Design-time Compliance of Service Compositions in Dynamic Service Environments

Heerko Groefsema
Johann Bernoulli Institute
Faculty of Mathematics and Natural Sciences
University of Groningen
h.groefsema@rug.nl

Nick van Beest
Software Systems Research Group
NICTA Queensland
nick.vanbeest@nicta.com.au

***Abstract*—In order to improve the flexibility of information systems, an increasing amount of business processes is being automated by implementing tasks as modular services in service compositions. As organizations are required to adhere to laws and regulations, with this increased flexibility there is a demand for automated compliance checking of business processes. Model checking is a technique which exhaustively and automatically verifies system models against specifications of interest, e.g. a finite state machine against a set of logic formulas. When model checking business processes, existing approaches either cause large amounts of overhead, linearize models to such an extent that activity parallelization is lost, offer only checking of runtime execution traces, or introduce new and unknown logics. In order to fully benefit from existing model checking techniques, we propose a mapping from workflow patterns to a class of labeled transition systems known as Kripke structures. With this mapping, we provide pre-runtime compliance checking using well-known branching time temporal logics. The approach is validated on a complex abstract process which includes a deferred choice, parallel branching, and a loop. The process is modeled using the Business Process Model and Notation (BPMN) standard, converted into a colored Petri net using the workflow patterns, and subsequently translated into a Kripke structure, which is then used for verification.**

I. Introduction

Laws and regulations are a common sight in all fields of business and government. Such regulations directly affect the way organizations conduct business. As such, business processes are increasingly supported by service-oriented information systems, in order to achieve higher flexibility, which allows them to anticipate on changing regulations. As the number of processes is increasing rapidly, it becomes significantly more complicated to continuously ensure the compliance of these business processes and the resulting new service compositions. As a result, organizations are becoming more and more interested in automatically checking the compliance of their business processes and service compositions.

Model checking is a technique to automatically verify a given system model against specifications of interest. To allow for algorithmic verification, a system model is commonly presented as a transition system and verified against a set of logic formulas. Model checking business processes models can be done for three purposes: monitoring (checking whether the model is executing correctly), auditing (checking whether the model has been executed correctly), or preventative (ensuring correctness of the model prior to its execution). Existing approaches tend to focus on monitoring and auditing, using the runtime execution trace of the business process [1]. Therefore, whenever a business process appears not to be compliant, its execution has already started, or could even have been completed. Consequently, rollbacks are required in order to undo any work not compliant with business rules. Design-time compliance verification, on the other hand, does not suffer from this disadvantage, as any discrepancy will be detected prior to execution. Existing design-time approaches, however, invent new or extended logics in order to support the different branching constructs implemented by business processes [2][3], generate transition systems with large amounts of overhead (e.g. [4]), or linearize the model to such an extent that parallelization information is lost [5][6][7]. Particularly in more complex systems, where parts of the same process are executed by different service providers, concurrency is an important aspect to be taken into account. However, the analysis of a large number of concurrent branches and activities in a business process quickly results in a state explosion in the underlying transition system.

Therefore, we present a novel approach allowing pre-runtime compliance checking that supports the different branching and merging constructs allowed by business process models, 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, as well as existing model checkers, can be applied during process verification.

First, a service composition is converted into a colored Petri net [8] (CPN) through the application of workflow patterns [9]. The resulting CPN is translated into a transition system known as a Kripke structure [10]. Although the Kripke structure closely resembles the well known reachability graph [11] (RG) of the CPN, it maintains parallelization information and allows correct specification of branching time temporal logics over transition occurrences. Prior to verification, the Kripke structure is reduced, resulting in a significant performance gain. Finally, properties over the possible sequential and concurrent service executions of the composition can be verified.

The resulting model 1) *allows correct interpretation of branching time temporal logic specifications over complex business process models*, 2) *provides full insight into possible parallel interleavings*, 3) *supports arbitrary cycles*, 4) *causes a limited state explosion compared to other approaches*, and 5) *allows further model reduction through equivalence with respect to stuttering*.

The paper is structured as follows. In Section II, we first introduce BPMN, CPN, and the one-on-one conversion between the two through workflow patterns. Next, in Section III, the conversion from CPN to Kripke structures is presented, the semantics of the branching time temporal logics is defined upon the CPN its possible executions, and the obtained model is normalized further. Next, in Section IV, we evaluate performance and the effects of model sizes for the conversion algorithm. In Section V, the related work is discussed. Finally, we conclude our work in Section VI.

II. Process Modeling

A. BPMN

In 2004, the Business Process Management Initiative introduced the Business Process Modelling Notation (BPMN). This business process modelling language has been developed with the specific purpose of providing a modelling language that is readily understandable by business users [12]. As such, the process flow is represented in a graph-oriented way, where the explicit control-flow is defined by events, activities, and gateways, which are connected through sequence flows and message flows [13][12]. In 2009, BPMN was updated to version 2.0, including detailed execution semantics for all BPMN elements [14].

