Formal Verification of Declarative Specifications of BPs: DCR2CPN-based Approach

Ikram Garfatta[1], Kaïs Klai[2], and Walid Gaaloul[1]

[1] Institut Mines-Télécom, Télécom SudParis, SAMOVAR UMR 5157, Évry, France
[2] University Sorbonne Paris North, LIPN UMR CNRS 7030, Villetaneuse, France
ikram_garfatta@telecom-sudparis.eu

Abstract. Declarative business process models, notably Dynamic Condition Response (DCR) graphs, have gained traction as an alternative to traditional imperative approaches. However, adequate verification tools for such models remain scarce. This paper proposes a model-checking approach for DCR graphs verification using Coloured Petri Nets (CPN) as a pivot language. The transformation from DCR to CPN is proven to preserve semantics, ensuring the resulting model aligns with the original. Once transformed, the CPN model can be checked by any model checker to verify specific properties. We have automated this process via a C++ prototype that generates CPNs in the specification language of *Helena*, the model checker that we choose to leverage in our work.

Keywords: DCR · Business Process Management · Model Checking · Coloured Petri Nets · Temporal properties.

1 Introduction

The most commonly used Business Process Model representations have long relied on imperative workflow descriptions, such as BPMN [20], YAWL [2], and Petri Nets [1]. Such representations are a convenient choice when operations are set and organized in a well-structured manner, allowing designers to define the control flow by specifying *how* tasks are executed. However, when processes must operate under unforeseeable circumstances, determining the order of operations becomes complicated. This is due to imperative models' limitations in providing enough flexibility for ad hoc processes and suitable tools to manage uncertainty and dynamics. To address this, researchers have proposed declarative approaches for business process representation, which are more suitable in such cases [24]. In recent years, these approaches have gained traction among BP designers, especially in sectors requiring high workflow flexibility, such as healthcare and case management [22]. Declarative representations succeed by implicitly describing processes through rules and constraints that govern tasks without specifying a fixed sequential order. They focus on *what* should be enacted rather than *how*. ConDec [22] and DecSerFlow [3] were among the first declarative languages proposed for process modeling, both supported by DECLARE [23], a prototype workflow management system. In this work, we are

interested in Dynamic Condition Response (DCR) graphs [12] as a means for business process representation as this language was proposed to improve on the execution efficiency problems encountered in the former languages [4].

The verification phase is crucial in the business process (BP) life cycle to ensure it meets specified requirements and avoids execution errors. This topic has been extensively studied for imperative representations, as highlighted in the survey conducted in [17]. BPMN, one of the most widely used modeling languages for BPs, has inspired numerous studies focusing on the formal verification of various aspects of BPMN models [8,21,30,26,10,6]. These studies frequently employ Petri nets as a formalism to enhance BPMN semantics. For example, in [8], the authors propose a mapping of BPMN's core elements into labeled Petri net patterns, which was implemented in a tool that automates Petri net generation and assists semantic analysis. Another approach in [21] analyzes BPMN processes using Colored Petri nets [15] as the target formal representation, using a modified BPEL4WS representation as a pivot language. YAWL, also based on Petri nets, extends the formalism to model complex workflows. An automation of transforming BPMN models into YAWL was proposed in [30] for verification using tools like Woflan [29].

These foundational studies marked the beginning of BP verification research, with subsequent studies using extensions of Petri nets to accommodate more complex models [16] and focusing on specific aspects, as in [5,14]. Other formal methods for BP verification include π-calculus [26], event calculus [10], and theorem proving [6].

Similar to the imperative languages of BP modelling, the emergence of the declarative modelling mode needs to be backed up by suitable tools that would allow the verification of its models. In fact, the implicit nature of the representation of the workflow makes it less obvious to be interpreted by the designers and therefore makes it easier for simple modelling errors to pass unnoticed. Moreover, the dynamic and complex nature of the domains that require flexible models puts more at stake and adds to the importance of having a verification tool that would insure the correctness of the process model. However, compared to the rich state of research on imperative business process model verification, there is a notable scarcity of studies on declarative model verification. To address this gap, we propose a model checking approach for DCR graphs using Coloured Petri Nets as a pivot language. We focus on DCR graphs because they offer simplicity and accessibility, with a graphical notation that enhances interpretability compared to models like EM-BrA^{2}CE [11], and fewer rule types than similar representations such as DECLARE [23], while maintaining expressiveness [28].

This expressive power presents challenges for formal verification, as it must manage the complexity of declarative models, which may have conflicting goals or requirements, leading to potential inconsistencies. Our approach involves defining a CPN model (CPN4DCR) that represents a DCR graph with equivalent semantics, allowing for state-based and event-based temporal property verification. We also propose an extension to model DCR choreographies, ensuring correctness through temporal property verification using the *Helena* [9] model

checker. The semantic equivalence between DCR graphs and our CPN model has been proven, with the translation implemented via our *DCR2CPN* tool[3].

It is worth noting that the closest approach that could be compared to ours was proposed in [7], where the authors transform DCR graphs into safe Petri nets with inhibitor arcs and read arcs to facilitate formal verification. However, the use of inhibitor arcs can be simplified by using complementary places, especially given that safe Petri nets are inherently bounded. Additionally, while the inclusion of read arcs aims to support concurrency, this feature is not practically implemented in their transformation since the model checker TAPAAL that they use does not support it. Their transformation often results in complex Petri net models, necessitating pruning to manage the size of the reachability graphs. This pruning complicates the verification process, as their bisimilarity proof, which ensures semantic equivalence between DCR and Petri nets, is conducted before pruning, leaving no guarantee of bisimilarity post-pruning. In contrast, our approach offers greater expressiveness since it not only supports the transformation of DCR choreographies but is also more extensible, allowing for future incorporation of additional DCR features such as time constraints, sub-processes, and data elements.

