A Formal Model for Compliance Verification of Service Compositions

Groefsema, Heerko; van Beest, Nick; Aiello, Marco

Published in:
Ieee transactions on services computing

DOI:
10.1109/TSC.2016.2579621

Publication date:
2018

A Formal Model for Compliance Verification of Service Compositions

Heerko Groefsema, Nick R. T. P. van Beest, and Marco Aiello, *Senior Member, IEEE*

Abstract—Business processes design and execution environments increasingly need support from modular services in service compositions to offer the flexibility required by rapidly changing requirements. With each evolution, however, the service composition must continue to adhere to laws and regulations, resulting in a demand for automated compliance checking. Existing approaches, if at all, either offer only verification after the fact or linearize models to such an extent that parallel information is lost. We propose a mapping of service compositions to Kripke structures by using colored Petri nets. The resulting model allows preventative compliance verification using well-known temporal logics and model checking techniques while providing full insight into parallel executing branches and the local next invocation. Furthermore, the mapping causes limited state explosion, and allows for significant further model reduction. The approach is validated on a case study from a telecom company in Australia and evaluated with respect to performance and expressiveness. We demonstrate that the proposed mapping has increased expressiveness while being less vulnerable to state explosion than existing approaches, and show that even large service compositions can be verified preventatively with existing model checking techniques.

Index Terms—Service composition, business process, compliance, verification, temporal logic, colored Petri net, Kripke structure

1 Introduction

Changing laws and regulations affect the way organizations conduct business, forcing them to achieve higher flexibility and business agility. Consequently, the information systems supporting these business processes are increasingly composed in a modular, service-oriented way. Due to the increasing number of business processes and their continuous evolution, it becomes significantly more complicated to continuously ensure the compliance of these business processes and the resulting new service compositions. For this reason, automated compliance checking of service compositions is an emerging field that in many cases has become a necessity.

Compliance verification aims to prove or disprove whether a service composition adheres to a set of rules that has been imposed on it through, for example, laws, regulations, or business policies. Where soundness verification aims at the verification of a limited set of requirements to verify reachability, termination, and possibly proper completion [1]—compliance verification requires verification of a broad set of specifications.

Existing techniques perform compliance verification at different stages of the business process lifecycle, during process design, enactment of its composition, or diagnosis. Monitoring techniques are deployed during process enactment, utilizing the runtime trace of a service composition to check if a model is executing correctly. Auditing techniques are deployed during the diagnosis phase and adopt, for example, process mining to verify if a service composition has been executed correctly.

Naturally, monitoring and auditing techniques are *after the fact* techniques, meaning that issues will only ever be detected after they already have occurred. As a result, expensive rollbacks are required to undo any erroneous execution before the application of sanctions. Instead, we propose a preventative approach. Preventative approaches are deployed during design-time, and aim to prevent large issues from ever occurring.

However, existing preventative approaches invent new or extended, unsupported, logics in order to maintain the different branching constructs implemented by service compositions [2], [3], generate transition systems with large amounts of overhead (e.g., [4]), or lose information on concurrency and local next invocations in the model due to linearization [5], [6], [7].

We present a novel approach allowing preventative compliance checking that supports the different branching and merging constructs allowed by business process models and their service compositions, while significantly reducing the complexity of the analysis compared to other approaches. In addition, our approach does not require new or extended logics. As a result, well-known model checking techniques and tools can be applied during verification. An initial version of the approach was presented in [8]. An overview is given in Fig. 1. In order to provide a design and implementation independent formalized model, a service composition is first formulated as a colored Petri net [9] (CPN) through the application of workflow patterns [10]. The CPN is then translated into a Kripke structure [11] using a novel model conversion which maintains both parallelization information as well as information on the next local service invocation within

- *H. Groefsema and M. Aiello are with the Johann Bernoulli Institute, Faculty of Mathematics and Natural Sciences, University of Groningen 9746, The Netherlands. E-mail: {h.groefsema, m.aiello}@rug.nl.*
- *N. R.T.P. van Beest is with Data61, CSIRO, Brisbane, Australia. E-mail: nick.vanbeest@data61.csiro.au.*

Manuscript received 16 Dec. 2015; revised 12 Apr. 2016; accepted 4 June 2016. Date of publication 9 June 2016; date of current version 8 June 2018. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TSC.2016.2579621

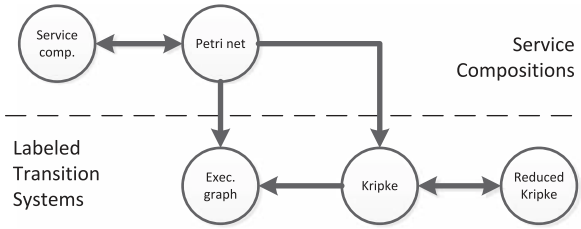

Fig. 1. Overview of the approach.

individual parallel branches or processes (i.e., the local next). Since Kripke structures are transition systems used to interpret temporal logics, the model allows verification of control flow specifications over service invocations within service compositions. Furthermore, the resulting Kripke structure can be reduced before verification. Finally, the results of verification are interpreted on the possible CPN executions. The resulting approach 1) *is process design and implementation independent*, 2) *allows for correct interpretation of temporal logic specifications*, 3) *provides full insight into possible parallel service invocations*, 4) *provides insight into the next local service invocation*, 5) *supports arbitrary cycles*, 6) *causes limited state explosion compared to other approaches*, and 7) *allows further model reduction through equivalence with respect to stuttering*.

This paper extends the work reported in [8] in the following ways: *(i)* it provides an extended reduction of the Kripke structure, allowing for a significant performance improvement in case of multiple complex formulas, *(ii)* it allows the application of CTL instead of CTL-X, *(iii)* it provides a real-life case study for validation, showing the feasibility in complex scenarios in practice, *(iv)* it provides an evaluation on expressive power when compared with existing conversion approaches and, as such, shows that our approach provides a more precise representation of process constructs, and *(v)* it provides a more extensive performance evaluation.

The paper is structured as follows. First, Section 2 discusses related work. Next, Section 3 discusses a customer support case study from the Australian telecom industry. Section 4 introduces CPN and its formal properties. Section 5 proceeds to present the model and specification used for verification, the translation from CPN to the verifiable model, and further model reduction. Subsequently, Section 6 continues and presents a proof of concept of the resulting model in the modeling language of the NuXMV model checker. Finally, Section 7 presents an evaluation of the proposed approach with respect to its expressive power and performance, before the work is concluded in Section 8.

2 Related Work

Existing preventative approaches include both formal and informal approaches. Formal approaches utilize both a formal representation of the used model and a formal specification. Informal approaches are those that lack either a formal representation of the used model, a formal specification, or both. For example, [12] directly verifies temporal logic based specifications on a service composition, without the proper support for different branching options through gateways, [6] takes an algorithmic approach, and [13] utilizes a reduction technique which results in an incomplete model. Instead, we propose a formal model upon which well-known temporal logics can be applied, and a reduction technique which only discards information on the global immediate next invocation.

Other approaches introduce new or newly extended formal specifications. For example, [2], and [14] both introduce new deontics logics to formulate compliance specifications, [3] proposes Temporal Process Logics (TPL), a modal propositional logic that is able to reason about possible process executions, [15] proposes a CTL based language, while [16] proposes a first-order extension of LTL to verify all possible process executions of artifact-centric systems. However, by introducing new or extended logics the power of accomplished logics as well as their supporting model checkers can not be exploited.

In order to simplify the challenge of formal preventative compliance verification, techniques often limit their application to acyclic models, i.e., models that do not include arbitrary cycles or loops. Acyclic compliance techniques include, for example, [17], [18], [19], and [20]. Arbitrary cycles are very powerful feature of process specifications and their service implementations which can not simply be overlooked. We propose an implementation where the notion of fairness is used to allow correct verification over cycles.

Further approaches encode service composition in such a way that a large amount of overhead is included within the state space of the model. This effect can often be traced to the decision of directly encoding service compositions into the modeling language of a model checker without careful analysis of the effect of the encoding on the internal state machine of the model checker. Approaches include [4], [21], [22], [23], [24], [25], [26], and [27]. For example, in [4], it is reported that the mapping causes a simple process of five activities and four transitions to be mapped to 201 states and 586 transitions.

Formal preventative compliance verification is generally achieved by obtaining a formal model (e.g., a Kripke structure) from the service composition. Parallel branching constructs are then supported by interleaving concurrently executing branches. Some approaches, however, disregard parallel information entirely. Such approaches include, for example, [7].

Other approaches do interleave parallel branches correctly, but interleave to such an extent that concurrent executions are linearized entirely, parallel information is lost, and duplicate states, with accompanying state explosion, are introduced. Such approaches include [28], and [29]. Parallelism is an important aspect in any service composition and, therefore, information towards possible parallel execution can be of particular importance to compliance verification. We propose a model which has been designed to minimize the state space while maintaining all temporal relations between service invocations, including concurrent ones. Furthermore, we show that the model can be significantly reduced due to this interleaving.

In [22], a translation from Petri nets to Kripke structures is proposed. By introducing intermediate states to the Kripke structure for each transition, the approach is able to define fairness conditions concerning the firing of transitions. However, we propose a smaller and simplified mapping from transitions and places to states in the Kripke structure, which provides the required domain specific occurrence information.

