

Colored Petri Nets to Verify Extended Event-Driven Process Chains

Kees van Hee, Olivia Oanea, and Natalia Sidorova

Department of Mathematics and Computer Science,
Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{k.m.v.hee, o.i.oanea, n.sidorova}@tue.nl

Abstract. Business processes are becoming more and more complex and at the same time their correctness is becoming a critical issue: The costs of errors in business information systems are growing due to the growing scale of their application and the growing degree of automation. In this paper we consider *Extended Event-driven Process Chains (eEPCs)*, a language which is widely used for modeling business processes, documenting industrial reference models and designing workflows. We describe how to translate eEPCs into timed colored Petri nets in order to verify processes given by eEPCs with the CPN Tools.

Keywords: Extended EPCs, semantics, verification, colored Petri nets.

1 Introduction

Event-driven Process Chains (EPC) [15,17] is a popular language for modeling business processes, documenting industrial reference models and designing workflows. EPCs describe the flow of control of business processes as a chain of functions, events, and logical connectors. Functions represent activities in a business process. An event expresses a precondition (trigger) for a function or a postcondition that signals the termination of a function. Logical connectors **and**, **or**, and **xor** are used according to their names to build the control flow of a process in a natural way.

EPCs extended with data, resources, time and probabilities, called *extended EPCs (eEPCs)* [17], are intensively used in commercial tools like *Architecture of Integrated Information Systems (ARIS)* [15] and *SAP R/3* [11]. These tools support modeling and simulation of organizational processes with eEPCs, and they are widely used in such branches of industry and consultancy as banks, insurance companies, transportation. The complexity of business processes in these branches is growing throughout the years. Due to informatisation, which concerns all aspects of organizational activities, less and less manual work is involved into the supervision of business processes. This accelerates processes significantly, but also puts higher requirements to the correctness of process specifications, since an error in a process design would demonstrate itself in an automated system too late, when it would already cause a snowball effect.

We choose to use an available tool for modeling, simulation and analysis of the constructed model, e.g. model checking which covers the whole system behavior, and we provide a translation from eEPCs to the input language of this tool. Petri nets are appropriate for modeling EPCs since all EPC elements can be translated to places and transitions of Petri nets in a natural way (see e.g. [1,7,13]). Extended EPCs have such additional features as data, time and probabilities. Therefore timed colored Petri nets (TCPNs) [10] are a natural choice for modeling eEPCs. CPN Tools [3] provides modeling, simulation, and model checking options for TCPNs and thus satisfies our requirements.

In this paper, we provide a formal definition for eEPCs and present their semantics in terms of a transition system. We provide a translation from eEPCs to TCPNs and describe how we can analyze the behavior of an eEPC with the CPN Tools. We conclude by comparing our method to other approaches for formalizing the syntax and the semantics of EPCs.

The rest of the paper is organized as follows. In Section 2 we describe the syntax of eEPCs. In Section 3 we provide the semantics of eEPCs as used in practice. Section 4 gives a translation from eEPCs to timed colored Petri nets and discusses some verification issues. In Section 5 we give an account of related work and discuss some future work.

2 Syntax of Extended Event-Driven Process Chains

In this section, we give a brief description of the syntax of eEPCs taking into account requirements given in [15] as well as the ones imposed by practice. We use ARIS [6,9,15,17] as a reference point of our study. However, this approach can be applied to other tools supporting eEPCs, since they are based on the same concepts.

ARIS offers a conceptual framework for describing companies, their organizational structure, processes and resources (material as well as human). In addition to process modelling, ARIS offers the possibility to analyze process performance based on simulation. In order to structure process modelling and to show different angles of an organization, ARIS distinguishes five main views:

- Data view** uses the entity-relationship models (ERM) to design data models: entities (e.g. data objects of the environment that are processed by the system), their attributes and relationships between entities;
- Function view** describes functions as tasks performed on objects to support different company goals; it includes descriptions of procedures, processes, subfunctions and elementary functions;
- Organization view** models the relations between company units and the classification of these units in the organizational hierarchy;
- Product/service view** describes the products and services produced by the company as a result of human act or technical procedures;
- Control view** integrates the previously mentioned views and defines the dynamic, behavioral aspects. The control flow of a process is described with

an EPC extended with the description of the resources and data involved in the process, and timed and probabilistic aspects of the behavior.

