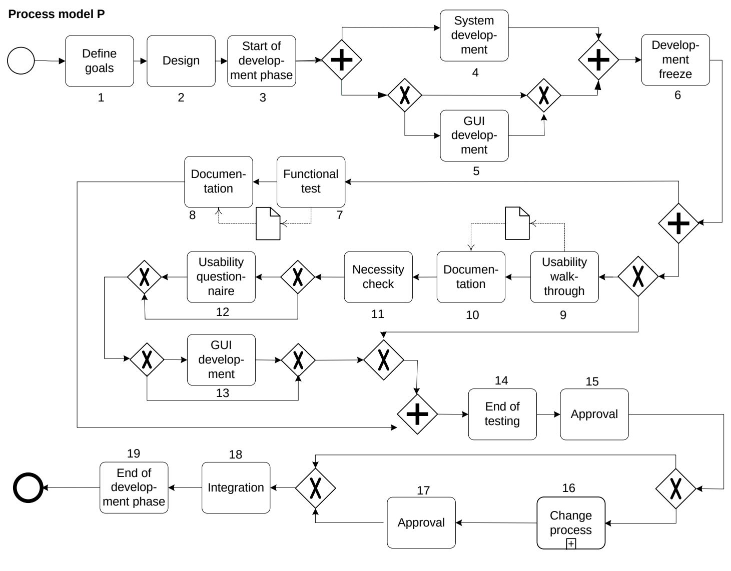

Fig. 1. Abstracted software development process (in BPMN notation)

Connected with supporting process-independent compliance rules, it is also necessary to enable the definition of compliance rules at different *design levels*. At user level an intuitive, high-level representation of compliance rules is desirable. Contrary, at system level, it must be possible to conduct automatic verification of process models against compliance rules. For this, it is necessary to *instantiate* compliance rules over a particular process model and to equip them with formal semantics for later verification.

In this paper, we present an approach for checking process-independent compliance rules over process models. This is achieved by instantiating these compliance rules over a particular process model using a domain model. The resulting compliance rule instances can be verified separated from each other. This enables individual compliance reports as well as individual solutions in case of compliance violations. Furthermore, we show how compliance rules can be defined at different design levels. At user level, compliance rules are specified using an intuitive visualization based on graph structures. At system level compliance rule graphs are equipped with formal semantics based on First Order Logic and execution traces. The latter guarantees for independence of a particular process meta model. The overall approach is validated by a pattern-based comparison to existing approaches, by a general discussion on the co-existence of compliance rules and process models, and by means of our prototypical implementation.

Sect. 2 discusses the compliance rule instantiation approach. Sect. 3 focuses on the design of instantiable compliance rule graphs. Their counterpart formalization is presented in Sect. 4. A validation of our approach is provided in Sect. 5. We close with a related work discussion in Sect. 6 and a summary in Sect. 7.

2 Compliance Rule Instantiation

Fig. 1 depicts the process model of a slightly abstracted software development process that we discovered within several practical student projects at Ulm University. As discussed in Sect. 1, several compliance rules might be imposed on the development process for quality and efficiency reasons (cf. Tab. 1). The basic challenge is now to check whether the process complies to these compliance rules or not. Additionally, in case of violations the system should yield helpful user feedback in precisely reporting the reason for the problem.

Compliance rules are generally defined at different abstraction levels ranging from rather abstract business policies to specific definitions [1]. To support different levels of abstraction for compliance rules, we enable the use of domain models. Based on these, compliance rules can be instantiated for certain processes. Note that domain models are demanded in many applications [4]. Moreover, we may also benefit from existing ontologies such as in the healthcare domain [5]. Take for example compliance rules c_2 and c_4 (cf. Tab. 1): both refer to test activities, where c_2 is general and c_4 more specific (`usability test`). Fig. 2 shows an extract of the domain model belonging to the development process. A test can be more specifically modeled as a `usability` or `functional test`, where `usability test` can additionally distinguished into `usability`

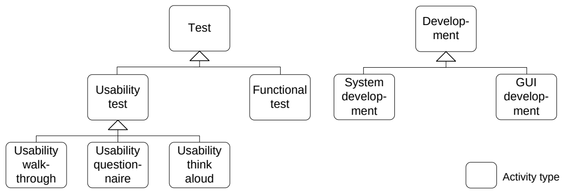

Fig. 2. An extract of the software development domain model

`walkthroughs`, `usability questionnaires`, or `usability think aloud`. A `development` might be either a `system development` or a `GUI development`.