The paper is structured as follows: Section 2 defines key concepts (DCR, CPN, LTL). Section 3 introduces a running example. Section 4 presents our main contribution for formal modeling and verification of DCR graphs. Section 5 details the implementation of *DCR2CPN*. Finally, Section 6 concludes and discusses future directions.

2 Background

2.1 DCR Graphs and Choreographies

We start by defining DCR graphs before giving the definition of DCR choreographies.

Definition 1. *A DCR graph is a tuple* $Gr = (E, M, Act, \rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%, \rightarrow\diamond, l)$ *where* $\mathcal{M}(Gr) = (2^E \times 2^E \times 2^E)$ *is the set of all markings:*

1. E *is the set of events, ranged over by* e.
2. $M \in \mathcal{M}(Gr)$ *is the marking of the graph (explained below).*
3. Act *is the set of actions.*
4. $\rightarrow\bullet, \bullet\rightarrow \subseteq E \times E$ *are the condition and response relations, respectively. In general,* e *is a condition for* e' *(*$e \rightarrow\bullet e'$*) means that* e *must have been executed at least once before executing* e'*. Having* e' *as a response for* e *(*$e\bullet\rightarrow e'$*) means that* e' *should be executed at least once after having executed* e.
5. $\rightarrow+, \rightarrow\% \subseteq E \times E$ *are the dynamic include and exclude relations, respectively, satisfying that* $\forall e \in E \,.\, e \rightarrow+ \cap\, e \rightarrow\% = \emptyset$.
6. $\rightarrow\diamond \subset E \times E$ *is the milestone relation.*
7. $l : E \rightarrow Act$ *is a labelling function mapping every event to an action.*

[3] https://github.com/Sol2CPN/DCR2CPN

DCR Graph: semantics A marking $M = (Ex, Re, In) \in \mathcal{M}(Gr)$ is a triplet of event sets where Ex represents the set of events that have previously been executed, Re the set of events that are pending responses required to be executed or excluded, and In the set of events that are currently included. The idea conveyed by the dynamic inclusion/exclusion relations is that only the currently included events are considered in evaluating the constraints. In other words, if e is a condition for e' ($e \rightarrow\bullet e'$), but is excluded from the graph then it no longer restricts the execution of e'. Moreover, if e' is the response for e ($e\bullet\rightarrow e'$) but is excluded from the graph, then it is no longer required to happen for the flow to be acceptable. The inclusion relation $e \rightarrow+ e'$ (resp. exclusion relation $e \rightarrow\% e'$) means that, whenever e is executed, e' becomes included in (resp. excluded from) the graph. The milestone relation is similar to the condition relation in that it is a blocking one. The difference is that it is based on the events in the pending response set. In other words, if e' is a milestone of e ($e' \rightarrow\diamond\ e$), then e cannot be executed as long as e' is in Re.

DCR choreography: syntax A DCR choreography is a DCR graph that can be executed in a distributed way. An event in a DCR choreography has an *initiator* and can potentially have one or more *receivers*. In the following, we adapt the definition given in [13] on account of simplicity and better adequacy with our work.

Definition 2. *A DCR choreography is a couple* (Gr, R) *where* R *is a set of roles and* Gr *a DCR graph whose labelling function* l *is defined as follows:* $l : E \rightarrow (Act \times R \times \mathcal{P}(R))$

DCR choreography: semantics A DCR choreography has the same semantics as a DCR graph with the added condition that only the *initiator* of an event can execute it. For more details on DCR Graphs we refer the readers to [18].

An example of a distributed DCR graph is given in figure 1. Visually, such a model can be represented as a directed graph with events (boxes) as nodes and five types of arrows for the five types of relations that can link them. We use the upper part of a box to indicate the *initiator* of the event and the bottom part to indicate the *receiver(s)*.

2.2 Coloured Petri Nets

A Petri net [19] is a formal model with mathematics-based execution semantics. It is a directed bipartite graph with two types of nodes: places (drawn as circles) and transitions (drawn as rectangles). Despite its efficiency in modelling and analysing systems, a basic Petri net falls short when the system is too complex, especially when representation of data is required. To overcome such limitations, *Coloured Petri nets* [15] were proposed as an extension equipping the tokens with colours or types allowing them to hold values.

The formal definition of CPN's syntax is given in Definition 3.

Definition 3 (Coloured Petri net). *A* Coloured Petri Net *is a nine-tuple* $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, *where:*

1. P *is a finite set of* places.
2. T *is a finite set of* transitions *such that* $P \cap T = \emptyset$.
3. $A \subseteq (P \times T) \cup (T \times P)$ *is a set of directed* arcs.
4. Σ *is a finite set of non-empty* colour sets.
5. V *is a finite set of* typed variables *such that* $Type[v] \in \Sigma$, $\forall v \in V$.
6. $C : P \to \Sigma$ *is a* colour set function *that assigns a colour set to each place.*
7. $G : T \to EXPR_V$, *where* $EXPR_V$ *is the set of expressions with variables in* V, *is a* guard function *that assigns a guard to each transition* t.
8. $E : A \to EXPR_V$ *is an* arc expression function *that assigns an arc expression to each arc* a *such that* $Type[E(a)] = C(p)_{MS}$ *(i.e., the type of an expression corresponds to the multiset of the colour of the arc's place).*
9. $I : P \to EXPR_\emptyset$ *is an* initialisation function *that assigns an initialisation expression to each place* p *such that* $Type[I(p)] = C(p)_{MS}$.