Finally, [30] proposes an approach towards Petri net verification based upon net unfolding. However, the approach

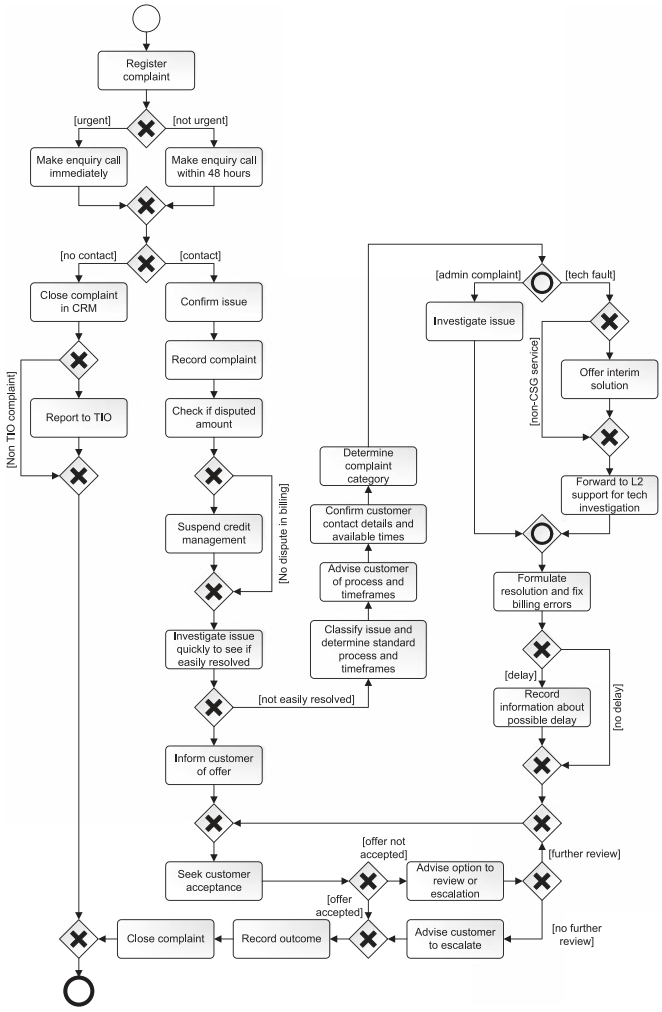

Fig. 2. The customer support process in BPMN notation.

bases its verification upon the marking of the net (i.e., tokens at places). Instead, we are interested in the firing of transitions. Although the enabling of transitions can be obtained from the marking of a net, it introduces the issue that a transition may be enabled without ever actually occurring. Although this could be solved by simply allowing only a single transition as output per place, and silent transitions otherwise, this produces significant overhead. As we are aiming to minimize overhead, a stronger sense of transition enabling is required.

3 Case Study: Customer Support

We present a case study from the telecom sector in Australia, which will be used throughout this paper. First, the business process is described. Subsequently, the rules to which the process should comply are presented.

In order to illustrate the methodology presented in this paper, a real case-study is used showing a customer support process in a telecom company in Australia, as illustrated in Fig. 2. The process is started when a complaint from a customer is received. The complaint is registered in the system and the customer is called back immediately or later, depending on the urgency of the complaint. If no further contact can be established with the customer, the complaint is closed in the CRM system and, if it concerns a Telecommunications Industry Ombudsman (TIO) complaint, the complaint is reported.

If contact is established with the customer, the issue is confirmed and the complaint is recorded. In case of a billing dispute, the credit management is suspended. If the issue can be easily resolved, the customer is informed of an offer to resolve the issue. The customer can accept or decline. In case the customer does not accept the offer, a new offer can be provided if available, or the customer can escalate the issue.

If the complaint is more complicated to resolve, the customer is advised about the timeframes required to resolve the issue and possible available times of the customer are discussed. Subsequently, non-technical issues are investigated and technical issues are forwarded to Level 2 (L2) support. When a solution is available, it is presented to the customer to be accepted or declined. If there is a possible delay, the customer is notified.

In order to ensure good service and fair outcomes for all consumers of telecommunications products in Australia, all service providers whom supply telecommunications products to customers in Australia are required to comply to the Telecommunications Consumer Protections (TCP) code of conduct. The code is registered by the Australian Communications and Media Authority (ACMA), which has appropriate powers of enforcement. As a result, the customer support process as described above has to comply with a number of rules in order to meet the code of conduct. A small number of those rules are enumerated below and used to evaluate our methodology in Section 7.

1) Resolutions to complaints should always be checked for acceptance with the customer, unless there is no contact with the customer.
2) Offers are either accepted or the customer is advised to escalate.
3) A complaint that is confirmed is recorded immediately.
4) Once a complaint has been confirmed, its outcome is always recorded.
5) Once a complaint has been confirmed, possible delays are recorded and communicated to the customer.
6) All issues are covered prior to formulating a resolution.
7) Escalated complaints are immediately recorded.

The following rules are not part of the TCP code, but can be inferred to verify a number of control-flow requirements.

8) When both technical and non-technical issues are involved in a complaint, they must be solved in parallel.
9) After the complaint category is determined, a resolution must always be provided to the customer.

4 Petri Nets

Service compositions can be modeled using many different notations. Often these notations require further formalization before formal verification can be applied. Colored Petri Nets are a popular modeling language used to formalize compositions. A CPN is a directed graph representing a process. CPNs consist of places, transitions, and arcs

between transition and place pairs. Transitions in CPNs represent service invocations in the service compositions, and places, that can hold tokens, represent the state between transitions. The previous transitions with an arc to that place have finished execution and a next transition with an arc from that place has been enabled. A CPN is defined as follows [9]:

Definition 1 (Colored Petri Net). *A Colored Petri Net is a 9-tuple* $CPN = (\Sigma, P, T, A, N, C, G, E, M_0)$, *where:*

- Σ *is a finite set of non-empty types, called* color sets,
- P *is a finite set of* places,
- T *is a finite set of* transitions,
- A *is a finite set of* arcs *such that* $P \cap T = P \cap A = T \cap A = \emptyset$,
- N *is a* node function *defined from* A *over* $P \times T \cup T \times P$,
- C *is a* color function *defined from* P *into* Σ,
- G *is a* guard function *defined from* T *into expressions such that* $\forall t \in T : [Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \Sigma]$,
- E *is an* arc expression function *defined from* A *into expressions such that* $\forall a \in A : [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$ *where* $p(a)$ *is the place of* $N(a)$,
- M_0, *the* initial marking, *is a function defined on* P, *such that* $M(p) \in [C(p) \rightarrow \mathbb{N}]_f$ *for all* $p \in P$.

The CPN state, often referred to as the *marking of CPN*, is a function M defined on P, such that $M(p) \in [C(p) \rightarrow \mathbb{N}]_f$ for all $p \in P$. Let p be a place and t a transition. Elements of $C(p)$ are called *colors*. p is an *input place* (*output place*) for t iff $(p,t) \in N((t,p) \in N)$ [9]. Every CPN is paired with an initial marking M_0. Transitions of a CPN may *occur* in order to change the marking of the CPN per the *firing rule* [9]. Places containing tokens in a marking enable possible binding elements (t,b), consisting of a transition t and a binding b of variables of t. A binding element is enabled if and only if enough tokens of the correct color are present at the input places of transition t and its guard evaluates true. More formally, iff $\forall p \in P : E(p,t)\langle b \rangle \leq M(p)$. An enabled binding element may occur, changing the marking, by removing tokens from the input places of t and adding tokens to the output places of t as dictated by the arc evaluation function. Then, a multiset Y of binding elements (t,b), or a step, is enabled iff $\forall p \in P : \sum_{(t,b) \in Y} E(p,t)\langle b \rangle \leq M(p)$, or if the sum of the binding elements is enabled. The occurrence of a step Y at a marking M_i produces a new marking M_j as denoted by $M_i \xrightarrow{Y} M_j$. All possible states of a CPN can be obtained from the initial marking through the firing rule. Utilizing CPNs as an intermediary form comes with the advantage that the marking (i.e., the distribution of tokens over places) can be seen as the process state, allowing a mapping of the state of the composition to the system model, instead of a mapping using the services of the composition (i.e., the transitions of the CPN).

Definition 2 (Reachability Graph). *The reachability graph (RG) of a CPN with markings* $M_0, \ldots, M_n$ *is a rooted directed graph* $G = (V, E, v_0)$, *where:*

- $V = \{M_0, \ldots, M_n\}$ *is the set of vertices*
- $v_0 = M_0$ *is the root node*
- $E = \{(M_i, (t,b), M_j) \mid M_i \in V \wedge M_i \xrightarrow{(t,b)} M_j\}$ *is the set of edges, where each edge represents the firing of a binding element* (t,b) *at a marking* M_i *such that a marking* M_j *is produced.*

The general approach for converting CPNs into transition systems is used when generating the reachability graph of the CPN (Definition 2) [31]. Starting from the initial marking M_0, states are created for each encountered marking while enabled binding elements occur to generate new markings. In this case, the occurrence of transitions can not be concluded from the states of the transition system but only from its relations. A discussion of the issues with respect to the expressive power provided by the reachability graph while applying temporal logics, and how the presented model solves those issues, is included in the evaluation section.

The customer support case is modeled using the BPMN Business Process Diagram standard [32]. Several methods to convert a BPMN diagram to Petri net exist, including [33]. However, in order to convert the customer support diagram depicted in Fig. 2 to a CPN, we use a mapping based directly upon the workflow patterns described in [10]. Patterns within service compositions can be easily identified by its description, motivation, context, and CPN overview. Patterns are then applied together using their CPN overviews to form the CPN of a composition. Although the graphs presented throughout this paper can, in theory, become infinite, we assume the correct application of workflow patterns to form sound service compositions with finite graphs. Other modeling standards (e.g., WS-BPEL) can be used and similarly converted, making the approach design and implementation specification independent. Fig. 3 depicts the CPN of the customer support process.

5 Design Time Business Process Verification

Before service compositions can be verified using model checking, they are translated from a formal CPN into a verifiable system model. The models and specifications required for verification are labeled transition systems and temporal logics.

5.1 Model and Specification

As the CPN state can be captured based on its marking, a state-based labeled transition system is used as the system model for model checking. A state-based labeled transition system is a transition system with a labeling function over its states, instead of (or in addition to) a labeling function over its flow relations. A Kripke structure is such a state-based labeled transition system [34].

Definition 3 (Kripke structure). *Let* AP *be a set of atomic propositions. A Kripke structure* K *over* AP *is a quadruple* $K = (S, S_0, R, L)$, *where:*

- S *is a finite set of states.*
- $S_0 \subseteq S$ *is a set of initial states.*
- $R \subseteq S \times S$ *is a transition relation such that it is left-total, meaning that for each* $s \in S$ *there exists a state* $s' \in S$ *such that* $(s, s') \in R$.