Without any further knowledge, compliance rule c_2 cannot be evaluated over development process P since P does not contain any test activity. In fact, c_2 can only be evaluated by *instantiation* over P using the corresponding domain model as depicted in Fig. 2. Based on the domain model, the general test activity referred to by c_2 can be instantiated by three more specific testing activities contained within P (i.e., `usability walkthrough`, `usability questionnaire`, and `functional test`). This results in three *compliance rule instances* c_{2_1}, c_{2_2}, and c_{2_3} derived from c_2 (cf. Tab. 2). The `usability walkthrough` and the `functional test` are both documented according to the control flow of P. Thus, c_{2_1} and c_{2_2} are satisfied. However, the `usability questionnaire` is not documented within P what violates instance c_{2_3}.

Table 2. Compliance rule instances for c2, cf. Tab. 1

c_{2_1}	The `functional test` has to be documented.
c_{2_2}	The `usability walkthrough` has to be documented.
c_{2_3}	The `usability questionnaire` has to be documented.

We see that by using instantiation the verification of c_2 over P becomes possible. However, which benefits are offered by maintaining c_2 and instantiating it "on demand" instead of replacing c_2 by its instances c_{2_1}, c_{2_2}, and c_{2_3} for P and the corresponding domain model? The different advantages become evident when looking at the modeling, maintenance, and evolution of compliance rules on the one side and compliance checking for process models on the other side as depicted in Fig. 3.

First of all, for more complex domain models and processes, without instantiation, the number of compliance rules might dramatically increase resulting in huge effort for compliance rule modeling and maintenance. Moreover, supporting high-level compliance rules and compliance rule instantiation eases the evolution of compliance rules. Imagine, for example, that compliance rule c_2 has

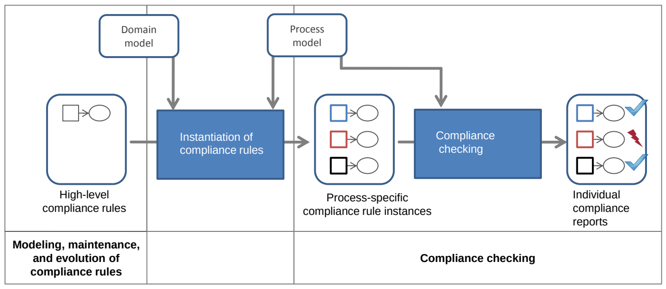
Domain model
Process model
High-level compliance rules
Instantiation of compliance rules
Process-specific compliance rule instances
Compliance checking
Individual compliance reports
Modeling, maintenance, and evolution of compliance rules
Compliance checking

Fig. 3. The SeaFlows approach: compliance rule instantiation

to be adapted (e.g., due to changes in the quality policies) such that each test activity not only has to be documented but also has to be approved. In this case, if the strategy to explicitly model all compliance rule instances of c_2 was applied, ${c_2}_1$, ${c_2}_2$, and ${c_2}_3$ would have to be individually modified in order to integrate the change. By contrast, following the approach proposed in this paper, only high-level rule c_2 has to be modified resulting in modified high-level rule c'_2. c'_2 then can be reinstantiated and checked for relevant process models.

Moreover, checking compliance at instance level results in fine-granule feedback on individual violations of rule instance (cf. Fig. 3). In turn, fine-granule feedback enables fine-granule treatment of compliance rule instance violations. Let us assume, for example, that c_2 is only of recommendation nature (i.e., enforcement level low). Since ${c_2}_3$ is violated over process model P, the process designer might decide to completey ignore c_2. However, since ${c_2}_1$ and ${c_2}_2$ are actually fulfilled, he might prefer to ignore ${c_2}_3$ instead of c_2.

Altogether, using compliance rule instantiation as proposed in this paper, we achieve minimal effort for compliance rule modeling and maintenance on the one hand, but enable individual compliance checks and corresponding reports for compliance rule instances on the other hand.

3 Instantiable Compliance Rule Graphs

The next challenge is to design instantiable compliance rules in a way that they can be easily understood by users. This task includes representation of (high-level) compliance rules as well as of compliance rule instances. We found that at both levels, graphs provide an intuitive visualization that can be equipped with formal semantics and verified at system level later on (cf. Sect. 4).

Process-independent Compliance Rules To support process-independent (i.e., high-level) compliance rule graphs and as well as their instantiation over particular process models, a data model as depicted in Fig. 4 is needed: a process model consists of a set of nodes distinguished by their node id to which activities are assigned. An activity is assigned to activity types. An activity type may be the sub-type of another activity. For example, the activity type `usability walkthrough` is a sub-type of the activity type `usability test` (cf. domain model in Fig. 2).