CPN: semantics For CPN $(P, T, A, \Sigma, V, C, G, E, I)$, we note:

- A *marking* is a function M that maps each place into a multiset of tokens.
- The *initial marking* M_0 is defined by $M_0(p) = \langle I(p) \rangle$ for all $p \in P$.
- The *variables of a transition* t are denoted by $Var(t) \subseteq V$.
- A *binding* of a transition t is a function b that maps each variable $v \in Var(t)$ into a value $b(v) \in Type[v]$. It is written as $\langle var_1 = val_1, ..., var_n = val_n \rangle$. The set of all bindings for a transition t is denoted $B(t)$.
- A *binding element* is a pair (t, b) such that $t \in T$ and $b \in B(t)$. The set of all binding elements $BE(t)$ for a transition t is defined by $BE(t) = \{(t, b) | b \in B(t)\}$. The set of all binding elements in a CPN model is denoted BE.

For more details on CPN we refer readers to [15].

2.3 Linear Temporal Logic

The approach presented in this paper is primarily based on model checking of CPN models w.r.t formulae expressed in Linear Temporal Logic (LTL). This logic was first introduced in [25] as a means to reason about concurrent programs.

In LTL, a classical timeline that starts "now" is considered, where every moment has a unique possible future. In other words, a model of LTL is an infinite sequence of indexed states ($i = 0, 1, 2, ...$) where each point in time has a unique successor. An LTL formula is evaluated over such a sequence of states starting from an i'th state. It contains a finite set $Prop$ of atomic propositions, the usual Boolean operators $\neg$, $\wedge$, $\vee$, and $\rightarrow$, in addition to temporal operators:

- Until ($\mathcal{U}$): $\varphi \, \mathcal{U} \, \psi$ is true if ψ is true *now* or φ is true *now* and remains so until ψ holds.
- Next ($\mathcal{X}$ or $\bigcirc$): $\mathcal{X} \, \varphi$ is true if φ is true in the next step.
- Globally ($\mathcal{G}$ or $\Box$): $\mathcal{G}\varphi$ is true if φ is true in every step.
- Future ($\mathcal{F}$ or $\Diamond$): $\mathcal{F} \, \varphi$ is true if φ is true now or in some future time step.

3 Running Example

In this section, we present a running example inspired from the BPMN choreography in [27], through which we will be explaining our proposed approach and the different notions therein used. We will also use this example to discuss the nature of the properties that we are interested in verifying. Our example is de-

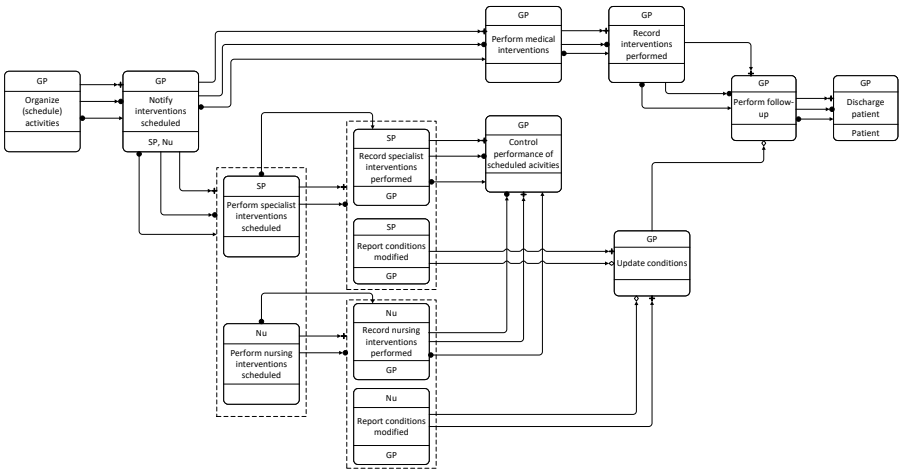


Fig. 1: IHC DCR choreography

picted by the DCR choreography shown in Figure 1, and represents a simplified process for care provision in an Integrated Home Care (*IHC*). Three participants take part in this choreography, namely a *General Practitioner* (*GP*) who is typically a physician in charge of the patient, a *Specialist Physician* (*SP*) who is a doctor certified in a specific area of medical practice and a Nu*rse* (*Nu*) who is a licensed general health-care provider. For lack of space, we refer the readers to [27] for a more details on this example.

4 A CPN Model for DCR Choreographies

As was mentioned in Section 1, the approach that we propose for the verification of DCR graphs is based on their transformation into CPN models that hold the same semantics. To this end, we start by presenting, in Section 4.1 a CPN model that allows the verification of state-based LTL properties. This models is then extended in Section 4.2 to allow the verification of both state- and event-based LTL properties. Finally, we present in Section 4.3 a second extension to our model that consists in a transformation of a DCR choreography by taking into account the concept of *roles*.

4.1 CPN4DCR - Initial Model

Definition 4 (CPN4DCR). *Given a DCR graph* $Gr = (E, M, Act, \rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%, \rightarrow\diamond, l)$*, a corresponding CPN model* $CPN4DCR = (P, T, A, \Sigma, V, C, G, E, I)$ *(depicted in figure 2a) is defined s.t.:*

- $P = \{S\}$*, where* S *is the single place of the model representing its state*
- $T = \{t_i, \forall i \in [1, n]\}$*, with* $n = |E|$ *the number of events in* Gr
- $A = \{(t_i, S), \forall t_i \in T\} \cup \{(S, t_i), \forall t_i \in T\}$
- $\Sigma = \{C_E, (C_E \times C_E \times C_E)\}$*, s.t.,* C_E *is defined as a set of integers type s.t., an event* $e_i \in E$ *is represented in* C_E *by its index* i*.*
- $V = \{Ex, Re, In, Ex', Re', In'\}$*, with* $Type[v] = C_E, \forall v \in V$
- $C = \{S \rightarrow (C_E \times C_E \times C_E)\}$
- $G = \{t_i \rightarrow guard_i, \forall i \in [1, n]\}$*, with* $n = |E|$
- $E = \{(S, t_i) \rightarrow \langle Ex, Re, In\rangle, \forall (S, t_i) \in A \cap (P \times T)\} \cup \{(t_i, S) \rightarrow \langle Ex', Re', In'\rangle, \forall (t_i, S) \in A \cap (T \times P)\}$ *with*
  - $Ex' = Ex \cup \{i\}$,
  - $Re' = (Re \backslash \{i\}) \cup \{j, \forall e_j \in e_i\bullet\rightarrow\}$ *and*
  - $In' = (In \cup \{j, \forall e_j \in e_i \rightarrow+\}) \backslash \{j, \forall e_j \in e_i \rightarrow\%\}$
- $I = \{S \rightarrow \langle S_1, S_2, S_3\rangle\}$ *with* $\langle S_1, S_2, S_3\rangle$ *the initial marking* M *of* Gr