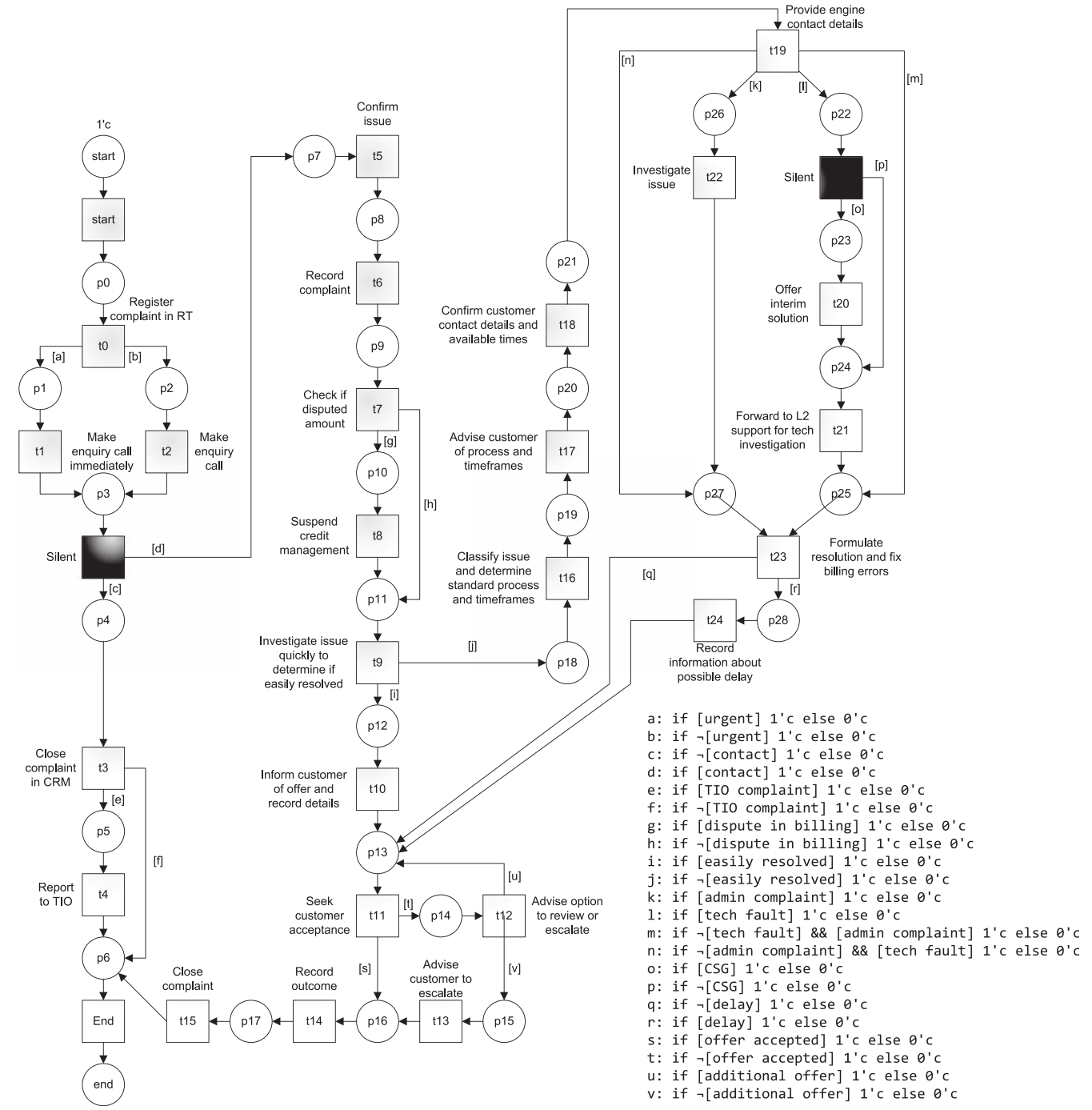

Fig. 3. The customer support process as CPN.

- *$L : S \rightarrow 2^{AP}$ is a labeling function with the set of atomic propositions that are true in that state.*

Kripke structures are often used to interpret temporal logics. Temporal logics are formalisms that are able to reason over linear- or branching-time execution paths within system models. Linear-time temporal logics specify events over sequences of states known as paths. A path π is an infinite sequence of states $\pi = s_0, s_1, \ldots$, such that the relation (s_i, s_{i+1}) exists for every $i \geq 0$. Branching-time temporal logics specify events over future states on branching paths, or tree-like structures, where each branch represents a possible execution path. One of the most notable branching-time temporal logics is Computation Tree Logic (CTL) [11]. Often used with formal verification, CTL is especially useful when considering the different branching constructs available to compositions.

Definition 4 (CTL syntax). *The language of well-formed CTL formulas is generated by the following grammar, assuming $p \in AP$.*

$$\begin{aligned}\phi ::= &\top \mid \bot \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid \phi \Rightarrow \phi \mid \phi \Leftrightarrow \phi \mid \\ &\mathrm{AX}\phi \mid \mathrm{EX}\phi \mid \mathrm{AG}\phi \mid \mathrm{EG}\phi \mid \\ &\mathrm{AF}\phi \mid \mathrm{EF}\phi \mid \mathrm{A}[\phi\ \mathrm{U}\ \phi] \mid \mathrm{E}[\phi\ \mathrm{U}\ \phi] \mid\end{aligned}$$

CTL is equipped with four *temporal operators*:

- $X\phi$ Nexttime: ϕ has to hold at the next state.
- $G\phi$ Globally: ϕ has to hold on every state on the entire subsequent path.
- $F\phi$ Future: ϕ eventually has to hold in a future state.
- $[\phi\ U\ \phi']$ Until: ϕ has to hold until ϕ', which holds at a state on the path from the current state or the current state itself.

Each temporal operator ψ in CTL is paired with an operator ϕ over paths. Operators over paths specify whether some or all branches possess properties starting at the current state. CTL defines the following operators over paths:

- $A\psi$ All: ψ holds on all paths flowing from the current state.
- $E\psi$ Exists: ψ holds on at least one path flowing from the current state.

The semantics of CTL are defined on a Kripke structure, which is shown formally in Appendix A. These definitions are required to present a correct model translation from CPN to Kripke structures, such that temporal logic formulas expressed using CTL can be verified upon the Kripke structure.

5.2 Verifiable Model of a Colored Petri Net

Since transitions in CPN relate directly to services when describing service compositions, a common technique for converting CPN into transition systems entails the inclusion of transitions as states in the transition system upon their occurrence. While traversing the CPN from its initial marking M_0, transitions are continuously added as states while they occur. A major drawback of this technique occurs when transitions are encountered multiple times during, for example, the interleaving of parallel paths. In such cases, the approach causes the inclusion of multiple copies of the same state. Due to this, an enormous amount of duplicate states are created. Instead, we define a verifiable model that only includes states for each marking and each set of binding elements that are not just enabled, but will occur at that marking. This set of parallel enabled binding elements at a marking is formalized in Definition 5.

Definition 5 (Parallel Enabled Binding Elements). *The set of all possible parallel enabled binding elements $Y_x(M)$ at a marking M is the set $Y_x(M) = \{Y \mid Y \in Y_p(M) \ \wedge \forall Y' \in Y_p(M) : Y \not\subset Y' \wedge Y \neq \emptyset\}$, with:*

- $Y_p(M) = \{Y \mid Y \in \mathcal{P}(Y_e) \ \wedge \forall p \in P : \sum_{(t,b)\in Y} E(p,t)\langle b\rangle \leq M(p)\}$ *as the enabled steps of the powerset $\mathcal{P}$ of $Y_e(M)$, and*
- $Y_e(M) = \{(t,b) \mid \forall p \in P : E(p,t)\langle b\rangle \leq M(p)\}$ *as the set of binding elements enabled at a marking M.*

In order to obtain a verifiable system model from the markings of a CPN, we first specify what the places containing tokens in a marking represent. Places containing tokens at a marking M allow binding elements to be enabled. Enabled binding elements may occur. The enabled powerset $Y_p(M)$ of the enabled binding elements represents the set of possible parallel occurrences. Then, by taking those sets that are not a subset of other sets of the enabled powerset, we find the different sets Y_xM of binding elements that may occur at that marking. Once a binding element of such a set occurs, the other elements of that set will occur in the future. This set, $Y_x(M)$, is used in upcoming definitions to determine the different labelings when multiple sets of binding occurrences can occur simultaneously at a marking while advancing between states.

Using these conventions, we convert a colored Petri net *CPN* into a Kripke structure K by creating states at each marking M_i for each set of binding elements that can occur concurrently at a marking M_i, and then having each binding element occur individually to find possible next states. Although binding elements could occur simultaneously, allowing these would only provide for additional relations, creating shorter paths between existing states when interleaving. Even though CPN could theoretically reach an infinite number of markings, the use of the sound and safe workflow patterns restrict the CPN in such a way that it always produces a number of markings that is finite. The verifiable system model of a business process model, called the transition graph, is formalized in Definition 6.

Definition 6 (Transition Graph). *Let AP be a set of atomic propositions. The transition graph of a CPN with markings $M_0, \ldots, M_n$ is a Kripke structure $K = (S, S_0, R, L)$ over AP, with:*

- $AP = \{M_0, \ldots, M_n\} \cup \{(t,b) \in Y \mid Y \in \{Y_x(M_0) \cup \ldots \cup Y_x(M_n)\}\}$
- $S = \{s_i^Y \mid Y \in Y_x(M_i)\}$
- $S_0 = \{s_0^Y \mid Y \in Y_x(M_0)\}$
- $L(s_i^Y) = \{M_i\} \cup \{(t,b) \mid (t,b) \in Y\}$
- $R = \{(s_i, s_j) \mid (t,b) \in L(s_i) \wedge M_i \in L(s_i) \ \wedge M_j \in L(s_j) \wedge M_i \xrightarrow{(t,b)} M_j\}$[1]

Definition 6 introduces a novel conversion from the marking of the CPN where a state, which is labeled with a binding element, can be interpreted as that binding element currently occurring. Binding elements, however, can be found as occurring over multiple states. A binding element has only occurred (i.e., finished occurring) when it is occurring at one state and not occurring at a next state. Binding elements occur concurrently during interleaving of parallel branches. In such cases, states are labeled with multiple binding elements. Although the transition graph is a graph containing states with labels over the markings $M_0, \ldots, M_n$ and steps (t,b), only the steps (t,b) are used as verification propositions. When b is understood, we simply write t (such as is the case with business process models translated from workflow patterns where only one functional binding $b = \langle c\rangle$ is used). Fig. 4 depicts the transition graph resulting from this conversion process on the customer support CPN depicted in Fig. 3. Although bindings are abstracted away in the examples to increase readability, they formally do exist in the models. As such, any information included within the model within complex bindings, such as data, is maintained within the transition graph (though, possibly, at the cost of model size and performance).