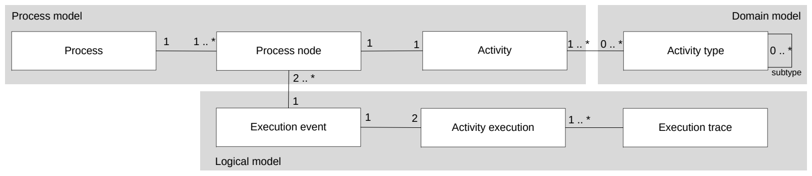

Fig. 4. The data model for instantiable compliance rules.

When looking at compliance rules c_1 to c_5 (cf. Tab. 1), it can be observed that compliance rules mostly reflect certain patterns. Typically, compliance rules require the occurrence and/or the absence of certain activities (*occurrence/absence* patterns). For c_1, for example, activity `define goals` should occur before starting the `development`. By contrast, for c_3 further `development` activities should be absent if activity `development freeze` has taken place. Furthermore, it can be observed that the occurrence or absence of certain activities is often conditional upon the occurrence or absence of other activities. Thus a compliance rule mostly comprises a triggering part (denoted as *antecedent* pattern) and a *consequence* pattern. To be able to instantiate process-inpdedendent compliance rule graphs later on, each node of a rule graph is associated with an activity type from the domain model (cf. Fig. 4). Directed edges connecting the nodes represent predecessor relations.

Based on these observations compliance rules c_1 to c_5 can be described by compliance rule graphs as depicted in Fig. 5. Compliance rule graph c_1, for example, states that the execution of activity `start of development process` has to be directly or indirectly preceded by the execution of the activity `define goals`. Compliance rule graph c_5 states that the execution of activity `end of testing` has to be succeeded by an execution of `approval` and `integration`. Between these two executions, however, no execution of the `change process` must occur. Note that compliance rule graphs might also contain data information as data object `Test` for c_2. How to integrate and evaluate such data information into compliance rules is subject to our future research.

Process-specific Instantiation of Compliance Rules Generally, it must be possible to define compliance rule graphs independent of a particular process model. For later verification, however, it is necessary to instantiate independent compliance rule graphs over process models.

The instantiation of compliance rule graphs is accomplished by "binding" antecedent compliance rule nodes to nodes of the process model (reflected by their node ids) with the associated activity type. Instantiation of compliance

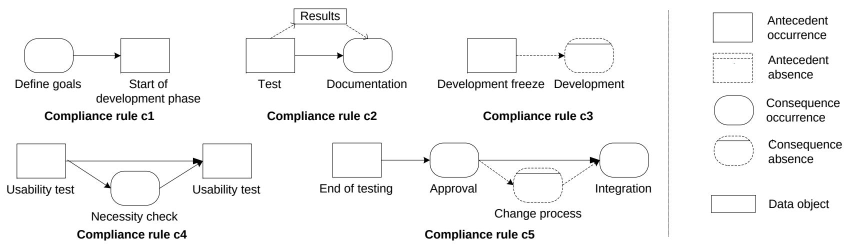

Fig. 5. Process-independent compliance rule graphs

rule c_2 (cf. Fig. 5) over process model P (cf. Fig. 1) using the domain model as depicted in Fig. 2, for example, results in three compliance rule instances as depicted in Fig. 6. Visualizing compliance rules that way, the user is able to locate exactly, which occurrences of a compliance rule are relevant, which are satisfied, and which are violated.

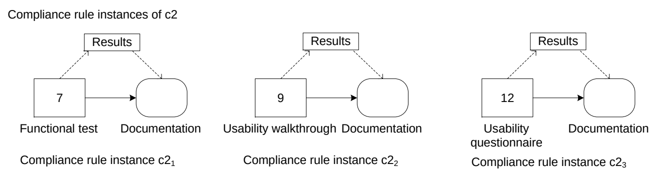

Fig. 6. Process-specific compliance rule instances

4 Formalization and Verification of Compliance Rules

The graphical compliance rule notation provides for more intuitive modeling of compliance rules since it hides formal details from the user. However, as we will show in this section, each compliance rule graph corresponds to a logical formula with defined semantics. The latter is necessary to enable the verification of processes against imposed compliance rules.