The control view is essential for process verification, so we concentrate our study on this view. In what follows, we define generic EPCs and extend them to eEPCs as they are presented in the control view of ARIS.

2.1 Syntax of EPCs

First we give some basic definitions from algebra and the graph theory we need here.

- Let S be a set. $|S|$ denotes the number of elements in S . \mathbb{B} denotes the boolean set, \mathbb{N} the set of natural numbers, \mathbb{Z} the set of integers, \mathbb{R} the set of real numbers and \mathbb{R}^+ the set of positive real numbers.
- A multiset (bag) over S is a mapping $m: S \rightarrow \mathbb{N}$. The set of all multisets over S is \mathbb{N}^S . We use $+$ and $-$ for the sum and the difference of two multisets and $=$, $<$, $>$, \leq , \geq for comparisons of multisets. \emptyset denotes the empty multiset and \in denotes element inclusion. We write $m = 2^a$ for a multiset m with $m(a) = 2$ and $m(x) = 0$ for any $x \in S - \{a\}$.
- Let $R \subseteq S \times S$ be a binary relation over a set S . R^{-1} denotes the converse relation of R , R^+ denotes the transitive closure of R , R^* denotes the transitive reflexive closure of R and $(R \cup R^{-1})^*$ is the symmetric, reflexive and transitive closure of R .
- A *directed graph* is a tuple $G = (N, A)$, where N is a set of nodes and $A \subseteq N \times N$ is a set of arcs. Every arc $a \in A$ is a pair $(n_1, n_2) \in N \times N$ consisting of the input node n_1 and the output node n_2 .
- A *path* σ of length m in a graph $G = (N, A)$ is a finite sequence of nodes $\sigma = n_0 n_1 \dots n_m$ ($n_i \in N$ for $i = 0, \dots, m$) such that each pair $(v_j, v_{j+1}) \in A$ ($j = 0, \dots, m - 1$) is an arc. We denote the length of a path by $|\sigma| (= m)$. σ is an empty path if $|\sigma| = 0$. We denote the set of all finite, possibly empty, paths by N^* and the set of finite non-empty paths by N^+ . A path σ is a prefix of a path γ if there is a path σ' with $\gamma = \sigma\sigma'$.
- A graph $G = (N, A)$ is *weakly connected* if for every two nodes $n_1, n_2 \in N$, $(n_1, n_2) \in (A \cup A^{-1})^*$.
- We denote the set of *output nodes* of $n \in N$ as n^\bullet , i.e. $n^\bullet = \{n' | (n, n') \in A\}$. Similarly, ${}^\bullet n = \{n' | (n', n) \in A\}$ is the set of *input nodes* of $n \in N$. Given a set of nodes $X \subseteq N$, we define ${}^\bullet X = \bigcup_{n \in X} {}^\bullet n$ and $X^\bullet = \bigcup_{n \in X} n^\bullet$.
- We denote the set of *ingoing arcs* of a node $n \in N$ as A_n^{in} , i.e. $A_n^{in} = \{(x, n) | x \in {}^\bullet n\}$ and the set of *outgoing arcs* of n as A_n^{out} , i.e. $A_n^{out} = \{(n, x) | x \in n^\bullet\}$. In case A_n^{in} is singleton, a_n^{in} denotes the ingoing arc of the node n . Similarly, if A_n^{out} is singleton, a_n^{out} denotes the outgoing arc of the node n .

Now we can give a definition of a generic EPC.

Definition 1 (EPCs). *An event-driven process chain (EPC) is defined by a weakly connected directed graph $G = (N, A)$ that satisfies the following properties:*