1. Although Definition 6 uses elements from the definition itself to define R (i.e., the labeling function L), this is merely done to produce a more concise and readable definition.

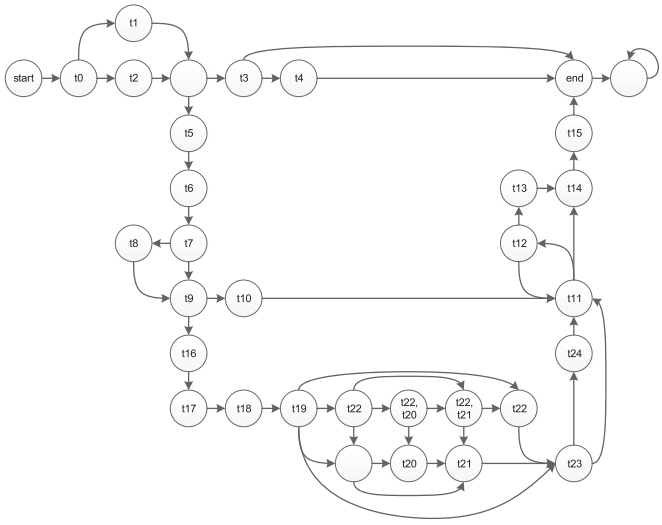

Fig. 4. The customer support process as a Kripke structure.

Even though states are labeled with markings $M_0, , M_n$, these should not be used as propositions when verifying by means of the transition graph. The markings are only included in the transition graph in order to obtain a correct model (i.e., to detect the difference between a marking where a step (t, b) is enabled without additional tokens at places and a similar marking with additional tokens unrelated to (t, b)). When verifying over markings, using the well-known reachability graph is preferred. The reachability graph can equally be obtained from the transition graph.

Definition 7 (Reachability Graph of a Transition Graph). *Let AP be a set of atomic propositions. The reachability graph of the Transition Graph $K = (S, S_0, R, L)$ over AP is a rooted directed graph $G = (V, E, v_0)$, with:*

- $V = \{M_i \mid s_i \in S : M_i \in L(s_i)\}$ *is the set of vertices*
- $v_0 = M_0 \mid s_0 \in S_0 : M_0 \in L(s_0)$ *is the root node*
- $E = \{(M_i, (t, b), M_j) \mid (s_i, s_j) \in R \wedge M_i \in L(s_i) \wedge M_j \in L(s_j) \wedge (t, b) = L(s_i) \setminus L(s_j) \setminus M_i\}$ *is the set of edges.*

Definition 7 completes the cycle of model conversions as depicted in Fig. 5. Together with earlier definitions, Definition 7 allows a CPN to be transformed into a transition graph, which then can be transformed back into CPN executions as described by the reachability graph.

CTL specifications can be interpreted on service compositions as depicted in Fig. 6. Compositions defined using CPN can be simulated to obtain a transition graph upon which the branching time temporal logic CTL can be interpreted. This interpretation can be understood upon the possible executions of the CPN as expressed by its reachability graph. A formal definition of the semantics of CTL, defined upon the reachability graph of a CPN, is provided in Appendix B, along with the formal proof that the semantics are sound and complete.

5.3 Model Reduction

The transition graph can be reduced before the model is verified by model checking. As model checking techniques verify models with given specifications in an exhaustive fashion, any reduction of the model benefits performance.

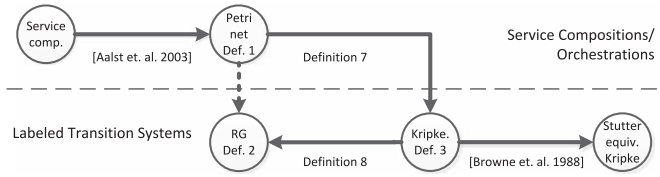

Fig. 5. Model conversion.

Model reduction can be achieved through the removal of unused atomic propositions and model equivalence under the absence of the nexttime operator, otherwise known as equivalence with respect to stuttering [35]. Equivalence with respect to stuttering is a useful notion when considering concurrent systems– or, in our case, concurrently executing branches. In such cases it may be dangerous to evaluate the nexttime operator, X, as it refers to the next global state (i.e., the typical interleaved execution of concurrent programs or branches) and not the next local state (i.e., the execution of one such program or branch) [36]. Instead, when one considers the nexttime operator, one actually means to describe that something occurs before other local occurrences, which, in turn, can be specified easily using the other operators. For example, the until operator can be used on the transition graph to specify the next local occurrence (e.g., $AG(t20 \Rightarrow A[t20Ut21])$ can be used to specify that $t20$ is followed by $t21$ in every execution branch of the compliant process depicted in Fig. 4). In order to offer a parallel interleaving safe evaluation of the nexttime operator, any nexttime operator is parsed into an until operator. As an additional benefit, without having to evaluate the nexttime operator, the model can be significantly reduced before verification through the notion of equivalence with respect to stuttering [35].

A finite Kripke structure K can be uniquely identified by a single CTL formula F_K [35]. As a result, F_K can be used to evaluate the equivalence of other Kripke structures K' to K. When considering F_K without nexttime operators, the equivalence of K' can be evaluated with respect to stuttering [35]. Two Kripke structures K and K' are equivalent with respect to stuttering if all paths from the initial states $s_0 \in S_0$ of K are stutter equivalent with the paths from the initial states $s'_0 \in S'_0$ of K' and vice versa. Two paths are stutter equivalent, as denoted by $\pi \sim_{st} \pi'$, if both paths can be partitioned into blocks of states $\pi = k_0, k_1, \ldots$ and $\pi' = k'_0, k'_1, ..$ such that $\forall s \in k_i, \forall s' \in k'_i : L(s) = L(s')$ for $i \geq 0$ [37].

To reduce the model, first those atomic propositions not used by specifications, with the exception of those relating to events, are removed. Then, the atomic propositions related to markings are removed from the labels of all states and the set AP such that $M_i \notin AP$ and $\forall s \in S : M_i \notin L(s)$ for $0 \leq i \leq n$. Finally, a stutter equivalent model with respect to the used atomic propositions is obtained.

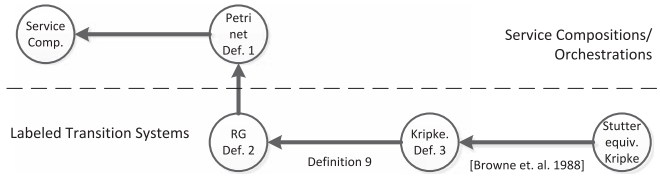

Fig. 6. Specification interpretation.

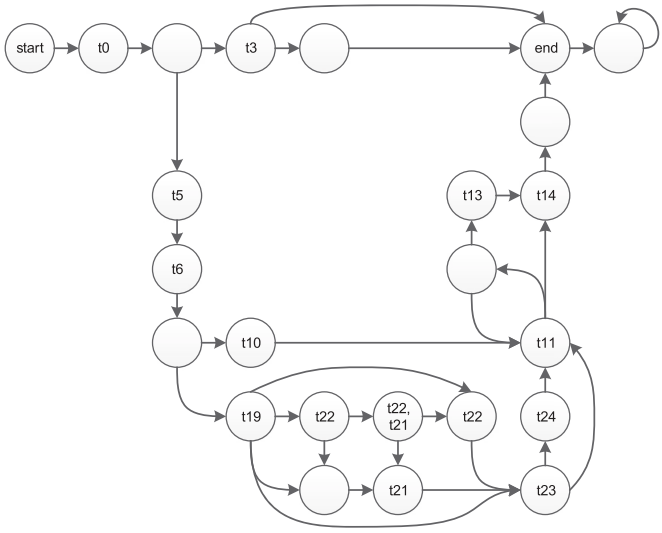

Fig. 7. The customer support process as a reduced Kripke structure w.r.t the atomic propositions required by the compliance specifications.

Although the removed labels are needed during the conversion process to ensure unique states to be generated, they can be removed at this point because they are not used by specifications or because specifications should only be expressed using bindings on activities or events of the business process (i.e., transitions) and not its progression information (i.e., marking). Note that the reduction is only possible after the removal of the progression information. Because of this, reduction can only be accomplished after obtaining the transition graph.

Fig. 7 depicts the stutter equivalent model of the Kripke structure depicted in Fig. 4 after the removal of the unused atomic propositions. Note that several unlabeled states remain. These can not be removed, as it would affect the evaluation of formulas (e.g., $AG(t19 \Rightarrow A[(t19 \vee t21 \vee t22)\ U\ t23])$ would incorrectly evaluate true).

6 Proof of Concept Implementation

Model checkers verify systems defined in the modeling language of the model checker. These modeling languages range from forms of programming code, which are then translated to transition systems, to simple transition systems themselves. In order to provide a proof of concept and provide verification results using the presented models, the Kripke structure is automatically translated into the modeling language of a well-known model checker and its related specifications subsequently verified.

NuXMV [38] is a symbolic model checker supporting both CTL and LTL based formulas. The modeling language of NuXMV allows the user to specify a state-based labeled transition system for verification directly, which is highly preferred when implementing a Kripke structure.