For all $t_i \in T$ *representing an event* e_i *in the DCR graph, we further precise:*

- $guard_i$ *is the conjunction of the conditions defining the enabling of the corresponding event* e_i*:*
  1. $i \in In$,
  2. $(\{j, \forall e_j \in \rightarrow\bullet e_i\} \cap In) \subseteq Ex$ *and*
  3. $\{j, \forall e_j \in \rightarrow\diamond e_i \cap In\} \subseteq (E \backslash Re)$
- *the expression* $\langle Ex', Re', In'\rangle$ *on its output arc is defined such that:*
  1. $Ex' = Ex \cup \{i\}$,
  2. $Re' = (Re \backslash \{i\}) \cup \{j, \forall e_j \in e_i\bullet\rightarrow\}$ *and*
  3. $In' = (In \cup \{j, \forall e_j \in e_i \rightarrow+\}) \backslash \{j, \forall e_j \in e_i \rightarrow\%\}$

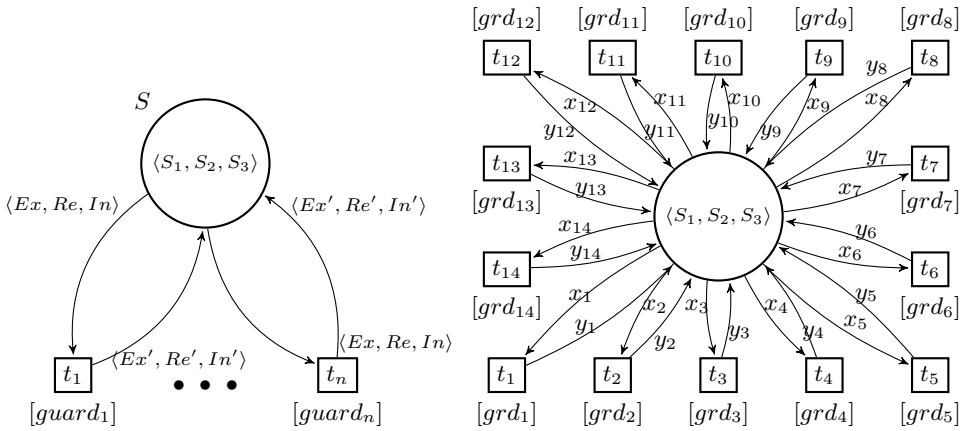
Applying this definition on the DCR graph that would be obtained if we were to omit the concept of *roles* from our choreography in figure 1 is shown in figure 2b.

Definition 5 (Marking Equivalence). *A marking* $M^{Gr} = \langle Ex, Re, In\rangle$ *of a DCR graph* Gr *is said to be equivalent to a marking* $M^C = \langle S \rightarrow \langle S_1, S_2, S_3\rangle\rangle$ *of a CPN model* C *iff*

- $\forall e_i \in Ex$ *(respectively* Re *and* In*),* $\exists i \in S_1$ *(respectively* S_2 *and* S_3*), and*
- $\forall i \in S_1$ *(respectively* S_2 *and* S_3*),* $\exists e_i \in Ex$ *(respectively* Re *and* In*)*

We note $M^{Gr} \equiv M^C$*.*

Definition 6 (Execution Sequence Equivalence). *An execution sequence of length* k*,* $\sigma_k^{Gr} = \langle e_i, ..., e_j\rangle$ *of a DCR graph* Gr *is said to be equivalent to an execution sequence of length* k*,* $\sigma_k^C = \langle t_i, ..., t_j\rangle$ *of a CPN model* C *iff*



(a) Initial CPN4DCR definition (b) Initial CPN4DCR of the IHC example

Fig. 2: Initial CPN4DCR

– $(M_1^{Gr} \equiv M_1^C \wedge M_1^{Gr} \xrightarrow{\sigma_k^{Gr}} M_2^{Gr} \wedge M_1^C \xrightarrow{\sigma_k^C} M_2^C) \implies M_2^{Gr} \equiv M_2^C$

We note $\sigma_k^{Gr} \equiv \sigma_k^C$.

Theorem 1. *Let Gr be a DCR graph and C the corresponding CPN model generated by following definition 4, then Gr and C are semantically equivalent.*

Proof. Let Gr be a DCR graph and C the corresponding CPN model generated by following definition 4. In order to prove that Gr and C are semantically equivalent we need to prove that

1. $\forall \sigma_k^{Gr} = \langle e_1, ..., e_k \rangle, \exists \sigma_k^C = \langle t_1, ..., t_k \rangle$, and
2. $\forall \sigma_k^C = \langle t_1, ..., t_k \rangle, \exists \sigma_k^{Gr} = \langle e_1, ..., e_k \rangle$

such that $\sigma_k^{Gr} \equiv \sigma_k^C$, $\forall k \in [1, m]$ with m the length of the longest execution sequence. We start by proving *(1)*:

– Let $P(n)$ be the statement: $\forall \sigma_n^{Gr} = \langle e_1, ..., e_n \rangle, \exists \sigma_n^C = \langle t_1, ..., t_n \rangle$ such that $\sigma_n^{Gr} \equiv \sigma_n^C$.
– $P(1) : \forall \sigma_1^{Gr} = \langle e_1 \rangle, \exists \sigma_1^C = \langle t_1 \rangle$ such that $\sigma_1^{Gr} \equiv \sigma_1^C$. This can be derived from Definition 4. In fact, the initial marking of C (M_0^C) being defined as equivalent to that of Gr (M_0^{Gr}), and the guard of each transition $t_i \in T$ being defined as to correspond to the enabling conditions of the relative event $e_i \in E$, we can deduce that the set of fireable transitions ($M_0^C \rightarrow$) corresponds to the set of enabled events ($M_0^{Gr} \rightarrow$). Additionally, the marking M_i^C obtained by firing t_i is equivalent to that obtained by executing e_i ($M_i^C \equiv M_i^{Gr}$) since the elements of M_i^C are defined as to correspond to the effect of the execution of e_i in Gr.

– Assume that $P(k) : \forall \sigma_k^{Gr} = \langle e_1, ..., e_k \rangle, \exists \sigma_k^C = \langle t_1, ..., t_k \rangle$ such that $\sigma_k^{Gr} \equiv \sigma_k^C$ is true for some $k \in [2, m-1]$. We will prove that $P(k+1) : \forall \sigma_{k+1}^{Gr} = \langle e_1, ..., e_{k+1} \rangle, \exists \sigma_{k+1}^C = \langle t_1, ..., t_{k+1} \rangle$ such that $\sigma_{k+1}^{Gr} \equiv \sigma_{k+1}^C$ is true.

$$\sigma_{k+1}^{Gr} \equiv \sigma_{k+1}^C \implies \exists e_{k+1} \in E, t_{k+1} \in T \text{ such that } \sigma_k^{Gr} \cdot e_{k+1} \equiv \sigma_k^C \cdot t_{k+1} \quad (1)$$

$$\sigma_k^{Gr} \equiv \sigma_k^C \iff (M_0^{Gr} \xrightarrow{\sigma_k^{Gr}} M_k^{Gr} \wedge M_0^C \xrightarrow{\sigma_k^C} M_k^C \wedge M_k^{Gr} \equiv M_k^C) \quad (2)$$

Analogously to the reasoning in the previous point, we can deduce that:

$$\begin{aligned} &\forall e_{k+1} \in E \text{ such that } M_k^{Gr} \xrightarrow{e_{k+1}} M_{k+1}^{Gr}, \\ &\exists t_{k+1} \in T \text{ such that } (M_k^C \xrightarrow{t_{k+1}} M_{k+1}^C \wedge M_{k+1}^{Gr} \equiv M_{k+1}^C) \end{aligned} \quad (3)$$

And therefore:

$$\forall \sigma_{k+1}^{Gr} = \langle e_1, ..., e_{k+1} \rangle, \exists \sigma_{k+1}^C = \langle t_1, ..., t_{k+1} \rangle \text{ such that } \sigma_{k+1}^{Gr} \equiv \sigma_{k+1}^C \quad (4)$$

The second part *(2)* is provable following a similar reasoning.

Discussion It is evident to see that a DCR graph is a model based on *events* or *tasks*, which in our corresponding CPN model we represent by transitions. It is therefore sensible to assume that the users would want to formulate the properties to be verified based on those tasks. Such properties can refer to *(i)* the execution of an event, which would translate into a property about the firing of its corresponding transition in the CPN model, or to *(ii)* the possibility of executing an event, which would translate into a property about the fireability of its corresponding transition. The nuance between the two cases might seem subtle, but the difference is rather plain when it comes to their expression in LTL. To explain this, we will refer to the simple example in figure 3. In fact, it would be easy to express a property on the firing of a transition (e.g., if transition t_1 is fired, transition t_2 *will* be fired in the future) using event-based LTL. Such a property would be, in that case, simply expressed as follows: $\mathcal{G}(t_1 \implies \mathcal{F}t_2)$. Expressing the same property using state-based LTL cannot be accomplished in such a straightforward and intuitive way. In general, to express the firing of a transition in state-based LTL we would have to resort to *markings*. Thus, "a transition t is fired" is expressed by the fact that we reach a marking m' that is the immediate successor of a marking m and where the difference between the two markings corresponds to the effects of firing t. More rigorously, t is fired if $\exists m'$ s.t $m \to m'$ with $m \wedge \mathcal{X}m - pre(t) + post(t)$.

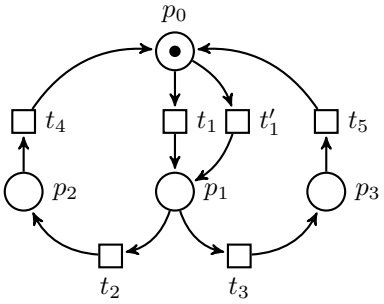


Fig. 3: LTL property example

On the other hand, it can be intuitive to express properties on the fireability of a transition (e.g., transition t_2 *may* be fired in the future) using state-based LTL as follows: $\mathcal{G}(\mathcal{F}p_1)$. Trying to express the same property in event-based LTL can be very hard to do in complex models as it is counter-intuitive. To be able to express both kinds of properties, we would ideally have a model checker that supports both event-based and state-based LTL formulae. In reality, model checkers only support one kind of LTL properties. To get around this technical limitation, we propose a second CPN model for DCR graphs that allows to easily and efficiently express event-based properties as state-based ones.