1. The set N of nodes is the union of three pairwise disjoint sets E , F and C , where
 - E is the set of events. $E = E_s \cup E_f \cup E_i$, where E_s , E_f and E_i are pairwise disjoint sets of start events, final events, and internal events respectively, with $|E_s| \geq 1$ and $|E_f| \geq 1$;
 - $F \neq \emptyset$ is a set of functions;
 - C is a set of connectors of types **xor**, **or**, **and**, i.e. $C = C_{\text{xor}} \cup C_{\text{or}} \cup C_{\text{and}}$, where C_{xor} , C_{or} and C_{and} are disjoint sets. Furthermore, each of these sets is partitioned into two sets representing split and join connectors: $C_{\text{xor}} = C_{\text{xs}} \cup C_{\text{xj}}$, $C_{\text{or}} = C_{\text{os}} \cup C_{\text{oj}}$ and $C_{\text{and}} = C_{\text{as}} \cup C_{\text{aj}}$, and C_{s} stands for set of split connectors, and C_{j} stands for the set of join connectors;
2. Every element from the set A of arcs connects two different nodes. Moreover,
 - $\bullet e_s = \emptyset$ and $e_f \bullet = \emptyset$, for each $e_s \in E_s$ and $e_f \in E_f$;
 - $|n \bullet| = 1$ for each $n \in F \cup E_i \cup E_s$, and $|\bullet n| = 1$ for each $n \in F \cup E_i \cup E_f$;
 - each split connector $c \in C_{\text{s}}$ satisfies $|\bullet c| = 1$ and $|c \bullet| > 1$; similarly each join connector $c \in C_{\text{j}}$ satisfies $|\bullet c| > 1$ and $|c \bullet| = 1$;
3. Each node is on a path from a start event to a final event, i.e. for any $n \in N$, there is a path σ from some $e_s \in E_s$ to some $e_f \in E_f$, such that $n \in \sigma$;
4. Functions and events alternate along the control flow, i.e. each path starting in an event $e \in E - E_f$ and ending in an event $e_f \in E_f$ has a prefix of the form $e\sigma_c f$, where $\sigma_c \in C^*$ and $f \in F$. Similarly, for each path σ starting in a function $f \in F$ and ending in an event $e_f \in E_f$ there is a prefix $f\sigma_c e$, where $\sigma_c \in C^*$ and $e \in E - E_s$;
5. Events do not precede the **xor** and the **or** split, i.e. $\forall c \in C_{\text{xs}} \cup C_{\text{os}}: \bullet c \cap E = \emptyset$;
6. There is no cycle that consists of connectors only, i.e. for any path $\sigma = v_0 v_1 \dots v_n \in C^+$: $n \geq 2$: $v_0 \neq v_n$.

2.2 Syntax of Extended EPCs

The control view of an eEPC has an EPC as a skeleton. Data attributes, resources, time and probabilities are linked to different EPC elements to form an extended EPC.

Functions represent activities that may take time, may require access to diverse resources and may perform operations on some data or resources.

Functions that perform operations on data attributes are annotated with expressions denoting the operation performed (see for example Figure 1(a)). Personnel, material or information resources can be used to execute functions. We call these objects capacity resources, since they are characterized by minimum and maximum capacities to run the process. Functions are annotated with a nonnegative integer or real constant denoting the number of resources required, produced or consumed. In Figure 1(b), function *finish products* produces 1000 items of resource *Item 1* with the capacity domain $[100, 5000]$ and consumes 500 items of resource *Item 2*.

Furthermore, functions can be either *timed*, i.e. they may have a duration, or *immediate*, i.e. they take zero time. The duration of each timed function is described by a probability distribution.

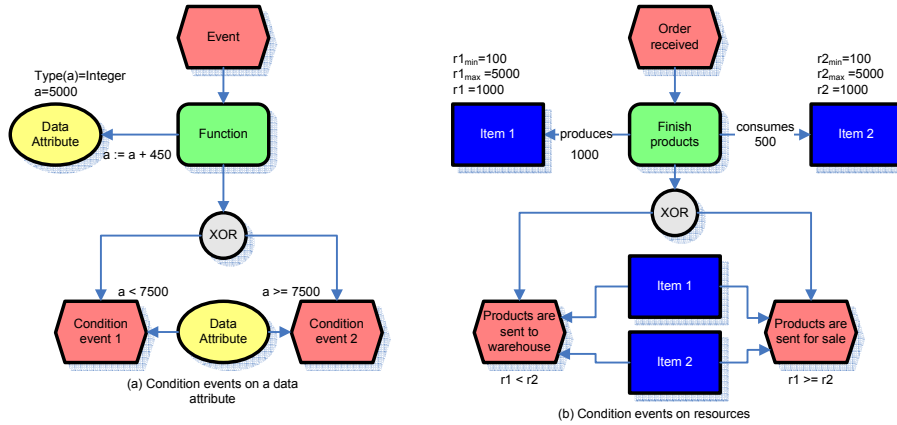


Fig. 1. Condition events on data attributes and resources

Events define either conditions on data attributes or resource capacities, or triggers from elements outside the process.