The general process model, which is later converted for verification, is based on the control-flow perspective of the BPMN standard. In order to allow for formal verification, the processes defined in BPMN are translated into Colored Petri Nets [8] and subsequently to Kripke structures [15], to obtain the possible states of the process.

In Figure 1, an abstract process is depicted using BPMN, which we use to graphically describe the basic conversion. The abstract process comprises two exclusive branches, of which one contains a loop and the other comprises a parallel split.

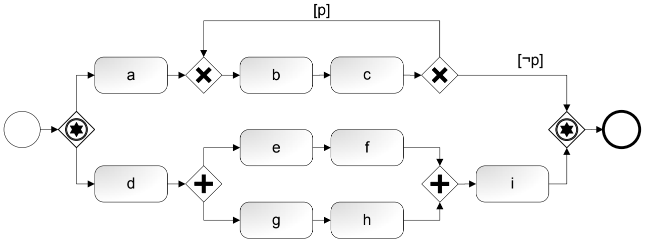


Fig. 1: The abstract process as BPMN.

B. Colored Petri Nets

A Colored Petri Net (CPN) is a directed graph representing a process. CPNs consist of places, transitions, and arcs between transition and place pairs. Transitions in CPNs represent activities or tasks of the business process, and places that can hold tokens, representing 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. Prior to illustrating the translation from BPMN to CPN, first a formal definition is provided of CPN and its reachability graph. A CPN is defined as follows [8]:

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 $M(p) \in [C(p) \to \mathbb{N}]_f$ 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 $M(p) \in [C(p) \to \mathbb{N}]_f$ 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 $(p,t) \in N$ $((t,p) \in N)$ [8]. 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* [8]. 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 activities of the business process composition (i.e. the transitions of the CPN).

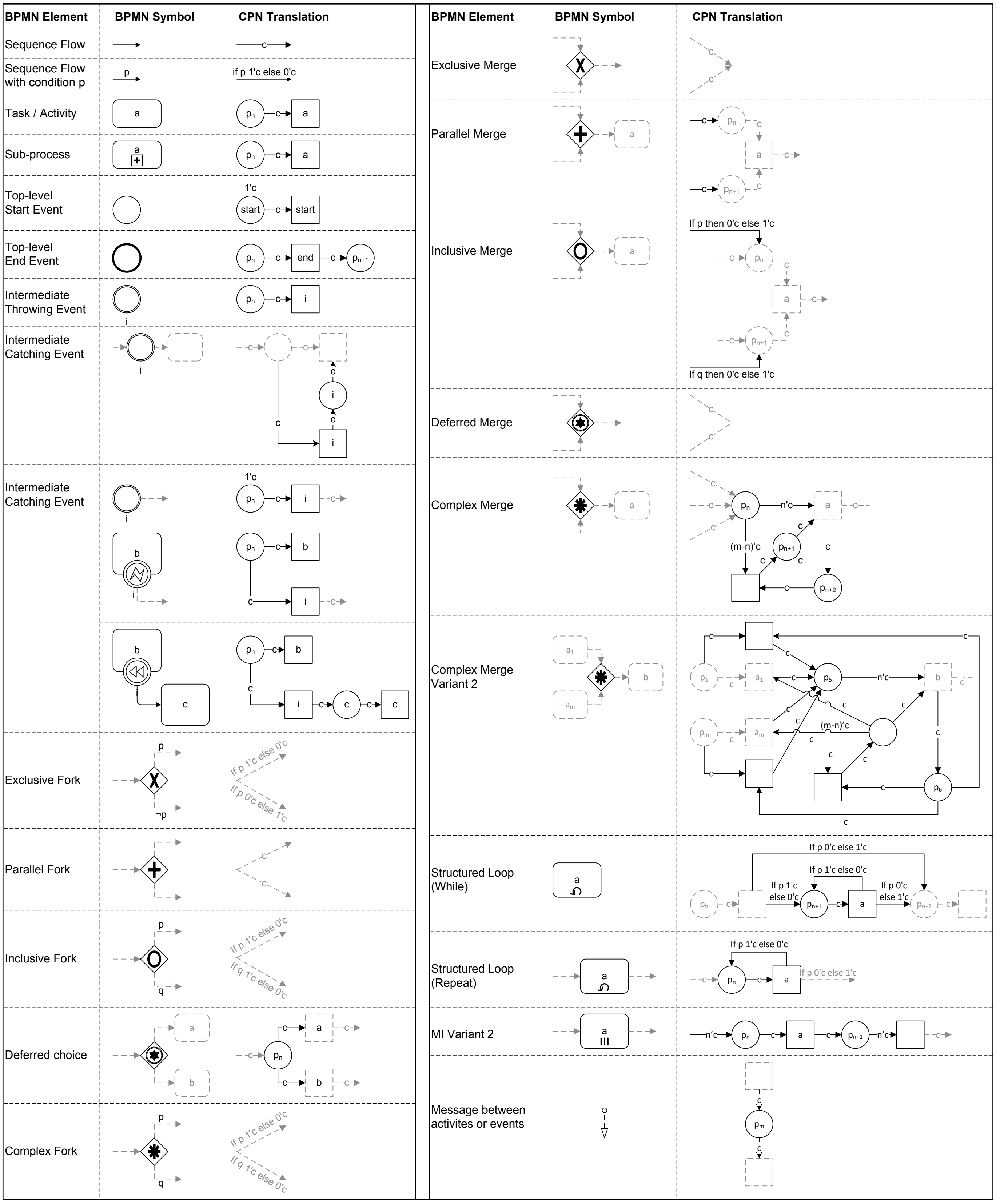
Definition 2 (Reachability Graph): The reachability graph of a CPN with markings $M_0, \ldots, M_n$ is a rooted directed graph $G = (V, E, v_0)$, where:

- $V = \{M_0, \ldots, M_n\}$ 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 (RG) (Definition 2) [11]. Starting from the initial marking M_0, states are created for each encountered marking while enabled binding elements occur to generate new markings. However, in this case, the occurrence of transitions can not be concluded from the states of the transition system but from its relations. States contain information on the marking of the net, that is they hold information on which places contain which tokens. When considering the states of the RG, the only certainty that can be concluded is that transitions are enabled at that marking – which does not ensure their occurrence – or, that one of a set of transitions *may* just have occurred to achieve the current marking. On the other hand, when verifying using the transitions labeled on the relations, the occurrence of the same transition at a marking could result in different markings due to conditional arcs (e.g. the exclusive choice workflow pattern). Verification of branching time temporal logics would then incorrectly consider both occurrences as being on different paths, where in actuality it is the same occurrence.

C. Converting from BPMN to CPN

For the conversion from a BPMN model to a CPN, we provide a translation for each BPMN element to its CPN representation. The translation is based on the workflow patterns as defined in [9]. However, in some cases, additions were required in order to provide a generic translation of the respective BPMN element. In Table I, an overview is presented of the conversion of BPMN elements to CPN constructs.



BPMN Element	BPMN Symbol	CPN Translation
Sequence Flow		c
Sequence Flow with condition p	p	if p 1'c else 0'c
Task / Activity	a	p_n c a
Sub-process	a	p_n c a
Top-level Start Event		1'c start c start
Top-level End Event		p_n c end c p_{n+1}
Intermediate Throwing Event	i	p_n c i
Intermediate Catching Event	i	c c c i c c i
Intermediate Catching Event	i	1'c p_n c i c
	b i	p_n c b c i c
	b c	p_n c b c i c c c c
Exclusive Fork	p ¬p	If p 1'c else 0'c If p 0'c else 1'c
Parallel Fork		c c
Inclusive Fork	p q	If p 1'c else 0'c If q 1'c else 0'c
Deferred choice	a b	c a c c p_n c b c
Complex Fork	p q	If p 1'c else 0'c If q 1'c else 0'c
Exclusive Merge		c c
Parallel Merge	a	c p_n c a c c p_{n+1} c
Inclusive Merge	a	If p then 0'c else 1'c c p_n c a c c p_{n+1} c If q then 0'c else 1'c
Deferred Merge		c c
Complex Merge	a	c c p_n n'c a c c c (m-n)'c p_{n+1} c c c c p_{n+2}
Complex Merge Variant 2	a_1 b a_m	c c c p_1 c a_1 c p_5 n'c b c c c c c (m-n)'c c p_m c a_m c c c c c c c p_6 c
Structured Loop (While)	a	If p 0'c else 1'c If p 1'c else 0'c If p 1'c else 0'c If p 0'c else 1'c p_n c p_{n+1} c a p_{n+2} c
Structured Loop (Repeat)	a	If p 1'c else 0'c c p_n c a If p 0'c else 1'c
MI Variant 2	a	n'c p_n c a c p_{n+1} n'c c
Message between activites or events		c p_m c

TABLE I: Conversion of BPMN elements into CPN constructs based on the workflow patterns as defined in [9].

In general, BPMN sequence flows are represented by arcs in the CPN and activities are represented by a place connected to a transition. In the table, the elements that are part of the construct are indicated with black lines, whereas the surrounding elements (depicted where necessary for clarity) are represented with grey dotted lines. This is necessary, because in some cases the translation is not represented by a separate construct in CPN. Rather, it affects the preceding or succeeding elements (e.g. consider the parallel merge). To avoid any unnecessary complexity, the patterns have been adopted to use one color (e.g. complex merge variants). At the same time, intermediate catching events have been changed to occur either once (or a set number of times) or when required in order to avoid an infinite number of possible markings. Naturally, these changes do not affect future labelings of states.