For the purpose of implementing a Kripke structure, the modeling language of NuXMV consists of a module with four distinct blocks. The first, *VAR*, defines the states of the transition system. The second, *DEFINE*, specifies each variable and assigns them to the states in which they are true. The third, *ASSIGN*, first defines the set of initial states and then specifies next states using the *next* function. In the final block, a set of temporal logic formulas and fairness conditions is listed. In case of the implementation of a Kripke structure $M = (S, I, R, L)$ with atomic propositions AP, the *VAR* block specifies all states $s \in S$, the *DEFINE* block specifies each variable $v \in AP$ and assigns it to states $s \in S$ for which $v \in L(s)$. The *ASSIGN* block lists the initial states $s \in I$ and then specifies each relation $(s, s') \in R$ as *next(s) := s'*. Finally, a list of temporal logic formulas and fairness conditions is included. In order to avoid issues with loops, a justice fairness condition is included. Justice conditions are considered to be true infinitely many times in fair paths. As such, infinite loops are omitted using justice conditions which state that loops may not repeat an infinite number of times.

Next, the customer support process, as presented in Section 3, will be verified on compliance with respect to the defined compliance rules. First, the informal rules specified in Section 3 are formulated as CTL specifications as follows:

1) $!E[!(t11 \vee t3)\ U\ end] \wedge AF(end)$
2) $AG(t10 \Rightarrow AF(t13 \vee t14))$
3) $AG(t5 \Rightarrow A[t5\ U\ t6])$
4) $AG(t5 \Rightarrow AF(t14))$
5) $AG(t5 \Rightarrow EF(t24))$
6) $!EF(t21 \wedge t23) \wedge !EF(t22 \wedge t23)$
7) $AG(t13 \Rightarrow A[t13\ U\ t14])$
8) $EF(t21 \wedge t22)$
9) $AG(t19 \Rightarrow EF(t23))$

Below, the customer support process is presented as a Kripke structure within the NuXMV modeling language.

```
MODULE main
  VAR
    state:{S0, S1, S2, S4, S5, S6, S7, S9, S10,
          S11,S16,S17,
          S18,S19,S20,S21,S22,S26,S28,S30,
          S31,S35,S37,S43};

  DEFINE
    start  := ( state = S0 );
    t0     := ( state = S1 );
    t3     := ( state = S4 );
    t5     := ( state = S9 );
    t6     := ( state = S10 );
    t10    := ( state = S43 );
    t11    := ( state = S18 );
    t13    := ( state = S20 );
    t14    := ( state = S21 );
    t19    := ( state = S16 );
    t21    := ( state = S31 ) | ( state = S37 );
    t22    := ( state = S28 ) | ( state = S35 )
                              | ( state = S37 );
    t23    := ( state = S17 );
    t24    := ( state = S26 );
    end    := ( state = S5 );
    loop   := ( state = S18 ) | ( state = S19 );

  ASSIGN
    init(state)  := {S0};
    next(state)  :=
      case
        state  =  S0  : {S1};
        state  =  S1  : {S2};
```

```
      state = S2  : {S4,S9};
      state = S4  : {S5,S7};
      state = S5  : {S6};
      state = S6  : {S6};
      state = S7  : {S5};
      state = S9  : {S10};
      state = S10 : {S11};
      state = S11 : {S16,S43};
      state = S16 : {S17,S28,S30,S35};
      state = S17 : {S18,S26};
      state = S18 : {S19,S21};
      state = S19 : {S18,S20};
      state = S20 : {S21};
      state = S21 : {S22};
      state = S22 : {S5};
      state = S26 : {S18};
      state = S28 : {S17};
      state = S30 : {S31};
      state = S31 : {S17};
      state = S35 : {S30,S37};
      state = S37 : {S28,S31};
      state = S43 : {S18};
   esac;

FAIRNESS !loop;
CTLSPEC !E[!(t11 | t3) U end] & AF(end);
CTLSPEC AG(t10 -> AF(t13 | t14));
CTLSPEC AG(t5 -> A[t5 U t6]);
CTLSPEC AG(t5 -> AF(t14));
CTLSPEC AG(t5 -> EF(t24));
CTLSPEC !EF(t21 & t23) & !EF(t22 & t23);
CTLSPEC AG(t13 -> A[t13 U t14]);
CTLSPEC EF(t21 & t22);
CTLSPEC AG(t19 -> EF(t23));
```

The resulting model is subsequently offered to NuXMV in order to automatically verify its compliance with the rules specified in Section 3. The first two specifications returned false, signifying compliance issues. These two rules state that a complaint can result in multiple successive offers and that complaints should always result in an offer unless there is no contact with the complaining party. When inspecting the process, it is indeed clear that offers only take place in certain branches of the process where there has been contact with the customer. Additionally, it is true that the customer is informed of an offer only once, and that when this offer is rejected customer acceptance is sought multiple times for this offer without any possibility of revision of the offer.

7 Evaluation

In order to demonstrate that the transition graph approach is not only able to return valuable compliance results, but also allows greater insight into concurrently executing branches while limiting model size and time to generate, the expressive power and performance of the approach are analyzed. First, the expressive power is discussed by comparing the transition graph with other graphs when verifying CTL formulas on different control-flow constructs. Subsequently, the performance and complexity of the conversion and reduction processes and the resulting model sizes are discussed.

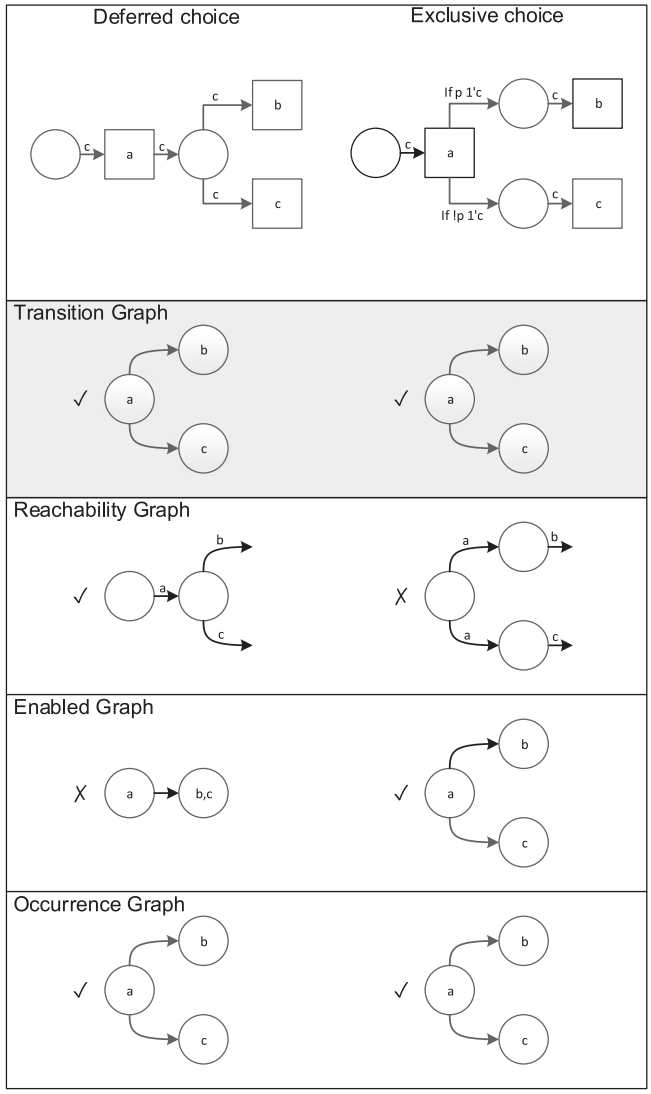

Fig. 8. Graph comparison of the deferred and exclusive choice constructs.

7.1 Expressive Power

The expressive power of the transition graph while using CTL is evaluated through its application on four workflow patterns [10] and compared to relevant related work, i.e., formal cyclic preventative approaches. The approaches are categorized into three types of graphs: the reachability graph [31], the transition occurrence graph similar to the approaches of [28] and [29], and a version of the reachability graph where states are labeled with enabled transitions (i.e., the enabled graph), similar in effect to [30]. The control-flow constructs used for comparison are deferred choice, exclusive choice, parallel split, and interleaved routing.

Fig. 8 depicts the deferred choice and exclusive choice construct together with their representations using the four different graphs. Although the deferred and exclusive choice are different constructs, both fulfill a similar role; a choice of paths depending on some event or condition.

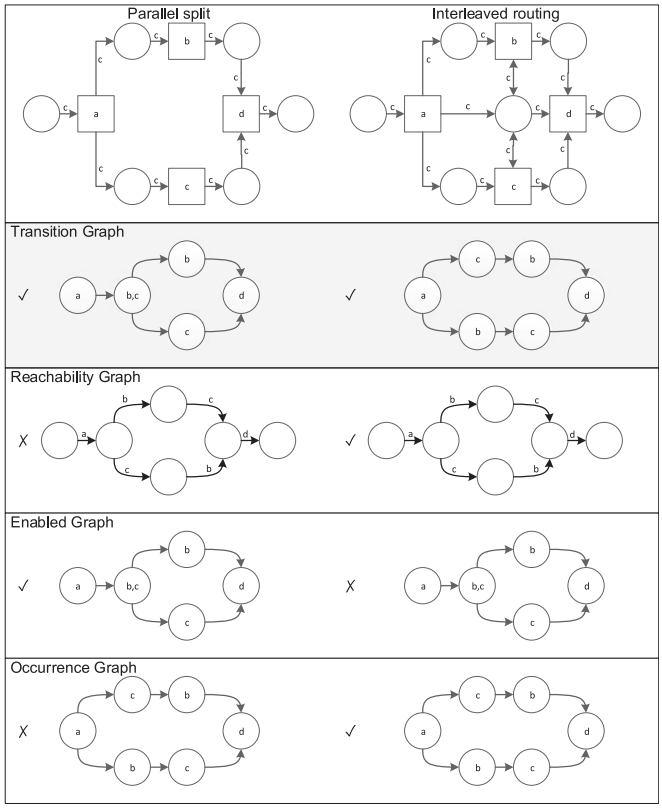

Fig. 9. Graph comparison of parallel split and interleaved routing constructs.