4.1 On Formalizing Compliance Rules

We opted for first-order predicate logic (FOL) to formalize compliance rules for several reasons. The use of a general and expressive logic such as FOL enables the extension of our approach in order to support further types of compliance rules not supported so far (e.g., authorization or organizational model compliance rules). Moreover, FOL allows for the definition of the antecedent-consequence-structure of compliance rules in a straight-forward and elegant manner. The nodes of compliance rule graphs are mapped to variables while properties of compliance rule graph nodes and the relations between nodes are mapped to corresponding predicates in a FOL formula. Due to lack of space we abstain from a complete definition here, but rather informally describe the structure in the following. The general structure of a compliance rule is as follows.

Structure 1 (Compliance rule) *Let $A_{\mathcal{T}}$ be the set of activity types of the domain. Then, a compliance rule is of the following form:*

$[true \mid antecedentOccurrencePatterns \mid antecedentAbsencePatterns \mid$
$antecedentOccurrencePatterns \wedge antecedentAbsencePatterns]$
$\rightarrow$
$consequence \vee consequence \vee \cdots \vee consequence$

$consequence :=$
$[consequenceOccurrencePatterns \mid consequenceAbsencePatterns \mid$
$consequenceOccurrencePatterns \wedge consequenceAbsencePatterns]$

In Struct. 1 the antecedent is either empty (i.e., the compliance rule is always activated) or consists of an *antecedent pattern*. The latter can be composed from occurrence patterns defining the occurrences of activity executions that activate the compliance rule. Compliance rule c_2 (cf. Fig. 5), for example, is activated by the occurrence of an activity execution associated to the activity type `test` while compliance rule c_4 is activated by a more complex occurrence pattern (namely the sequence of two activity executions). The antecedent pattern may also consist of absence patterns defining the absence of particular activity executions. This allows for refining the occurrence pattern by putting additional conditions on the absence of activity executions (for example, to express patterns such as "if no approval takes place between the end of development and the integration"). If the antecedent of a compliance rule applies, one of the rule's *consequence patterns* must also apply in order to satisfy the rule. Each consequence pattern, in turn, may consists of occurrence as well as absence patterns and corresponding relations. Compliance rule c_3 (cf. Fig. 5), for example, has a consequence absence pattern in its consequence part while the consequence part of compliance rule c5 is composed from both consequence occurrence and consequence absence patterns.

The formula for compliance rule c_2 is given below. It expresses that each *activity execution* associated to the activity type `test` has to be followed by an activity execution associated to the activity type `documentation` with the same `results` data object. This is a process-independent compliance rule, since it only references to activity types from the domain model (cf. Fig. 2).

Compliance rule c2:
$\forall t(ActivityType(t, \texttt{test}) \rightarrow \exists d : (ActivityType(d, \texttt{documentation}) \wedge Pred(t, d) \wedge results(t) = results(d)))$

Based on the development process (cf. Fig. 1), we can identify the process nodes that are associated to the activity types referenced in the formula (namely `test` and `documentation`). The formula for c_2 can be adapted accordingly by refering to activity executions associated to particular nodes in the process.

Process-specific compliance rule $\mathbf{c}_2$:
$\forall t(ProcessNode(t, \mathtt{7}) \vee ProcessNode(t, \mathtt{9}) \vee ProcessNode(t, \mathtt{12}) \rightarrow$
$\exists d : (ActivityType(d, \mathtt{documentation}) \wedge Pred(t, d) \wedge results(t) = results(d)))$

Based on this, we obtain the resulting compliance rule instances of c2 as follows.

$\mathbf{c}_{2_1}$**:** $ProcessNode(t, \mathtt{7}) \rightarrow \exists d : (ActivityType(d, \mathtt{documentation}) \wedge Pred(t, d) \wedge results(t) = results(d))$
$\mathbf{c}_{2_2}$**:** $ProcessNode(t, \mathtt{9}) \rightarrow \exists d : (ActivityType(d, \mathtt{documentation}) \wedge Pred(t, d) \wedge results(t) = results(d))$
$\mathbf{c}_{2_3}$**:** $ProcessNode(t, \mathtt{12}) \rightarrow \exists d : (ActivityType(d, \mathtt{documentation}) \wedge Pred(t, d) \wedge results(t) = results(d))$

4.2 Interpretation of Compliance Rules

So far we showed that compliance rules are represented by FOL formulas. To round up the formalization of compliance rules, we also have to provide formal semantics for these formulas. The formal semantics has to be defined over an adequate logical model, that serves as interface between the compliance rule perspective and the process perspective. To support a variety of business process models, the logical model must be independent of a particular process meta-model. As discussed in our previous work in [6, 3, 7] *execution traces* are a suitable logical model since they are applicable to any process meta-model with formal execution semantics.