Using the conversion provided in Table I, the abstract BPMN model depicted in Figure 1 can be translated into a CPN. In Figure 2, the resulting CPN is depicted graphically.

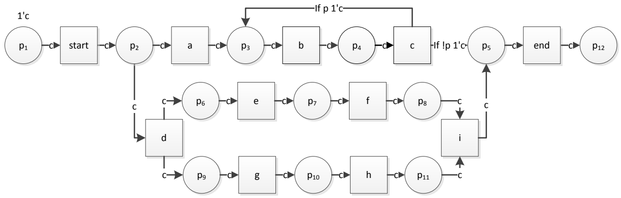


Fig. 2: The abstract process as CPN.

III. Design Time Business Process Verification

Before service compositions can be verified using model checking techniques, they first need to be translated from a CPN into a verifiable system model. The different models required for verification are introduced. Subsequently, the conversion process is presented, after which the resulting branching time temporal logic interpretation is defined on the reachability graph. Finally, model reduction is discussed.

A. Model and Specification

When model checking, a system model —often a transition system— is verified against specifications of interest. 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 [15].

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 $(s, s') \in R$.
- $L : S \rightarrow 2^{AP}$ 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 such as Computation Tree Logic (CTL) [10]. CTL is a branching time temporal logic which specifies temporal operators over future states on branching paths, or tree-like structures, where each branch represents a possible execution path. CTL pairs the operators E and A (exists/always) with the temporal operators X, F, G, and U to specify that a properly holds either on a path or all paths, and in the next state, eventually in a state, in all future states, or until another property holds, respectively. However, when considering concurrent systems –or in our case, concurrently executing branches– 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) [16]. 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. As such, the use of CTL-X (CTL minus the nexttime operator) is preferred in such a case.

Next, these definitions are used to present a correct model translation from CPN to Kripke structures, such that temporal logic formulas expressed using CTL-X can be verified upon the Kripke structure.

B. Verifiable Model

Since transitions in CPN relate directly to activities when describing business processes, 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 which only includes states for each marking and each set of transitions that are not just enabled, but will occur at that marking.

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. Let us define $Y_e(M)$ as the set of binding elements enabled at a marking M: $Y_e(M) = \{(t,b) \mid \forall p \in P : E(p,t)\langle b\rangle \leq M(p)\}$. Then, $Y_p(M)$ are the enabled steps of the powerset P of $Y_e(M)$ – or, more formally, $Y_p(M) = \{Y \mid Y \in P(Y_e) \wedge \forall p \in P : \sum_{(t,b)\in Y} E(p,t)\langle b\rangle \leq M(p)\}$. Finally, $Y_x(M)$ are those elements of the enabled powerset $Y_p(M)$ which are not subset of any other element of the powerset: $Y_x(M) = \{Y \mid Y \in Y_p(M) \wedge \forall Y' \in Y_p(M) : Y \not\subset Y' \wedge Y \neq \emptyset\}$. This set, $Y_x(M)$, is used in upcoming definitions to determine the different labelings when multiple sets of binding occurrences could occur concurrently at the same marking.

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 4.

Definition 4 (Transition Graph): Let AP be a set of atomic propositions. The transition graph of a CPN with markings $M_0, ..., M_n$ is a Kripke structure $K = (S, S_0, R, L)$ over AP, with:

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

[1] Although Definition 4 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.

Definition 4 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 $M_0,...,M_n$ 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 where only one functional binding $b = \langle c \rangle$ is used). Figure 3 depicts the transition graph resulting from this conversion process on the abstract CPN depicted in Figure 2.

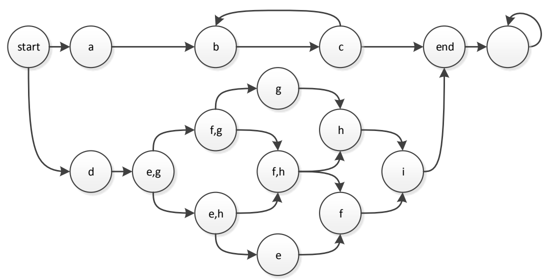


Fig. 3: The transition graph of the abstract process.

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 5 (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 5 completes the cycle of model conversions. Together with earlier definitions, Definition 5 allows a CPN to be transformed into a transition graph, which then can be transformed back into a CPN through an intermediary reachability graph step. Using these steps, the semantics of CTL-X can be defined upon the possible executions of a CPN. An occurrence path of a CPN is a sequence of sets of enabled transitions that can occur concurrently $\pi = y_1, y_2, ...$ 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, $y_i \in Y_x(M_i)$ 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 $(t,b) \in y_i$. The semantics of CTL-X on the possible executions of a colored Petri net is defined using the minimal set of CTL-X operators $\{\neg, \vee, EG, EU\}$:

Definition 6 (CTL-X 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-X. When b is understood, t is written only. The relation $\models$ is defined inductively as follows:

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

Lemma 1 (Truth Lemma): For any CTL-X 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 6. If ϕ is in the form $\phi_1 \vee \phi_2$ then by Definition 6 $y_i \models \phi_1$ 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 $EG\phi$, then $\exists \pi = y_i, y_{i+1}, y_{i+2} \ldots \mid \forall n : (n \geq 0 \wedge y_{i+n} \models \phi)$, 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. ■

The other well known CTL-X operators can be obtained through the following equivalences:

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

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. 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 $M_i \models \phi$.