4.2 CPN4DCR - Generalization for event-based properties

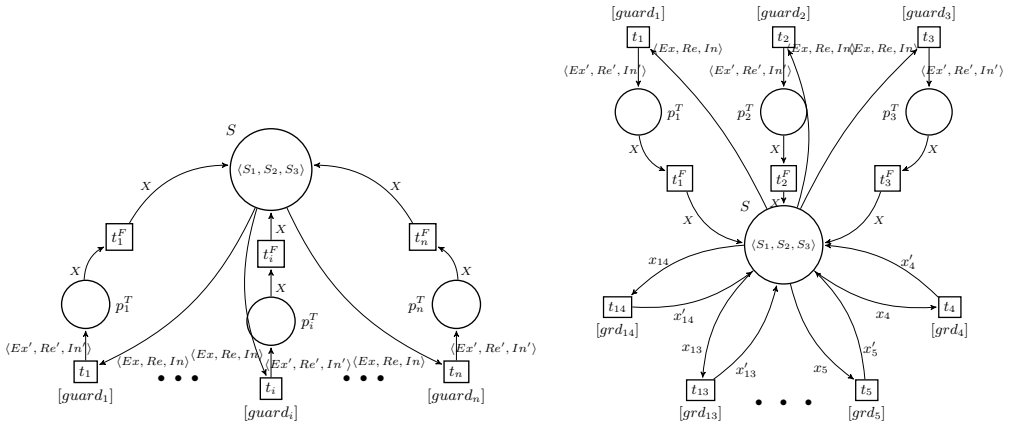
If we go back to the example in figure 3, we note that, if we remove the transition t'_1, we can find a much simpler state-based LTL property to express the event-based one that we had given as example ($\mathcal{G}(t_1 \implies \mathcal{F}t_2)$). In fact, we would not have to resort to expressing a property on the markings of the model, and the following expression: $\mathcal{G}(p_1 \implies \mathcal{F}p_2)$ would be enough to express an equivalent state-based property to the former one. This is mainly due to the fact that by removing the transition t'_1, the place p_1 can only be marked by the firing of t_1, and therefore having a token in p_1 forcibly implies that t_1 has been fired. The idea behind our second model is inspired by this particular case and consists mainly in introducing a set of additional places P^T to the first model, such that we would have one and only one place $p^T \in P^T$ marked when a transition (corresponding to an event in the DCR graph) is fired. We also add a set of *fake* transitions T^F whose role is solely to connect the added places to the main place S of our first model. The definition of our second model can then be given as follows:

Definition 7 (CPN4DCR2). *Given a DCR graph* $Gr = (E, M, Act, \rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%, \rightarrow\diamond, l)$ *and its corresponding initial CPN4DCR model* $CPN4DCR = (P, T, A, \Sigma, V, C, G, E, I)$ *as defined in definition 4, an event-oriented generalization CPN4DCR model* $G\_CPN4DCR = (P', T', A', \Sigma', V', C', G', E', I')$ *(depicted in figure 4a) is defined s.t.:*

- $P' = P \cup P^T$ *where* $P^T = \{p_i^T, \forall i \in [1, n]\}$, *with* $n = |E|$ *the number of events in* Gr
- $T' = T \cup T^F$ *where* $T^F = \{t_i^F, \forall i \in [1, n]\}$, *with* $n = |E|$ *the number of events in* Gr
- $A' = \{(S, t_i), \forall t_i \in T\} \cup \{(t_i, p_i^T), \forall t_i \in T$ *and* $p_i^T \in P^T\} \cup \{(p_i^T, t_i^F), \forall p_i^T \in P^T$ *and* $t_i^F \in T^F\} \cup \{(t_i^F, S), \forall t_i^F \in T^F\}$
- $\Sigma' = \Sigma$
- $V' = V \cup \{X\}$, *with* $Type[X] = (C_E \times C_E \times C_E)$
- $C' = C \cup \{p_{T_i} \rightarrow (C_E \times C_E \times C_E), \forall p_i^T \in P^T\}$
- $G' = G$
- $E' = \{a \rightarrow \langle Ex, Re, In\rangle, \forall a \in A \cap (\{S\} \times T)\} \cup \{(t_i, p_i^T) \rightarrow \langle Ex', Re', In'\rangle, \forall (t_i, p_i^T) \in A \cap (T \times P^T)\} \cup \{a \rightarrow X, \forall a \in A \cap (P^T \times T^F) \cup (T^F, \{S\})\}$ *with (1)* $Ex' = Ex \cup \{i\}$, *(2)* $Re' = (Re \backslash \{i\}) \cup \{j, \forall e_j \in e_i\bullet\rightarrow\}$ *and (3)* $In' = (In \cup \{j, \forall e_j \in e_i \rightarrow+\}) \backslash \{j, \forall e_j \in e_i \rightarrow\%\}$

– $I' = I$

Optimization of the model Adding *fake* transitions and places to our CPN model would indeed increase the size of its state space since we are basically creating intermediate states between the original states from the initial model. This would degrade the performance of the model checker that is supposed to take in charge the verification of the model. In order to minimize the number of extra states to be created, we only create *fake* transitions for the events involved in the property to be verified. For instance, let us consider the following property on the *IHC* example in figure 1: $prop_{DCR} : \mathcal{G}((ReportConditionsModifiedSP \vee ReportConditionsModifiedNu) \implies \mathcal{F}\ UpdateConditionsGP)$ which basically expresses the fact that if any change in the conditions of the patient is reported by either the *SP* or the *Nu*, the *GP* has to update the conditions of the patient in their report. The application of this definition on the DCR graph that would



(a) Generalized CPN4DCR definition (b) Generalized CPN4DCR of IHC example

Fig. 4: Generalized CPN4DCR

be obtained if we were to omit the concept of *roles* from this choreography, with the aim of verifying the property $prop_{DCR}$ is shown in figure 4b.

4.3 CPN4DCR - Extension for DCR choreographies

In order to be able to verify properties on DCR choreographies, we need to take into account the concept of *roles* in our CPN4DCR model. To do so, we mainly add a place R, initially containing the set of $\langle initiator, receivers \rangle$ in the DCR choreography model, linked to all transitions representing events.