Generally, depending on their particular purpose, execution traces comprise different kinds of information. At minimum, execution traces store information on the execution of activities for a particular process instance (e.g., start, end, or start/end events for activity executions). Additional information might comprise actor assignments, input or output data, and timestamps (see, for example, the MXML execution traces used in ProM [8]). In the context of this paper, it is important to be able to instantiate compliance rules over process models. Thus, within execution traces, it should be possible to distinguish between concepts such as nodes and activities (cf. data model in Fig. 4). Thus, we define an ordered execution trace σ over a process model P as follows:
$\sigma_P := <e_1, \ldots, e_k>$ with
$e_i \in \{Start(activity, node, timestamp),\ End(activity, node, timestamp)\}$ where

- *activity* denotes the activity event e_i is associated with
- *node* denotes the process node an activity is associated with
- *timestamp* represents an abstract timestamp

For our formal interpretation of compliance rules, we need the *activity-oriented view* σ'_P of event-based execution traces σ_P. σ'_P represents the ordered activity executions in σ_P:
$\sigma'_P := <a_1, \ldots, a_m>$ with

$a_x = (activity_x, node_x, startTime_x, endTime_x),\ x = 1, .., m$ with $\exists e_i, e_j \in \sigma_P$:

- $e_i = Start(activity_x, node_x, startTime_x)$,
- $e_j = End(activity_x, node_x, endTime_x)$,
- $i < j$ and $\nexists e_l \in \sigma_P : \; i < l < j \; and \; e_l = End(activity_x, node_x, \ldots)$ and
- $\forall a_r, a_p : r < p \Rightarrow startTime_{a_r} < startTime_{a_p}$

Based on the notion of execution traces and activity executions we can provide a default interpretation of compliance rules as follows. The interpretation relates the predicates in the compliance rule formula to activity executions in the execution trace. In the following, we focus on correct execution traces (e.g., each start event has a corresponding end event).

Definition 1 (Interpretation of compliance rules). *Let $A_\mathcal{T}$ be the set of activity types of the domain model. Let $\sigma' = <a_1, \ldots, a_m>$ be an activity-oriented view of an execution trace. Then, the interpretation over σ' over $A_\mathcal{T}$ is a tuple $I_{\sigma'} = <D_{\sigma'}, d_{\sigma'}>$ with:*
$D_{\sigma'}$ is the domain of the interpretation $I_{\sigma'}$ with $D_{\sigma'} := \{a_1, \ldots, a_m\}$
Let further $N_{\sigma'} = \{n | \exists a = (activity_a, node_a, startTime_a, endTime_a) \in D_{\sigma'} : n = node_a\}$ be the set of process nodes associated to activity executions in σ'.
$d_{\sigma'}$ is a function interpreting the predicates ActivityType, ProcessNode, and Pred[3] over σ' as follows:

- $d_{\sigma'}(ActivityType) \mapsto \{(a, A), \; a = (activity_a, node_a, startTime_a, endTime_a) \in D_{\sigma'}, A \in A_\mathcal{T} \mid activity_a = A \; \vee \; activity_a \; is \; a \; subtype \; of \; A\}$
- $d_{\sigma'}(ProcessNode) \mapsto \{(a, n), \; a = (activity_a, node_a, startTime_a, endTime_a) \in D_{\sigma'}, n \in N_{\sigma'} \mid n = node_a\}$
- $d_{\sigma'}(Pred) \mapsto \{(a, b), \; a = (activity_a, node_a, startTime_a, endTime_a) \in D_{\sigma'}, b = (activity_b, node_b, startTime_b, endTime_b) \in D_{\sigma'} \mid endTime_a < startTime_b\}$

Based on Def. 1 compliance rule formulas can be interpreted over execution traces. Def. 2 provides the formal criteria necessary for compliance verification.

Definition 2 (Satisfaction of compliance rules). *Let P be a process model and let Σ'_P be the set of all activity-oriented views on execution traces of P (i.e., all traces P is able to produce). Let further $\sigma'_P \in \Sigma'_P$ be one activity-oriented view of an execution trace of P.*
We say σ'_P satisfies c (notation: $\sigma'_P \models c$) if and only if:
  $I_{\sigma'_P} \models c$.
We say P satisfies c (notation: $P \models c$) if and only if:
  $\forall \sigma'_P \in \Sigma'_P$ *holds* $\sigma'_P \models c$.