Using the definitions above, CTL-X specifications can be used to verify BPMN service compositions. Compositions defined using BPMN can be translated into CPN, which in turn can be simulated to obtain a transition graph upon which the branching time temporal logic CTL-X can be interpreted. This interpretation can then be understood upon the possible executions of the CPN as expressed by its reachability graph. Next, further model reduction is discussed.

C. 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.

Two model reduction steps are available, both of which are based upon the removal of unused atomic propositions and model equivalence under the absence of the nexttime operator, otherwise known as equivalence with respect to stuttering [17]. 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. Instead, the until operator can be used on the transition graph to specify the same (e.g. $AG(e \Rightarrow A[e\ U\ f])$ can be used to specify that e is followed by f in every execution branch of the process).

A finite Kripke structure K can be uniquely identified by a single CTL formula F_K [17]. 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 [17]. 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 $\pi \sim_{st} \pi'$ 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 $i \geq 0$ [18].

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. Although the removed labels were 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 activities or events of the business process (i.e. transitions) and not its progression information (i.e. marking).

Figure 4 depicts the stutter equivalent model of the Kripke structure depicted in Figure 3 after the removal of the unused atomic propositions a, b, c, e, h, and i. Note that several unlabeled states remain. These can not be removed, as it would affect the evaluation of formulas (e.g. $AG(d \Rightarrow A[(d \vee f \vee g)\ U\ end])$ would incorrectly evaluate to true).

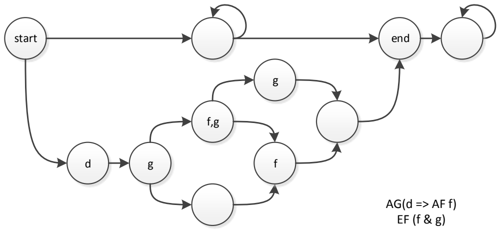


Fig. 4: The abstract process as an optimized Kripke structure w.r.t the atomic propositions d, f, and g.

IV. Performance Evaluation

The performance of the approach was evaluated by executing an implementation of Definition 4 on artificial service compositions of several sizes that were specifically generated for performance evaluation purposes by specifying a gate type, number of branches, and branch length. 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. The conversion algorithm was implemented using Java 7. The results of the performance tests can be found in Table II.

The columns of Table II provide information on the case number, the process (process containing sequence/exclusive/-parallel branching, the number of branches n, and number of activities per branch m), the Kripke structure (number of states S, relations R, and atomic propositions AP), the reduced Kripke structure (number of states S and percentage of original, relations R and percentage of original, and atomic propositions AP), and the performance of the conversion algorithm during initialization, model conversion, and model reduction. Note that post reduction sizes vary due to the random removal of 50% of atomic propositions.

Test cases 1-8 of Table II demonstrate that sequential processes and processes including exclusive paths are of no concern to performance. These processes are converted within 3 to 22 milliseconds and reduced within 0 to 5 milliseconds.

Compositions including parallel regions introduce an increased complexity of $\prod_{i=1}^{n}(m_i+1)$ states where m_i is the length of branch i (for equal branch lengths the complexity is $(m+1)^n$). This increased complexity is introduced due to the interleaving of concurrent activities on parallel branches. Other approaches, however, completely linearize the possible interleavings and therefore introduce a much larger complexity while proving limited insight on parallel behavior. Test cases 9-11 of Table II demonstrate that parallel interleavings of average sizes are converted within 96 milliseconds and reduced within 78 milliseconds. Increasing the length of the branches in test cases 13-14, and 18 of Table II from 5 to 50 activities increases the time to convert to 102 milliseconds with two branches and 185 seconds with four branches and increases the time to reduce to 90 milliseconds and 37 seconds.

The effect of model reduction displays varying results. Because sequential processes generate relatively simple Kripke structures, model reduction shows limited effects with reductions of 15% to 30%. For sequential processes, the worst case reduction with less than half of the AP removed is 0%. Processes with parallel interleaved paths, however, show a much larger effect with a 46% to 72% reduction. In this case, while the complexity of the Kripke structures increases with additional branches, model reduction naturally gains increased effect. With each removed atomic proposition, a significant amount of interleaved states is reduced.

Although the resulting interleaving is responsible for a state explosion, this is of little concern for processes 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 in multiple sets, each resulting in a much smaller reduced Kripke. 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.