Processes are instantiated at **start events**. Start events may be grouped in **start event sets**¹ that contain events which are synchronized, i.e. a process is started at the same time at all the events of the respective set. A probability distribution is assigned to each start event set, denoting the frequency with which process instances are created for the events in the respective set.

Conditions (boolean expressions) on data attributes or on resources determine the terms of the respective event. An event that follows an **or** split or an **xor** split connector and is determined by a condition is called a **condition event**. Condition events may have **attributes** or **capacity resources** connected to them and conditions are specified as:

- conditions on one operand that have a constant value of the same type as the attribute or the capacity value of a resource as comparison criterion. Figure 1(a) shows two condition events annotated with boolean expressions on a data attribute;
- conditions on two operands that compare two attribute values or the capacity values of two resources. In Figure 1(b), the condition events *products are sent to warehouse* and *products are sent for sale* are annotated with the boolean expressions $r1 < r2$ and $r1 \geq r2$ on the resources *Item 1* and *Item 2*.

The rest of events are used to model triggers from outside the process and they have a probability value assigned. This value is used during the simulation to determine whether the execution stops or continues at the respective event. Probability values for events following **and** split connectors are 1 since the execution cannot stop at events following such a connector. The sum of probability

¹ In ARIS, event diagrams are used to represent start event sets.

values for events following **xor** split connectors is exactly 1 as the execution can continue only on one outgoing branch. Furthermore, the sum of probability values for events following **or** split connectors is greater than 1 as the execution can continue on one or more outgoing branches.

Or join connectors may also contain some timeout information, called *synchronization timeout*.

We give a formal definition of eEPCs as they are used in ARIS Toolset.

Definition 2 (eEPC). *An extended event-driven process chain (eEPC) is a tuple $G_e = (G, \mathcal{A}, \mathcal{R}, \mathbf{Type}, \mathbf{Expr}, PDF, \mathbf{Pr})$, where*

- G is an underlying EPC.
- \mathcal{A} is a set of data attributes. We write $\mathcal{A} = \bigcup_{f \in F} \mathcal{A}_f$ for a partition of the set of data attributes w.r.t. the function performing operations on them.
- \mathcal{R} is a set of capacity resources. We partition the set of resources according to the function performing operations on them: $\mathcal{R} = \bigcup_{f \in F} \mathcal{R}_f$ such that these sets are not disjoint (several functions can perform operations on the same resource). Moreover, we consider a partition of the set of resources used by a function f into used, produced and consumed resources, i.e. $\mathcal{R}_f = \mathcal{R}_f^u \cup \mathcal{R}_f^p \cup \mathcal{R}_f^c$ such that these sets are disjoint (a function can perform just one type of operation on a resource).
- **Type** maps each attribute to one of the types **Text**, **Enum**, \mathbb{B} , \mathbb{Z} , or \mathbb{R} and each capacity resource r to a real or integer subtype $[r_{\min}, r_{\max}]$, where r_{\min} and r_{\max} are the minimum and maximum capacities of resource r .
- **Expr** = $\mathbf{Expr}^b \cup \bigcup_{x \in \mathcal{R} \cup \mathcal{A}} \mathbf{Expr}_x$ maps condition events and functions into expressions on the attributes or capacity resources linked to them as follows:
 - E_c denotes the set of **condition events**, i.e. events following **or** and **xor** split connectors that have conditions on data attributes or resources. Every condition event $e \in E_c$ is mapped to a boolean expression $\mathbf{Expr}^b(e)$ of the form $v_1 \mathbf{x} v_2$ or $v_1 \mathbf{x} c$, where v_1, v_2 are either attributes or resource capacities, c is a constant so that $\mathbf{Type}(v_1) = \mathbf{Type}(v_2) = \mathbf{Type}(c)$ and \mathbf{x} is the comparison operator compatible with the types used.
 - every function f performing an operation on an attribute a is mapped to an expression on a , namely $\mathbf{Expr}_a(f)$, having the form $a := c$, where c is a constant value with $\mathbf{Type}(a) = \mathbf{Type}(c)$, or $a := a \mathbf{x} c$, with $\mathbf{Type}(a) = \mathbb{Z}$ or $\mathbf{Type}(a) = \mathbb{R}$, constant c with $\mathbf{Type}(a) = \mathbf{Type}(c)$ and $\mathbf{x} \in \{+, -, *\}$.
 - $\mathbf{Expr}_r(f)$ maps function f using, producing or consuming a resource $r \in \mathcal{R}_f$ to constant $c_r^f \in \mathbf{Type}(r)$, denoting the quantity of resources used, consumed or produced.
- F_t denotes the timed functions, and C_s denotes the set of **or** join connectors with synchronization timeout. We consider the set of start events E_s partitioned into start event sets, i.e. $\bigcup_{d \in I_s} E_s^d$, for some index set I_s .