To illustrate the formal semantics defined above, consider again compliance rule c_5 (cf. Fig. 5) and the development process (cf. Fig. 1). Based on the execution traces that the development process can produce, Def. 1 can be applied to verify the development process against c_5. Informally, the straight-forward way to verify compliance rules corresponds to reachability analysis as applied

[3] Constants in a compliance rule formula (i.e., node identifiers in compliance rule instances) are mapped to themself and hence, are ommitted in the interpretation.

for checking the soundness of process models [9]. In all execution traces of the process, there is an occurrence of `end of testing` (node 14). Hence, compliance rule c_5 is activated in all executions of the development process. After `end of testing`, either the upper (case 1) or the lower split branch (case 2) is executed. In the first case, the corresponding execution trace contains the execution of the `approval` (node 15), directly followed by the execution of the `integration` (node 18). Hence, this trace satisfies c_5. In the second case, the `change process` (node 16) is executed after the `approval` (node 15). However, another `approval` activity (node 17) is executed afterwards, that is directly followed by the `integration`. According to the interpretation from Def. 1, c_5 is also satisfied in the second case.

5 Discussion and Validation

Generally, different scenarios of integrating business processes and compliance rules are conceivable. At the one side of the spectrum, all relevant compliance rules might be integrated (as far as possible) within the process model. This might be achieved by a multitude of partly nested alternative branchings. Full merging of rules into process models could be desirable for scenarios with simple process models and compliance rules that are mandatory for all process models. However, this approach has also several drawbacks. The first one is the possibly enormous complexity of the resulting process models in case of a multitude of compliance rules imposed on them. This can be compared to approaches for process configuration and variants, keeping all variants or rules within one model. As research has discussed, for many scenarios, this approach quickly leads to overly complex process models that cannot be understood by users any longer [10]. In addition, from our practical examples, we know that often not all compliance rules are mandatory, but rather have a suggesting character. Examples include medical guidelines that might be overruled by the doctor at any time. However, if all compliance rules are "hard-wired" within the process models, this optional character gets lost.

On the other side of the spectrum, we could also think of representing process models and compliance rules entirely as rules, for example within rule engines such as ILOG JRules or by using declarative approaches such as ConDec [11]. Aside from the fact that declarative process description can be covered by our approach, it cannot be neglected that in practice, most process models are described and automated using graph-based notations. Specifically, in practice, we will often find a coexistence of business process (models) and compliance rules imposed on them. Another reason for this coexistence is that compliance rules might be introduced after the process model has been designed and enacted.

For these reasons, we mainly aim at supporting the coexistence of process models and compliance rules which complies to approaches that aim at balancing flexibility and control by combining imperative and declarative process description [12]. However, since our logical model is based on execution traces (cf. Sect. 3), SeaFlows can also deal with pure process-model-based and declarative scenarios. In the latter case, execution traces are a suitable representation of process instances defined by following the imposed constraints [13].

5.1 Pattern-based Validation

We collected recurring compliance rule patterns from literature and modeled them using the SeaFlows compliance rule formalism. Many pattern-based approaches [14, 15, 13] base their patterns on patterns collected by Dwyer and Corbett [16]. Simple (particularly binary) patterns, such as *global scope presence*, *global scope absence*, *after scope absence*, *before scope absence*, *response*, and *precedence* can be modeled similarly to some of the compliance rules from Fig. 5 (e.g., compliance rule c3 corresponds to *after scope absence*). Hence, we omit the illustration of these patterns due to space limitations. Three advanced patterns [14, 16] are depicted in Fig. 7. The *response with absence* rule states that A must be followed by an occurrence of C without B occurring in the meantime. The *precedence with absence rule* states that C must be preceeded by A such that no B occurs in between. The *after scope precedence chain* states that the sequence A. . . B must occur between S and a succeeding occurrence of C.

In constrast to approaches based on a limited set of patterns and respective combinations thereof using the logical conjunction (cf. Sect. 6), the SeaFlows approach is compositional. It allows for modeling compliance rules with an antecedent and a consequence part. The modeled compliance rule graphs can be automatically mapped to logical formulas (cf. Sect. 4).

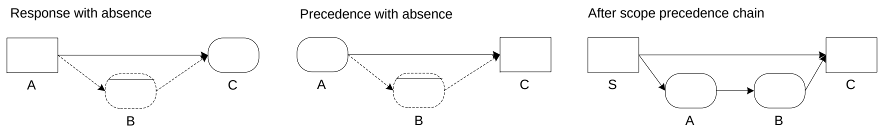

Fig. 7. Advanced compliance rule patterns [14]

5.2 Technical Validation

Fig. 8 shows the SeaFlows compliance rule editor. It allows for separately modeling the antecedent and consequence patterns of compliance rules (depicted in separate boxes). The SeaFlows editor is integrated into the AristaFlow BPM Suite that is based on ADEPT [6]. Each compliance rule node can be assigned an activity type from the AristaFlow Activity Repository. The latter also provides the activity types used to model processes and serves as domain model in our implementation. For convenient compliance rule modeling, the SeaFlows editor allows for modeling parametrized recurring compliance rules patterns. If the user wants to model a compliance rule with the same structure, he may apply the parametrized compliance rule pattern.