After the occurrence of transition a, either transition b or c occurs. This functionality can indeed be seen in both the transition and occurrence graphs. The enabled graph, however, poses a problem when representing the deferred choice. After transition a is enabled, both b and c are enabled simultaneously. Without a way to foresee which of these two transitions actually occurs, verification of any logic formula on this graph can only guarantee the enabled state of transitions and never its occurrence. Although the reachability graph, in principle, produces correct graphs where after the occurrence of transition a either b or c occurs, its representation of the exclusive choice pattern does introduce issues with verification of branching time temporal logics. For example, the response formula $AG(a \Rightarrow EFb)$ (a is eventually followed by b in some path) would evaluate to false because the occurrence of a on the lower branch which is followed only by c. While the formula would evaluate true for the upper branch, it would evaluate false in its entirety due to the false result of the lower branch. This undesired behavior is introduced because the reachability graph is treating the occurrence of transition a with the boolean p evaluating true as a different occurrence of transition a with p evaluating false. In reality, when looking at the pattern, it is not the transition that differs, but the conditions on the outward arcs. The occurring transition executes the same with different values of p. Indeed, when mapping this pattern to a BPMN model, activity a would precede the exclusive choice gate which then splits into two paths leading to either activity b or c. A functionality which the reachability graph clearly does not represent correctly.

TABLE 1
Conversion and Reduction Algorithm Results of the Telecom Processes

Kripke structure			Red. Kripke structure			Performance	
$\|S\|$	$\|R\|$	$\|AP\|$	$\|S\|$	$\|R\|$	$\|AP\|$	Conversion	Reduction
33	49	26	24	35	15	29 ms	1 ms

Fig. 9 depicts the parallel split/join and interleaved routing patterns and their representations using the four different graphs. Although the two patterns look similar, their behavior is different. Where the parallel split/join pattern splits into two parallel executing branches which only synchronize at the join, the interleaved routing performs some form of synchronization during execution of each branch such that no two transitions can occur in parallel, but does so without inferring an explicit occurrence order between branches. In essence, the occurrence of the transitions on the different branches is interleaved. The functionality of these two patterns can indeed be seen in the transition graph. When representing the parallel split using the transition graph, transition b and c start occurring in parallel, after which either b or c has finished occurring (i.e., has occurred) and the other remains in an occurring state until it also has finished occurring and synchronizes into d. Furthermore, when representing the interleaved routing pattern, the transition graph indeed linearizes the occurring transitions on each branch as specified. For the remaining graphs one immediately notices an issue of expressive power. Indeed, the remaining graphs each represent both patterns with the exact same graph, even through the patterns behave in very different ways. For the enabled graph this means that the interleaved routing construct is not being linearized because, naturally, both transition b and c are enabled even though they may not occur simultaneously. In case of both the reachability graph and the occurrence graph, the parallel split/join construct is linearized as well. As the occurrence of transitions b and c is being linearized, any parallel occurrence information as well as local next occurrence information is lost. Where the transition graph can be used to verify that transitions b and c can occur in parallel by verifying the existence of both labels at one state (i.e., $EF(b \wedge c)$), the reachability and occurrence graphs can only be used to verify whether transitions b and c occur until d (i.e., $AG(a \Rightarrow A[(b \vee c)U\ d]$). Even when using LTL including always and exists path mechanisms, specifications can only verify that sometimes transition b occurs before transition c and vice versa (i.e., $\exists G(c \Rightarrow Fb)$ and $\exists G(b \Rightarrow Fc)$), a specification which could also be satisfied by a simple loop. The same is true for the local next occurrence, which can be verified using the transition graph by verifying whether a label holds until the next (i.e., $AG(b \Rightarrow E[b\ U\ d])$). A specification which correctly holds for the parallel split pattern and not for the interleaved routing pattern when considering the transition graph. Again, the other graphs fail to capture the difference.

As the transition graph can be used to verify parallel and local next occurrences, it bridges a clear gap in the expressive power required for verification of service compositions.

TABLE 2
Conversion Algorithm Results of Compositions with n Branches of m Activities

	Composition			Kripke structure			Performance
#	Type	n	m	$\|S\|$	$\|R\|$	$\|AP\|$	Conversion
1	SEQ	1	5	8	8	7	3 ms
2	XOR	2	5	13	14	12	3 ms
3	XOR	3	5	18	20	17	3 ms
4	XOR	4	5	23	26	22	3 ms
9	AND	2	5	38	63	12	4 ms
10	AND	3	5	218	543	17	27 ms
11	AND	4	5	1298	4323	22	96 ms
5	SEQ	1	50	53	53	52	5 ms
6	XOR	2	50	103	104	102	12 ms
7	XOR	3	50	153	155	152	16 ms
8	XOR	4	50	203	206	202	22 ms
13	AND	2	50	2603	5103	102	102 ms
14	AND	3	50	132653	390153	152	1.853ms
18	AND	4	50	6765203	26530203	202	184.758 ms
12	AND	4	8	6563	23331	34	214 ms
15	AND	3	80	531443	1574643	242	10.903 ms
17	AND	4	30	923523	3574923	122	20.345 ms
19	AND	5	15	1048578	4915203	77	22.444 ms
16	AND	3	100	1030303	3060303	302	25.023 ms
20	AND	6	10	1771563	9663063	62	44.364 ms
21	AND	7	8	4782971	29760699	58	158.008 ms

TABLE 3
Reduction Algorithm Results of Compositions with n Branches of m Activities after Reduction by 50 percent of Randomly Chosen Atomic Propositions

	Reduced Kripke structure			Performance
#	$\|S\|$	$\|R\|$	$\|AP\|$	Reduction
1	6 (25%)	6 (25%)	4	0 ms
2	10 (23%)	11 (21%)	6	0 ms
3	15 (17%)	17 (15%)	9	0 ms
4	19 (17%)	22 (15%)	11	0 ms
9	15 (61%)	22 (65%)	6	1 ms
10	67 (69%)	148 (73%)	9	10 ms
11	403 (69%)	1245 (71%)	11	78 ms
5	37 (30%)	37 (30%)	26	2 ms
6	76 (26%)	77 (26%)	51	2 ms
7	114 (25%)	116 (25%)	76	4 ms
8	151 (26%)	154 (25%)	101	5 ms
13	1409 (46%)	2741 (46%)	51	90 ms
14	50823 (62%)	148319 (62%)	76	704 ms
18	1915203 (72%)	7454605 (72%)	101	37.294 ms
12	2523 (62%)	8716 (63%)	17	100 ms
15	207363 (61%)	612122 (61%)	121	2.297 ms
17	241227 (74%)	924885 (74%)	61	4.481 ms
19	168003 (84%)	762205 (84%)	39	6.714 ms
16	453603 (66%)	1344692 (66%)	151	3.548 ms
20	396903 (88%)	2100427 (88%)	31	12.519 ms
21	774147 (84%)	4577290 (85%)	29	43.765 ms

7.2 Performance

The performance of the approach was first evaluated by executing an implementation of Definition 6 on the case study. Performance tests were attained using a system with an Intel Core I7-4771 CPU at 3.50 GHz, 32 GB of memory, running Windows 7 x64 and Java 7. The results are shown in Table 1.

The case consists of 33 states and 48 transitions in its original form. After reduction with respect to the relevant AP used in the formulas, the amount of states is reduced by 27 percent and the transitions by 29 percent. However, it is clear from the table that the required execution time for the proposed conversion and reduction is negligible when applied to the real-life case. The customer support process, although consisting of a realistic amount of services, does not comprise excessively complex control-flow structures. In fact, it is (like many compositions), predominantly sequential. As such, the effect on performance remains rather limited.

As such, in order to test the limits of the algorithm with respect to performance under increasing concurrency within parallel branches, the approach was subsequently evaluated by executing an implementation of Definition 6 on artificial service compositions. These artificial compositions were specifically generated for performance evaluation purposes by specifying a gate type, number of branches, and branch length. The results can be found in Tables 2 and 3.

Table 2 displays information on the performance and results of the conversion process. Its columns describe the case number, the composition (containing sequence/exclusive/parallel branching, the number of branches n, and number of services per branch m), the resulting Kripke structure (number of states S, relations R, and atomic propositions AP), and the performance of the conversion algorithm.

Test cases 1-8 demonstrate that sequential compositions and compositions including exclusive paths are of no concern to the conversion performance, as they are converted within milliseconds. However, processes including parallel regions introduce an increased complexity. This increased complexity is introduced due to the interleaving of concurrent services on parallel branches. Although the interleaving is highly efficient due to the lack of complete linearization, it does produce $\prod_{i=1}^{n}(m_i+1)$ states, where m_i is the length of branch number i out of n branches (for equal branch lengths the complexity is $(m+1)^n$). This is confirmed by test cases 9-11, which show that parallel interleavings of equal sizes to the exclusive compositions (cases 2-4) are converted within 96 milliseconds. When increasing the length of the branches in test cases 13-14, and 18 from 5 to 50 services, the time to convert is increased to 102 milliseconds for two branches, 1.8 seconds for three branches, and 185 seconds for four branches. Finally, test cases 15-17 and 19-21 demonstrate that compositions with extremely large parallel sections of 56 to 300 services are converted in 10 to 158 seconds. However, as the number of parallel branches increases, there is a severe limit on the length of the branches supported. This is due to the fact that the number of branches n is the largest factor of the complexity formula. Although extremely large interleavings require several minutes to compute, this is negligible when verifying pre-runtime.

Next, we evaluate the model reduction algorithm. Model reduction is achieved through the removal of atomic propositions that are not relevant to specifications and equivalence with respect to stuttering. In order to evaluate the degree of reduction, we randomly remove 50 percent of atomic propositions of the cases listed in Table 2 before applying the reduction algorithm. Table 3 displays information on the resulting reduced Kripke structure (number of states S and percentage of original, relations R and percentage of original, and number of remaining atomic propositions AP) and the time it took to reduce the model. Note that reduced Kripke structure sizes may vary due to the random removal of atomic propositions.