V. Related Work

Processes have been the target of formal verification for a variety of reasons. In our survey [1], we identified the main goals of process verification. The first goal focuses on the verification of processes regarding the reachability and termination properties. When also considering the absence of any running activities at process termination, i.e. proper completion, we refer to process soundness. First presented in [19], process soundness is verified using Workflow nets. The technique is perfected in [20] by introducing support for OR-joins and cancellation regions. The second goal focuses on the verification of process compliance.

In [21], 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.

In [22], a framework for design-time process compliance of event-driven process chains using CTL is presented. The framework allows CTL constraints to be evaluated directly upon the process structure. However, as CTL is specified over Kripke structures, it does not support different forks and joins. In [23], a design-time compliance framework based upon annotated BPMN is presented. It includes the ability to

	Process			Kripke structure			Reduced Kripke structure			Performance results		
#	Type	n	m	$\|S\|$	$\|R\|$	$\|AP\|$	$\|S\|$ (% reduction)	$\|R\|$ (% reduction)	$\|AP\|$	Initialization	Conversion	Reduction
1	SEQ	1	5	8	8	7	6 (25%)	6 (25%)	4	0 ms	3 ms	0 ms
2	XOR	2	5	13	14	12	10 (23%)	11 (21%)	6	0 ms	3 ms	0 ms
3	XOR	3	5	18	20	17	15 (17%)	17 (15%)	9	1 ms	3 ms	0 ms
4	XOR	4	5	23	26	22	19 (17%)	22 (15%)	11	1 ms	3 ms	0 ms
9	AND	2	5	38	63	12	15 (61%)	22 (65%)	6	1 ms	4 ms	1 ms
10	AND	3	5	218	543	17	67 (69%)	148 (73%)	9	1 ms	27 ms	10 ms
11	AND	4	5	1298	4323	22	403 (69%)	1245 (71%)	11	1 ms	96 ms	78 ms
5	SEQ	1	50	53	53	52	37 (30%)	37 (30%)	26	1 ms	5 ms	2 ms
6	XOR	2	50	103	104	102	76 (26%)	77 (26%)	51	1 ms	12 ms	2 ms
7	XOR	3	50	153	155	152	114 (25%)	116 (25%)	76	2 ms	16 ms	4 ms
8	XOR	4	50	203	206	202	151 (26%)	154 (25%)	101	2 ms	22 ms	5 ms
13	AND	2	50	2603	5103	102	1409 (46%)	2741 (46%)	51	2 ms	102 ms	90 ms
14	AND	3	50	132653	390153	152	50823 (62%)	148319 (62%)	76	2 ms	1.853 ms	704 ms
18	AND	4	50	6765203	26530203	202	1915203 (72%)	7454605 (72%)	101	2 ms	184.758 ms	37.294 ms

TABLE II: Performance results of the initialization, conversion, and reduction of processes with n branches of m activities, and resulting Kripke structure sizes before and after reduction by 50% of randomly chosen atomic propositions.

automatically resolve non-compliance by converting BPMN models into semantic process models. It lacks, however, full loop support. A framework for a priori verification of ConDec [24] processes is presented by [25], where the LTL process specification of ConDec is translated into an abductive logic program and used as input for verification. However, the verification algorithm is unable to terminate under certain loops. In [26], two methods are presented for model checking compliance of annotated, yet acyclic, processes. Finally, [27] propose model checking compliance of Web Service Business Process Execution Language (WS-BPEL) processes by translating to pi-calculus and then to an automaton. However, due to the heavy synchronization of the interleaving method, all parallelization is lost. Instead, we presented a process conversion including parallel and loop support.

In order to solve issues with loops and different forks and joins, [3] proposes Temporal Process Logics (TPL), a modal propositional logic that is able to reason about possible process executions. A temporal deontic logic (PENELOPE) is introduced by [28], for specifying obligations and permissions over activities rather than propositions. In [29], a CTL based language ABSL is proposed for specifying life cycle properties of artifacts, while [30] proposes a first-order extension of LTL to verify all possible process executions of artifact-centric systems for compliance. However, by introducing new or extended logics the power of known and accomplished model checkers can not be exploited.

In [4], the authors propose model checking web service flow language (WSFL) collaborations using the SPIN model checker. The WSFL is encoded into Promela, the modeling language of SPIN, by mapping activities and transitions to Promela processes and channels. A method for model checking compliance through Amber processes using SPIN is proposed by [31]. Finally, [32] builds upon the earlier proposed BPMN-Q, a query language for BPMN. Constraints are translated from BPMN-Q to PLTL (Past-time LTL) and checked against all paths in the process in the form of sub-graphs after being reduced and translated to Petri nets and subsequently to the NuSMV2 input language. In [33], a conversion from BPMN to Workflow nets, a class of Petri nets, is proposed to allow for formal analysis. However, high-level CPNs are used to express workflow patterns [9]. As such, we propose a mapping from BPMN to CPN, in order to support a wide range of patterns expressed by BPMN that are not supported by workflow nets (e.g. exclusive choice). In addition, large amounts of overhead can be introduced without careful use of modeling languages. For example, in [4], it is reported that the intermediate Promela mapping causes a simple process of five activities and four transitions to be mapped to 201 states and 586 transitions in SPIN's internal state machine. Instead, we propose a careful conversion from CPNs to Kripke structures with a minimal amount of overhead.