Definition 8 (CPN4DCR3). *Given a DCR choreography* (Gr, R) *where* $Gr = (E, M, Act, \rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%, \rightarrow\diamond, l)$*, the corresponding initial CPN4DCR model*

for Gr: $CPN4DCR = (P, T, A, \Sigma, V, C, G, E, I)$ as in Definition 4, and its generalization $G\_CPN4DCR = (P', T', A', \Sigma', V', C', G', E', I')$, the extended CPN4DCR model for choreographies $E\_CPN4DCR = (P'', T'', A'', \Sigma'', V'', C'', G'', E'', I'')$ (depicted in figure 5) is defined s.t.:

- $P'' = P' \cup \{R\}$
- $T'' = T'$
- $A'' = A' \cup \{(R, t_i), \forall t_i \in T\} \cup \{(t_i, R), \forall t_i \in T\}$
- $\Sigma'' = \Sigma \cup \{C_R, (C_R \times \mathcal{P}(C_R))\}$, *where* C_R *is a colour defined as a string type* ($C_R = STRING$) *to represent* roles *in DCR*
- $V'' = V' \cup \{V_R, V_{SR}\}$, *with* $Type[V_R] = C_R$ *and* $Type[V_{SR}] = \mathcal{P}(C_R)$
- $C'' = C' \cup \{R \rightarrow (C_R \times \mathcal{P}(C_R))\}$
- $G'' = G'$
- $E'' = E' \cup \{a \rightarrow \langle V_R, V_{SR}\rangle, \forall a \in A'' \cap (\{R\} \times T) \cup (T \times \{R\})\}$
- $I'' = I' \cup \{R \rightarrow \{r_i, \forall i \in [1..k]\}$, *where each* r_i *being a token representing the initiator ans potential receivers of an event* $e \in E$ *with* $k = |\bigcup_{j=1}^{|E|} l(e_j)|$.

For all $t_i \in T$ representing an event e_i in the DCR graph, we further precise that the expression $\langle V_R, V_{SR}\rangle$ on the arcs connecting t_i to R is defined such that $\langle V_R, V_{SR}\rangle = l(e_i)$. For lack of space, the figure of the CPN model obtained by the application of this definition on the DCR choreography in figure 1 is included in our online repository[4].

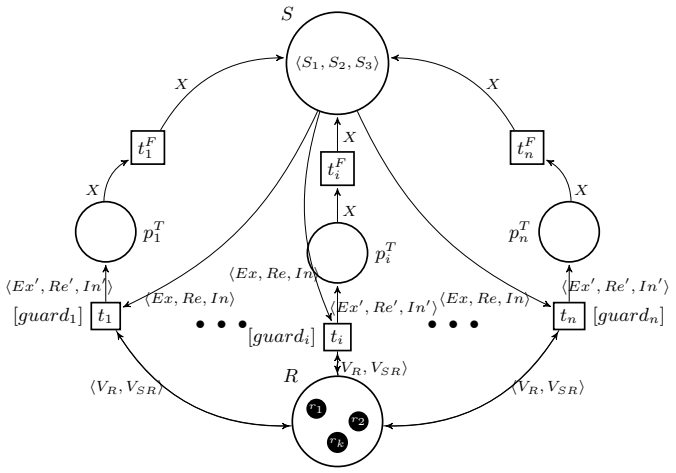


Fig. 5: Extended CPN4DCR for choreographies

[4] See footnote 3.

5 Proof of Concept

To demonstrate the feasibility of our approach we implemented a proof of concept in C++ that applies the formal definitions proposed in this paper. Our *DCR2CPN* prototype, available online in our repository[4], can be used to generate Coloured Petri Nets written in the specification language of *Helena* [9] from DCR graphs introduced as *XML* files[5]. The implementation consists of a parser for DCR graphs, a translator for converting DCR graphs to CPN models, and a CPN specification files generator. The output of our tool can then be fed, along with an LTL property to be verified, into *Helena* which will either confirm that the property holds or provide a counter-example if it does not. Our tool has been tested on the running example as well as a few other DCR models (see the repository for the artifacts). Leveraging *Helena* we were able to verify a variety of properties such as: *absence of dead activities:* there are no activities that can never be executed, *absence of incomplete process executions:* a process instance can always complete its execution, *reachability of an activity:* all executions of a process must pass by a certain activity (e.g., in our *IHC* example, we can check that all executions must pass by the "Discharge patient" activity), *acceptance of the flow:* the process execution always terminates in an accepting state (i.e., all pending responses are eventually executed or excluded), *other case-specific properties:* for example, in our use case, we can check that if any change in the conditions of the patient is reported by either the *SP* or the *Nu*, the *GP* has to update the conditions of the patient in their report. An exhaustive evaluation using a benchmark of DCR graphs is indeed to be carried out, but these preliminary results bode well as to the validity and efficiency of our approach.

6 Conclusion

Our work proposes a formal verification approach for DCR graphs based on CPN as a pivot formalism. We propose CPN4DCR, a CPN model that we have proven to be semantically equivalent to a DCR graph. This initial model shows some limitations when it comes to the verification of event-based LTL properties, which we solve by proposing a generalization of this model that allows us to verify both state- and event-based properties. Then we propose an extension that allows to additionally represent DCR choreographies in CPN. The endgame is to develop a verification tool that would leverage our CPN4DCR model as well as the *Helena* model checker to verify properties expressed on DCR models. This tool would hide the intricacies of CPN and model checking from the user in order to be accessible to non formal modelling experts. At the time being, we have automated the transformation of a DCR graph into the initial CPN4DCR model. Our next step is to complete the automation of the generalized and extended CPN4DCR models. We will also work on a transformation module for the property to be verified. This module would automatically generate an LTL property expressed on the CPN4DCR elements, given a property that the user

[5] https://documentation.dcr.design/documentation/dcr-xml/

would express on the input DCR model. The *Helena* model checker needs to be integrated in the tool to be invoked on the CPN4DCR model along with the generated LTL property. The final step is to present the result returned by *Helena* (i.e., the counter example given in case of violation of the verified property, which is expressed on the elements of CPN4DCR) into a result comprehensible by the user (i.e., expressed on the elements of the DCR).