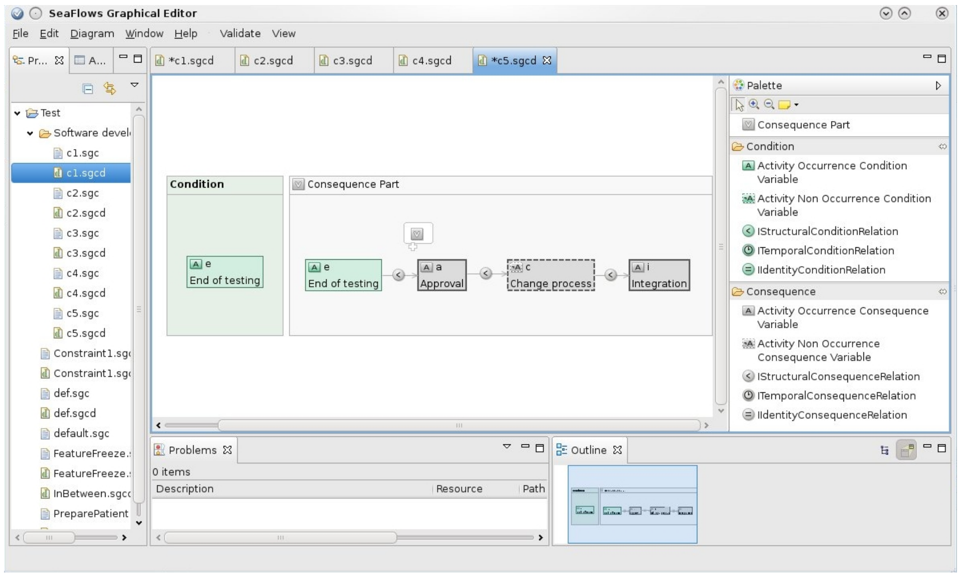

Fig. 8. The SeaFlows graphical compliance rule editor

6 Related Work

Many approaches have been proposed in literature to model compliance rules. We focus on approaches for modeling compliance rules over the occurrence and ordering relations of activity executions within a process with the goal of process verification. Existing approaches range from rather informal annotations of process models with compliance rules, over formal languages, to visual patterns.

To enable the verification of processes against imposed compliance rules, the latter have to provide formal semantics. This requirement is met by compliance rules specified in formal languages. Among the formal languages proposed for compliance rule modeling, we often come across temporal logics such as linear temporal logic (LTL) or computational tree logic (CTL) [17]. Both are fragments of first-order logic (FOL). Due to its linear time semantics, which is more suitable in the business process context [18], LTL has been clearly prefered over CTL which has branching time semantics. Albeit its expressiveness to model compliance rules with regard to the occurrence and ordering relations of activities, pure LTL has limitations when it comes to the incorporation of context conditions within compliance rules. This is due to the fact that one cannot directly address a particular state in LTL. Hence, to ensure the extendability of our approach, we opted for the formalization in FOL. The formal contract language (FCL), designed to specify business contracts, can also be applied to model compliance requirements [19, 20]. However, FCL is based on a rather state-oriented than activity-oriented paradigm. In general, it requires certain skills to model compliance rules using a formal language which might be an obstacle to the practical application of corresponding approaches.

Due to the difficulties of modeling compliance rules using formal languages in practice, many approaches from literature suggest visual notations to hide the formal details from the modeler. In [18], Liu et al. propose a graphical business property specification language (BPSL) that is based on linear temporal logic. BPSL provides visual notations for logical operators. In addition, it defines dedicated operators for recurring logical patterns. In contrast to our approach, BPSL does not support the explicit structure of antecedent and consequence patterns within a compliance rule.

Other approaches aim at establishing a set of recurring compliance rule patterns. The patterns are either given visually [21, 14, 11] or are organized in some kind of a pattern ontology [15, 22, 12]. Generally, these patterns are based on property patterns collected by Dwyer and Corbett [16]. Each pattern, in turn, can be mapped to a logical formula (e.g., in LTL). This enables the formal verification. Clearly, establishing set of rule patterns recurring in a business domain is an effective approach to facilitate compliance rule modeling. This can also be accomplished with our compliance rule language. Although the rule patterns can usually be combined using the logical conjunction, a fixed set of patterns can still be too restrictive for particular application scenarios. In these case, compositional approaches such as our approach are advantageous.