In [34], compliance is modeled based on DecSerFlow [35], an LTL based declarative runtime specification for service compositions. By translating DecSerFlow rules into an extended form of event calculus, they are able to model compensation actions at runtime when a choreography violates compliance. In [36], the authors propose compliance constraints in the form of graphs. These Compliance Rule Graphs (CRG) specify compliance rules using the occurrence or absence of antecedents and consequences from a process. Event patterns, described by these CRG, are satisfied when the occurrences, absence of antecedents and consequences of process elements are matched by the execution trace of the process. By offering pre-runtime checking instead, we avoid the execution of non-compliant processes including any resulting roll-backs.

Another approach towards compliance verification is that of refinement checking. When refinement checking, a process is compliant when it is a refinement of rules expressed as another process. A method for model checking UML sequence diagrams is proposed by [37], aiming at process assurance and design. A process is implemented into the Communicating Sequential Processes (CSP) format and checked by the Failures-Divergences Refinement (FDR) checker against atomicity properties encoded as another CSP process. In [38], a BPEL implementation of a process is verified against a UML message state chart. Both BPEL and UML are translated into a Finite State Process (FSP), which are then verified towards each other. Reo, a channel based coordination language, is proposed by [39] as an intermediate layer for verifying compliance. BPMN is translated to Reo and verified using constraint automata. Instead, we allow well-known logics, such as CTL, to be used to describe compliance specifications.

VI. Conclusion

We presented a novel approach to pre-runtime compliance checking. The approach supports well-known branching time temporal logics over the different branching and merging constructs allowed in service compositions. As such, it goes beyond existing approaches, which either check compliance during or after execution, cause large amounts of overhead, or require new or extended logics.

In our approach, the compliance of a composition can be checked starting from a graphical business process modelling notation. Although BPMN is used throughout the paper, other notations can be translated into CPN form using similar pattern mappings. As such, the process is first translated into a CPN, and subsequently converted into a Kripke structure in such a way that the different branching and merging constructs allowed in compositions are maintained and verifiable using branching-time temporal logics. Furthermore, the original process can be reverse engineered from the Kripke structure, allowing the results from model checking to be directly applied upon the original process. The approach in itself is notation independent due to the formal intermediate CPN form and pattern mapping.

Extensive performance tests confirm that, even for processes with large parallel regions, the conversion algorithm performs well. Moreover, very large processes can be easily reduced further before verification.

Our approach is particularly valuable in highly changeable environments, where organizations are required to adhere to frequently changing laws and regulations. Although service-oriented environments do provide the required flexibility with respect to business process support, the automated compliance checking approach in this paper ensures that new service compositions are compliant with respective laws and regulations. For future work we plan to evaluate the approach on a large real-life case and compare the results with other approaches.

Acknowledgements

We thank Marco Aiello, Doina Bucur, and Artem Polyvyanyy for their valuable feedback. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council.

References

[1] H. Groefsema and D. Bucur, "A survey of formal business process verification: From soundness to variability," in *Proc. of the 3rd International Symposium on Business Modeling and Software Design*, 2013.

[2] G. Governatori, Z. Milosevic, and S. Sadiq, "Compliance checking between business processes and business contracts," in *Enterprise Distributed Object Computing Conference, 2006. EDOC'06. 10th IEEE International*. IEEE, 2006, pp. 221–232.

[3] P. Bulanov, A. Lazovik, and M. Aiello, "Business process customization using process merging techniques," in *Service-Oriented Computing and Applications (SOCA), 2011 IEEE Int. Conf. on*. IEEE, 2011, pp. 1–4.

[4] S. Nakajima, "Verification of Web service flows with model-checking techniques," in *Proc. Int. Symp. on 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*. Springer, 2005, pp. 84–98.

[6] S. Sadiq, M. Orlowska, and W. Sadiq, "Specification and validation of process constraints for flexible workflows," *Information System*, vol. 30, no. 5, pp. 349–378, 2005.

[7] S. Feja, A. Speck, and E. Pulvermüller, "Business process verification." in *GI Jahrestagung*, 2009, pp. 4037–4051.

[8] K. Jensen, "Coloured petri nets and the invariant method," *Theoretical Computer Science*, vol. 14, pp. 317–336, 1981.

[9] W. Van Der Aalst, A. Ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, pp. 5–51, 2003.

[10] E. A. Emerson and J. Y. Halpern, "Decision procedures and expressiveness in the temporal logic of branching time," in *Proc. of the 14th annual ACM symposium on Theory of computing*, 1982, pp. 169–180.