Test cases 1-8, which contain sequential compositions and compositions including exclusive paths, are of no concern to the performance of the reduction algorithm either. These

compositions are reduced within 0 to 5 milliseconds. Compositions including parallel regions display the greatest reduction results. When increasing the number of parallel branches, reduction effects increase accordingly. Test cases 9-11 demonstrate that parallel interleavings of average sizes are reduced within 78 milliseconds. Increasing the length of the branches from 5 to 50 services increases the time to reduce to 90 milliseconds, 704 milliseconds, and 37 seconds for compositions with two, three, and four branches respectively.

The effect of model reduction displays varying results. Because sequential compositions generate relatively simple Kripke structures, model reduction shows limited effects with reductions of 15 to 30 percent. For sequential compositions, the worst case reduction with less than half of the AP removed is 0 percent. Processes with parallel interleaved paths, however, show a much larger effect with a 46 to 72 percent reduction. In this case, while the complexity of the Kripke structures increases with additional branches, model reduction naturally gains increased effect due to the particular applied interleaving. Because transition occurrence is interleaved with concurrent information, and not entirely sequentialized, reduction with regard to equivalence under stuttering gains increased effect. With each removed atomic proposition, a significant amount of interleaved states is reduced. Compositions including extremely large sections of parallel interleavings even demonstrate reductions between 61 and 88 percent.

Although the resulting interleaving is responsible for a state explosion, this is of little concern for compositions with average and even large parallel areas. Normal sized compositions are generated and ready to be verified instantly. Extremely large parallel areas do introduce increased complexity. However, when a limited number of atomic propositions from these areas is used, these still can be reduced significantly and used for verification pre-runtime. Furthermore, when a model does turn out to be too large for model checking, our approach allows to split formulas into multiple sets, each resulting in a much smaller reduced Kripke structure. Each formula set can then be checked on its respective Kripke reduction, which results in a significant performance gain. In this respect, the size of the reduced model is directly related to the number of atomic propositions used within the set of formulas.

8 Conclusion

A novel approach to preventative compliance checking was presented. The presented approach goes beyond existing approaches, which either check compliance during or after execution, require new or extended logics unsupported by the verification community, or directly translate to the modeling language of a model checker causing large amounts of overhead. Instead of requiring unsupported logics to allow verification over the various branching behaviors of compositions, the presented approach captures the behavior in the model itself while limiting its size, allowing consistent and correct verification of control flow properties over different branching constructs using well-known branching time temporal logics.

The presented approach is independent of the design and implementation specification. The application of CPN and a workflow pattern mapping allows compositions described by different specifications to be represented as CPN. Any composition represented as a CPN through the application of the workflow patterns can subsequently be converted into a Kripke structure for verification through the application of the transition graph. Note, however, that although the transition graph represents the executions of CPN with great effect and expressive power, it does so with domain specific knowledge (i.e., the workflow patterns) and may be unsuitable for use within other domains.

Evaluations on expressive power demonstrate that, other than the generally employed transition systems, the transition graph correctly captures well-known business process and composition patterns. Furthermore, it maintains information on parallel service invocations and the local next service invocation: an ability which is unique to the presented approach.

Results of verification can be understood upon the possible executions of CPN, and similarly upon the possible executions of the original compositions. Although the interpretation of verification is formally presented through the branching-time temporal logic CTL, the transition graph can also be used in combination with the linear-time temporal logic LTL and their superset CTL*. Definitions and proofs regarding their semantics can be easily inferred by the reader from those given in Appendix B for CTL. The use of CTL* would, however, require the use of a different or updated model checker and related modeling language translation.

Evaluations on performance confirm that the conversion algorithm performs well, even for compositions with exceedingly large parallel regions. Moreover, very large compositions can be significantly reduced before actual verification. Furthermore, the approach allows to split formulas in multiple sets, each resulting in a much smaller reduced Kripke structure. Each formula set can then be checked on its respective Kripke reduction, which results in a significant performance gain. As such, the size of the reduced model is directly related to the number of atomic propositions used within the set of formulas.

The approach is particularly valuable in highly changeable environments. Although service-oriented environments do provide the required flexibility with respect to business process support, the automated compliance checking approach presented in this paper ensures that the control flow of adapted service compositions remain compliant with regulations.

For future work, we plan to assess the impact of the approach, by analysing a large set of real-life compositions and determine how often compositions are not compliant. Furthermore, we plan to evaluate the method with domain experts to assess the usefulness of the approach for identifying and resolving compliance issues in practical scenarios.

Appendix A

CTL Semantics

The semantics of CTL are defined on a Kripke structure M using the minimal set of CTL operators $\{\neg, \vee, EX, EG, EU\}$.

Definition 8 (Semantics of CTL). *$M, s_i \models \phi$ means that the formula ϕ holds at state s_i of the model M. When the model M is understood, $s_i \models \phi$ is written instead. The relation $\models$ is defined inductively as follows:*

$$
\begin{array}{lll}
s_i \models \top & \text{iff} & s_i \not\models \bot \\
s_i \models p & \text{iff} & p \in L(s_i) \\
s_i \models \neg\phi & \text{iff} & s_i \not\models \phi \\
s_i \models \phi \vee \phi' & \text{iff} & s_i \models \phi \text{ or } s_i \models \phi' \\
s_i \models EX\ \phi & \text{iff} & \exists (s_i, s_{i+1}) \in R \,|\, s_{i+1} \models \phi \\
s_i \models EG\ \phi & \text{iff} & \exists \pi = s_i, s_{i+1}, s_{i+2}, \ldots \,| \\
 & & \forall n : (n \geq 0 \wedge s_{i+n} \models \phi) \\
s_i \models E[\phi\ U\ \phi'] & \text{iff} & \exists \pi = s_i, s_{i+1}, s_{i+2}, \ldots \,| \\
 & & \exists m : (m \geq 0 \wedge s_{i+m} \models \phi' \\
 & & \wedge \forall n : (0 \leq n < m : s_{i+n} \models \phi))
\end{array}
$$

Further CTL operators can be obtained through the following equivalences:

$EF\phi \equiv E[true\ U\ \phi]$
$AF\phi \equiv \neg EG\neg\phi$
$AX\phi \equiv \neg EX\neg\phi$
$AG\phi \equiv \neg EF\neg\phi$
$A[\phi\ U\ \phi'] \equiv \neg(E[\neg\phi'\ U\ \neg(\phi \vee \phi')] \vee EG\neg\phi')$

APPENDIX B

CTL Interpretation

The results of verification using CTL can be interpreted upon the possible executions of the composition as described by the CPN. The semantics of CTL on the possible executions of a CPN (i.e., its reachability graph) using the minimal set of CTL operators $\{\neg, \vee, EX, EG, EU\}$ is defined as follows:

Definition 9 (CTL semantics on Reachability Graph). *$G, y_i \models \phi$ means that the formula ϕ holds at $y_i \in Y_x(M_i)$ of marking M_i of the reachability graph G. When the model G is understood, $y_i \models \phi$ is written instead. The steps (t, b) form the propositions of the language of CTL. When b is understood, t is written only. The relation $\models$ is defined inductively as follows:*

$$
\begin{array}{lll}
y_i \models (t, b) & \text{iff} & (t, b) \in y_i \\
y_i \models \neg\phi & \text{iff} & y_i \not\models \phi \\
y_i \models \phi \vee \phi' & \text{iff} & y_i \models \phi \text{ or } y_i \models \phi' \\
y_i \models EX\ \phi & \text{iff} & \exists \pi = y_i, y_{i+1}, \ldots \,|\, y_{i+1} \models \phi \\
y_i \models EG\ \phi & \text{iff} & \exists \pi = y_i, y_{i+1}, y_{i+2}, \ldots \,| \\
 & & \forall n : (n \geq 0 \wedge y_{i+n} \models \phi) \\
y_i \models E[\phi\ U\ \phi'] & \text{iff} & \exists \pi = y_i, y_{i+1}, y_{i+2}, \ldots \,| \\
 & & \exists m : (m \geq 0 \wedge y_{i+m} \models \phi' \\
 & & \wedge \forall n : (0 \leq n < m : y_{i+n} \models \phi))
\end{array}
$$

Let Φ be a set of formulas. $G, y_i \models \Phi$ iff for each $\phi \in \Phi$ it holds that $G, y_i \models \phi$. Then, a set of formulas Φ is consistent if $\Phi \not\vdash \bot$.

Lemma 1 (Lindenbaum's Lemma). *For each consistent set Φ, there is a maximally consistent set Φ' such that $\Phi \subseteq \Phi'$. In other words, every consistent set Φ can be extended to a maximally consistent set.*

Lemma 2 (Truth Lemma). *For any CTL formula ϕ: $G, y_i \models \phi$ iff $\phi \in y_i$.*

Proof. The proof is by structural induction on ϕ. If ϕ consists of a propositional letter (t, b) then by Definition 9. If ϕ is in the form $\phi_1 \vee \phi_2$ then by Definition 9 $y_i \models \phi_1$ or $y_i \models \phi_2$, and $\phi_1 \in y_i$ or $\phi_2 \in y_i$. If ϕ is of the form $\neg\phi$, then $y_i \not\models \phi$, and $\phi \notin y_i$. If ϕ is of the form $EX\phi$, then $\exists \pi = y_i, y_{i+1}, \ldots \,|\, y_{i+1} \models \phi$, and thus $\phi \in y_{i+1}$. If ϕ is of the form $EG\phi$, then $\exists \pi = y_i, y_{i+1}, y_{i+2} \ldots \,| \forall n : (n \geq 0 \wedge y_{i+n} \models \phi)$, and thus $\forall n : (n \geq 0 \wedge \phi \in y_{i+n})$. The case where ϕ is of the form $E[\phi_1\ U\ \phi_2]$ follows similarly. □

Theorem 1. *Every maximally consistent set Φ has a model, i.e., there is a model G and state y_i such that for all $\phi \in \Phi$, $G, y_i \models \phi$.*

Proof. Suppose that Φ is a consistent set. By the Lindenbaum's Lemma, there is a maximally consistent set Φ' such that $\Phi \subseteq \Phi'$. Then, by the Truth Lemma, for each $\phi \in \Phi'$, we have $G, \Phi' \models \phi$. Then, every formula in Φ is true at Φ' in the graph. □

Theorem 2. *If $\Phi \models \phi$ then $\Phi \vdash \phi$.*

Proof. By reductio ad absurdum, suppose that $\Phi \not\vdash \phi$. Then, $\Phi \cup \{\neg\phi\}$ is consistent. By the above theorem, there is a model of $\Phi \cup \{\neg\phi\}$. Hence, $\Phi \not\models \phi$. □

Verification of a formula ϕ on the possible executions of a CPN proves that ϕ does or does not hold at a certain point of its execution. More specifically, a formula ϕ may or may not hold at a version $y_i \in Y_x(M_i)$ of marking M_i (Definition 5). When a step (t, b) holds at y_i, that step is occurring. When a formula ϕ holds at all versions $y_i \in Y_x(M_i)$ of a marking M_i, it can be written that $M_i \models \phi$.