References

1. van der Aalst, W.M.P.: The application of petri nets to workflow management. J. Circuits Syst. Comput. **8**(1), 21–66 (1998)
2. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: yet another workflow language. Inf. Syst. **30**(4), 245–275 (2005)
3. van der Aalst, W.M.P., Pesic, M.: Decserflow: Towards a truly declarative service flow language. In: Web Services and Formal Methods, Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4184, pp. 1–23. Springer (2006)
4. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Comput. Sci. Res. Dev. **23**(2), 99–113 (2009)
5. Boubaker, S., Klai, K., Kortas, H., Gaaloul, W.: A formal model for business process configuration verification supporting or-join semantics. In: On the Move to Meaningful Internet Systems. OTM. Lecture Notes in Computer Science, vol. 11229, pp. 623–642 (2018)
6. Bryans, J.W., Wei, W.: Formal analysis of BPMN models using event-b. In: Formal Methods for Industrial Critical Systems - 15th International Workshop, FMICS. Lecture Notes in Computer Science, vol. 6371, pp. 33–49. Springer (2010)
7. Cosma, V.P., Hildebrandt, T.T., Slaats, T.: Transforming dynamic condition response graphs to safe petri nets. In: Application and Theory of Petri Nets and Concurrency - 44th International Conference, PETRI NETS 2023, Lisbon, Portugal, June 25-30, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13929, pp. 417–439. Springer (2023)
8. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008)
9. Evangelista, S.: High level petri nets analysis with helena. In: Applications and Theory of Petri Nets 2005. pp. 455–464. Berlin, Heidelberg (2005)
10. Gaaloul, W., Bhiri, S., Rouached, M.: Event-based design and runtime verification of composite service transactional behavior. IEEE Trans. Serv. Comput. **3**(1), 32–45 (2010)
11. Goedertier, S., Vanthienen, J.: Declarative process modeling with business vocabulary and business rules. In: On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, Proceedings, Part I. vol. 4805, pp. 603–612. Springer (2007)
12. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software. EPTCS, vol. 69, pp. 59–73 (2010)
13. Hildebrandt, T.T., Slaats, T., López, H.A., Debois, S., Carbone, M.: Declarative choreographies and liveness. In: Formal Techniques for Distributed Objects, Components, and Systems FORTE 2019. Lecture Notes in Computer Science, vol. 11535, pp. 129–147. Springer (2019)

14. Houhou, S., Baarir, S., Poizat, P., Quéinnec, P., Kahloul, L.: A first-order logic verification framework for communication-parametric and time-aware BPMN collaborations. Inf. Syst. **104**, 101765 (2022)
15. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer Publishing Company, Incorporated, 1st edn. (2009)
16. Kheldoun, A., Barkaoui, K., Ioualalen, M.: Formal verification of complex business processes based on high-level petri nets. Inf. Sci. **385**, 39–54 (2017)
17. Morimoto, S.: A survey of formal verification for business process modeling. In: Computational Science - ICCS 2008, 8th International Conference. Lecture Notes in Computer Science, vol. 5102, pp. 514–522. Springer (2008)
18. Mukkamala, R.R.: A Formal Model For Declarative Workflows Dynamic Condition Response Graphs. Ph.D. thesis (06 2012)
19. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (1989)
20. OMG: Business process model and notation (bpmn) 2.0. https://www.omg.org/spec/BPMN/ (2014)
21. Ou-Yang, C., Lin, Y.D.: Bpmn-based business process model feasibility analysis: a petri net approach. International Journal of Production Research **46**, 3763 – 3781 (2008)
22. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws. Lecture Notes in Computer Science, vol. 4103, pp. 169–180. Springer (2006)
23. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference. pp. 287–300. IEEE Computer Society (2007)
24. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011. vol. 99, pp. 383–394 (2011)
25. Pnueli, A.: The temporal logic of programs. In: Annual Symposium on Foundations of Computer Science, Providence. pp. 46–57. IEEE Computer Society (1977)
26. Puhlmann, F., Weske, M.: Using the *pi*-calculus for formalizing workflow patterns. In: Business Process Management, 3rd International Conference, BPM 2005, Nancy, France, September 5-8, 2005, Proceedings. vol. 3649, pp. 153–168 (2005)
27. Russo, V., Ciampi, M., Esposito, M.: A business process model for integrated home care. In: The 6th International Conference on Emerging Ubiquitous Systems and Pervasive Networks. Procedia Computer Science, vol. 63, pp. 300–307 (2015)
28. Schönig, S., Jablonski, S.: Comparing declarative process modelling languages from the organisational perspective. In: Business Process Management Workshops - BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 - September 3, 2015, Revised Papers. Lecture Notes in Business Information Processing, vol. 256, pp. 17–29. Springer (2015)
29. Verbeek, E., van der Aalst, W.M.P.: Woflan 2.0: A petri-net-based workflow diagnosis tool. In: Application and Theory of Petri Nets 2000, ICATPN 2000, Proceeding. Lecture Notes in Computer Science, vol. 1825, pp. 475–484. Springer (2000)
30. Ye, J., Sun, S., Song, W., Wen, L.: Formal semantics of bpmn process models using yawl. In: 2008 Second International Symposium on Intelligent Information Technology Application. vol. 2, pp. 70–74 (2008)