[11] 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.

[12] S. White, "Business process modeling notation (bpmn) version 1.0." 2004, business Process Management Initiative, BPMI.org.

[13] O. Kopp, D. Martin, D. Wutke, and F. Leymann, "On the choice between graph-based and block-structured business process modeling languages," in *Modellierung betrieblicher Informationssysteme (MobIS 2008)*, vol. 141, 2008, pp. 59–72.

[14] OMG, "Business process model and notation beta 1 for version 2.0," Needham, MA, USA, 2009.

[15] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. The MIT Press, Cambridge, Massachusetts and London, UK, 1999.

[16] L. Lamport, "What good is temporal logic?" in *IFIP congress*, vol. 83, 1983, pp. 657–668.

[17] 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.

[18] J. F. Groote and F. W. Vaandrager, "An efficient algorithm for branching bisimulation and stuttering equivalence," in *ICALP '90*, 1990, pp. 626–638.

[19] W. Van Der Aalst, "Workflow verification: Finding control-flow errors using petri-net-based techniques," in *Business Process Management*. Springer, 2000, pp. 161–183.

[20] M. T. Wynn, H. Verbeek, W. van der Aalst, A. ter Hofstede, and D. Edmond, "Business process verification - finally a reality!" *Business Process Management Journal*, vol. 15, no. 1, pp. 74–92, 2009.

[21] T. Latvala and K. Heljanko, "Coping with strong fairness," *Fundamenta Informaticae*, vol. 43, no. 1-4, pp. 175–193, 2000.

[22] 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.

[23] A. Ghose and G. Koliadis, "Auditing business process compliance," in *Proc. 5th Int. Conf. on Service-Oriented Computing*. Springer, 2007, pp. 169–180.

[24] M. Pesic and W. M. P. van der Aalst, "A declarative approach for flexible business processes management," in *Proc. Int. Conf. on Business Process Management Workshops*. Springer-Verlag, 2006, pp. 169–180.

[25] 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.

[26] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems," *Data and Knowl. Eng.*, vol. 66, pp. 438–466, 2008.

[27] Y. Liu, S. Müller, and K. Xu, "A static compliance-checking framework for business process models," *IBM Systems Journal*, vol. 46, pp. 335–361, 2007.

[28] S. Goedertier and J. Vanthienen, "Designing compliant business processes with obligations and permissions," in *Proc. Int. Conf. on Business Process Management Workshops*, ser. BPM. Springer, 2006, pp. 5–14.

[29] C. Gerede and J. Su, "Specification and verification of artifact behaviors in business process models," in *Service-Oriented Computing (ICSOC)*, ser. LNCS. Springer, 2007, vol. 4749, pp. 181–192.

[30] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu, "Automatic verification of data-centric business processes," in *Proc. 12th Int. Conf. on Database Theory*, ser. ICDT. ACM, 2009, pp. 252–267.

[31] W. Janssen, R. Mateescu, S. Mauw, and J. Springintveld, "Verifying business processes using SPIN," in *Proc. of the 4th Int. SPIN Workshop*, 1998, pp. 21–36.

[32] A. Awad, G. Decker, and M. Weske, "Efficient compliance checking using BPMN-Q and temporal logic," in *Proc. 6th Int. Conf. on Business Process Management*, ser. BPM '08. Springer, 2008, pp. 326–341.

[33] R. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in bpmn," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[34] F. Chesani, P. Mello, M. Montali, and P. Torroni, "Web services and formal methods," R. Bruni and K. Wolf, Eds. Springer, 2009, ch. Verification of Choreographies During Execution Using the Reactive Event Calculus, pp. 55–72.

[35] W. Aalst and M. Pesic, "Decserflow: Towards a truly declarative service flow language," in *Web Services and Formal Methods*, ser. Lecture Notes in Computer Science, M. Bravetti, M. Nez, and G. Zavattaro, Eds. Springer Berlin Heidelberg, 2006, vol. 4184, pp. 1–23.

[36] L. T. Ly, S. Rinderle-Ma, D. Knuplesch, and P. Dadam, "Monitoring business process compliance using compliance rule graphs," in *Proc. Confederated Int. Conf. On the move to meaningful internet systems - Volume I*, ser. OTM. Springer-Verlag, 2011, pp. 82–99.

[37] B. Anderson, J. V. Hansen, P. Lowry, and S. Summers, "Model checking for E-business control and assurance," *Systems, Man, and Cybernetics, IEEE Trans. on*, vol. 35, no. 3, pp. 445–450, 2005.

[38] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based verification of web service compositions," in *Proc. 18th IEEE Int. Conf. on Automated Software Engineering (ASE)*, 2003, pp. 152–163.

[39] F. Arbab, N. Kokash, and S. Meng, "Towards using Reo for compliance-aware business process modeling," in *Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2009, vol. 17, pp. 108–123.