An occurrence path of a CPN is a sequence of sets of enabled transitions that can occur concurrently $\pi = y_1, y_2, \ldots$ with $y_i \in Y_x(M_i)$ and $M_i \xrightarrow{(t_i, b_i) \in y_i} M_{i+1}$ for $i > 0$.

Here, $y_i \in Y_x(M_i)$ (Definition 5) are versions of the marking M_i where different sets of binding elements are enabled (i.e., those that can occur simultaneously). A binding element (t, b) is occurring at y_i iff $(t, b) \in y_i$.

ACKNOWLEDGMENTS

We thank Doina Bucur, Artem Polyvyanyy, and Mustafa Hashmi for their feedback, and especially thank Guido Governatori for sharing his case study and feedback.

REFERENCES

[1] W. van der Aalst, "Workflow verification: Finding control-flow errors using petri-net-based techniques," in *Business Process Management*. Berlin, Germany: Springer, 2000, pp. 161–183.
[2] G. Governatori, Z. Milosevic, and S. Sadiq, "Compliance checking between business processes and business contracts," in *Proc. EDOC'06. 10th IEEE Int. Enterprise Distrib. Object Comput. Conf.*, 2006, pp. 221–232.
[3] P. Bulanov, A. Lazovik, and M. Aiello, "Business process customization using process merging techniques," in *Proc. IEEE Int. Conf. Serv.-Oriented Comput. Appl.*, 2011, pp. 1–4.
[4] S. Nakajima, "Verification of web service flows with model-checking techniques," in *Proc. Int. Symp. Cyber Worlds*, 2002, pp. 378–385.
[5] Y. Choi and J. L. Zhao, "Decomposition-based verification of cyclic workflows," in *Automated Technology for Verification and Analysis*. Berlin, Germany: Springer, 2005, pp. 84–98.
[6] S. Sadiq, M. Orlowska, and W. Sadiq, "Specification and validation of process constraints for flexible workflows," *Inf. Syst.*, vol. 30, no. 5, pp. 349–378, 2005.
[7] S. Feja, A. Speck, and E. Pulvermüller, "Business process verification," in *Proc. GI Jahrestagung*, 2009, pp. 4037–4051.

[8] H. Groefsema and N. van Beest, "Design-time compliance of service compositions in dynamic service environments," in *Proc. IEEE Int. Conf. Serv.-Oriented Comput. Appl.*, 2015, pp. 108–115.
[9] K. Jensen, "Coloured petri nets and the invariant method," *Theor. Comput. Sci.*, vol. 14, pp. 317–336, 1981.
[10] W. van der Aalst, A. T. Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, pp. 5–51, 2003.
[11] E. A. Emerson and J. Y. Halpern, "Decision procedures and expressiveness in the temporal logic of branching time," in *Proc. 14th Annu. ACM Symp. Theory Comput.*, 1982, pp. 169–180.
[12] E. Pulvermueller, S. Feja, and A. Speck, "Developer-friendly verification of process-based systems," *Knowl.-Based Syst.*, vol. 23, no. 7, pp. 667–676, 2010.
[13] A. Awad, G. Decker, and M. Weske, "Efficient compliance checking using BPMN-Q and temporal logic," in *Proc. 6th Int. Conf. Bus. Process Manage.*, 2008, pp. 326–341.
[14] S. Goedertier and J. Vanthienen, "Designing compliant business processes with obligations and permissions," in *Proc. Int. Conf. Bus. Proc. Manage. Workshops*, 2006, pp. 5–14.
[15] C. Gerede and J. Su, "Specification and verification of artifact behaviors in business process models," in *Proc. Serv.-Int. Conf. Oriented Comput.*, 2007, vol. 4749, pp. 181–192.
[16] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu, "Automatic verification of data-centric business processes," in *Proc. 12th Int. Conf. Database Theory*, 2009, pp. 252–267.
[17] A. Ghose and G. Koliadis, "Auditing business process compliance," in *Proc. 5th Int. Conf. Serv.-Oriented Comput.* 2007, pp. 169–180.
[18] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems," *Data Knowl. Eng.*, vol. 66, pp. 438–466, 2008.
[19] C. Favre and H. Vlzer, "Symbolic execution of acyclic workflow graphs," in *Proc. Business Process Management*, 2010, vol. 6336, pp. 260–275.
[20] M. Montali, P. Torroni, F. Chesani, P. Mello, M. Alberti, and E. Lamma, "Abductive logic programming as an effective technology for the static verification of declarative business processes," *Fund. Inf.*, vol. 102, no. 3-4, pp. 325–361, 2010.
[21] W. Janssen, R. Mateescu, S. Mauw, and J. Springintveld, "Verifying business processes using SPIN," in *Proc. 4th Int. Shar. Prog. Neonatology Workshop*, 1998, pp. 21–36.
[22] T. Latvala and K. Heljanko, "Coping with strong fairness," *Fundam. Inform.*, vol. 43, no. 1-4, pp. 175–193, 2000.
[23] R. Eshuis and R. Wieringa, "Tool support for verifying uml activity diagrams," *IEEE Trans. Softw. Eng.*, vol. 30, no. 7, pp. 437–447, Jul. 2004.
[24] B. Anderson, J. V. Hansen, P. Lowry, and S. Summers, "Model checking for E-business control and assurance," *IEEE Trans. Syst. Man Cybern.*, vol. 35, no. 3, pp. 445–450, Aug. 2005.
[25] D. Bianculli, C. Ghezzi, and P. Spoletini, "A model checking approach to verify BPEL4WS workflows," in *Proc. IEEE Int. Conf. Serv.-Oriented Comput. Appl.*, 2007, pp. 13–20.
[26] O. Kherbouche, A. Ahmad, and H. Basson, "Using model checking to control the structural errors in BPMN models," in *Proc. IEEE 7th Int. Conf. Res. Challenges Inform. Sci.*, 2013, pp. 1–12.
[27] A. Kheldoun, K. Barkaoui, and M. Ioualalen, "Specification and verification of complex business processes - a high-level petri net-based approach," in *Proc. Bus. Proc. Manage.*, 2015, vol. 9253, pp. 55–71.
[28] Y. Liu, S. Müller, and K. Xu, "A static compliance-checking framework for business process models," *J. IBM Syst.*, vol. 46, pp. 335–361, 2007.
[29] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based verification of web service compositions," in *Proc. 18th IEEE Int. Conf. Autom. Softw. Eng.*, 2003, pp. 152–163.
[30] J. Esparza, "Model checking using net unfoldings," in *Proc. Theory Practice Softw. Dev.*, 1993, vol. 668, pp. 613–628.
[31] P. Huber, A. M. Jensen, Li, L. O. Jepsen, and K. Jensen, "Reachability trees for high-level petri nets," *Theor. Comput. Sci.*, vol. 45, no. 3, pp. 261–292, 1986.
[32] OMG, "Business process model and notation beta 1 for version 2.0," Needham, MA, USA, 2009, http://www.omg.org/spec/BPMN/2.0/PDF
[33] R. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in bpmn," *Inform. Softw. Technol.*, vol. 50, no. 12, pp. 1281–1294, 2008.
[34] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.
[35] M. C. Browne, E. M. Clarke, and O. Grümberg, "Characterizing finite kripke structures in propositional temporal logic," *Theor. Comput. Sci.*, vol. 59, no. 1-2, pp. 115–131, Jul. 1988.
[36] L. Lamport, "What good is temporal logic?" in *Proc. Int. Fed. Inform. Process. Congr.*, vol. 83, 1983, pp. 657–668.
[37] J. F. Groote and F. W. Vaandrager, "An efficient algorithm for branching bisimulation and stuttering equivalence," in *Proc. Int. Colloquium Automata Lang. Program.*, 1990, pp. 626–638.
[38] R. Cavada, et al., "The nuxmv symbolic model checker," in *Int. Conf. Comput. Aided Verif.*, 2014, pp. 334–342.

Heerko Groefsema received the BA degree in information and communication technology at the Hanzehogeschool Groningen in 2004, and the BSc and MSc degrees in computer science at the University of Groningen in 2006 and 2008, respectively. He is working toward the PhD degree at the Distributed Systems group of the University of Groningen, The Netherlands. His research interests include business process and service composition compliance, verification, and variability.

Nick R. T. P. van Beest received the PhD degree in information systems from the University of Groningen, The Netherlands, in 2013. He is a researcher at Data61, CSIRO, in Australia. He is a visiting researcher at the Queensland University of Technology (Australia) and the University of Tartu (Estonia). His research experience covers artificial intelligence, process mining, business process compliance, knowledge-intensive business processes, and distributed systems.

Marco Aiello received the MSc degree in engineering from the University of Rome La Sapienza cum Laude and the PhD degree in logic from the University of Amsterdam. He is a professor of Distributed Systems at the University of Groningen (UG), head of the Distributed Systems unit, member of the board of the Johann Bernoulli Institute of Mathematics and Computer Science and member of the Board of the startup company Nerdalize BV. Before joining the UG he was a Lise Meitner fellow at the Technical University of Vienna (from which he obtained the Habilitation), and assistant professor at the University of Trento. He is a member of the IEEE.