7 Summary and Outlook

We presented an approach to support the design and verification of compliance rules at different abstraction levels ranging from high-level, process independent rules to rules specified over particular process models. The main challenge was to enable the verification of process models against process-independent compliance rules. We solved this by introducing the mechanism of compliance rule instantiation using domain models and showed that instantiation results in many advantages such as easening design, mainenance, and evolution of compliance rules as well as fine-granule compliance reports. Moreover, we introduced an intuitive visualization of compliance rules and instances together with their formal semantics based on FOL and execution traces. Finally, we showed the feasability of our approach based on a pattern-based validation and by means of our powerful prototype. In future work, we will equip compliance rule graphs with *operational semantics* to enable more efficient compliance checking. In addition, we will further investigate on runtime issues. At runtime when process instances are created from process models and executed, the compliance with imposed compliance rules may change [7]. That is why it can become necessary to monitor the compliance during process execution. Moreover, we

References

1. Sadiq, S., Governatori, G., Naimiri, K.: Modeling control objectives for business process compliance. In: Proc. BPM '07. (2007)

2. Awad, A., Decker, G., Weske, M.: Efficient compliance checking using BPMN-Q and temporal logic. In: Proc. BPM 2008. LNCS 5240 (2008) 326–341
3. Ly, L.T., Rinderle-Ma, S., Dadam, P.: Integration and verification of semantic constraints in adaptive process management systems. Data and Knowledge Engineering **64** (2008) 3–23
4. Filipowska, A., Hepp, M., Kaczmarek, M., Markovic, I.: Organisational ontology framework for semantic business process management. In: Proc. BIS 2009. Volume 21 of LNBIP., Springer (2009) 1–12
5. Kumar, A., Smith, B., Pisanelli, D., Gangemi, A., Stefanelli, M.: An ontological framework for the implementation of clinical guidelines in health care organizations. Stud Health Technol Inform. **102** (2004) 95–107
6. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases **16** (2004) 91–116
7. Ly, L.T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems. Information Systems Frontiers (2009) Accepted for publication.
8. van der Aalst, W., et al.: Prom 4.0: Comprehensive support for real process analysis. In: Proc. ICATPN 2007. Volume 4546 of LNCS., Springer (2007) 484–494
9. van der Aalst, W.: Verification of workflow nets. In: Int'l Conf. on Application and Theory of Petri Nets. (1997) 407–426
10. Hallerbach, A., Bauer, T., Reichert, M.: Managing process variants in the process lifecycle. In: Proc. ICEIS'08. (2008) 154–161
11. Pesic, M., Schonenberg, M., Sidorova, N., van der Aalst, W.: Constraint-based workflow models: Change made easy. In: OTM 2007, Part I. Number 4803 in LNCS, Springer (2007) 77–94
12. Sadiq, S., Orlowska, M., Sadiq, W.: Specification and validation of process constraints for flexible workflows. Inf. Syst. **30** (2005) 349–378
13. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Control to Users. PhD thesis, Eindhoven University of Technology (2008)
14. Awad, A., Weske, M.: Visualization of compliance violation in business process models. In: Proc. BPI'09. (2009)
15. Yu, J., Manh, T.P., Hand, J., Jin, Y.: Pattern-based property specification and verification for service composition. CeCSES Report SUT.CeCSES-TR010, Swinburne University of Technology (2006)
16. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proc. ICSE'99. (1999) 411 – 420
17. Ghose, A., Koliadis, G.: Auditing business process compliance. In: Proc. ICSOC '07. Volume 4749 of LNCS., Springer (2007) 169–180
18. Liu, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Systems Journal **46** (2007) 335–361
19. Governatori, G., Hoffmann, J., Sadiq, S., Weber, I.: Detecting regulatory compliance for business process models through semantic annotations. In: Proc. BPD'08. (2008)
20. Lu, R., Sadiq, S., Governatori, G.: Compliance aware process design. In: Proc. BPM Workshops '07. (2007)
21. van der Aalst, W., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: Proc. Web Services and Formal Methods (WS-FM'06). Volume 4184 of LNCS., Springer (2006) 1–23
22. Namiri, K., Stojanovic, N.: Pattern-based design and validation of business process compliance. In: Proc. OTM 2007. Volume 4803 of LNCS., Springer (2007) 59–76