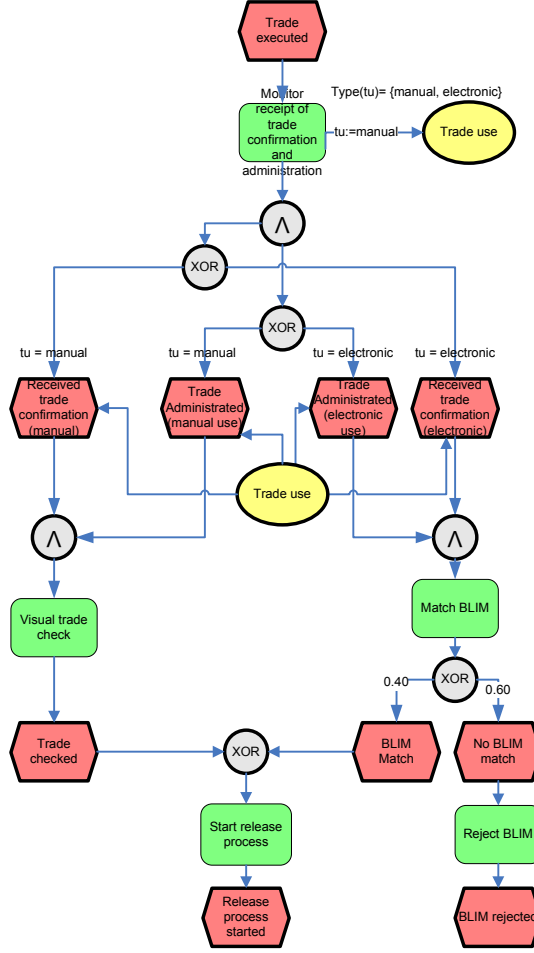


Fig. 2. Trade matching eEPC

$PDF = \bigcup_{k \in (F_t \cup C_s \cup I_s)} pdf_k$ denotes a family of continuous or discrete probability distributions² for the duration of timed functions, for the synchronization timeouts of synchronized **or** join connectors, and for the delays of start event sets.

- $\mathbf{Pr}: E - E_s - E_c \rightarrow [0, 1]$ which maps events to their probability values such that:
 - $\sum_{e \in E^c} \mathbf{Pr}(e) = 1$, for each set of events E^c following an **xor** split connector $c \in C_{\mathbf{XS}}$ that have probability values;

² In this paper, a probability distribution $pdf \in PDF$ refers to a probability distribution function (we do not mention the word function in order to avoid confusion with functions as nodes of eEPCs).

- $\sum_{e \in E^c} \Pr(e) \geq 1$, for each set of events E^c that have probability values and follow an **or** split connector $c \in C_{\text{os}}$;
- $\Pr(e) = 1$ for each $e \in C_{\text{as}}$.

Figure 2 shows an eEPC modeling a part of a trade matching process taking place in a company. The process checks the timely receipt of a confirmation, administrates the trade internally and matches the confirmation against the internal data before the release process can be started.

The process starts with the event *trade executed (deal made)* that triggers the function *monitor receipt of trade confirmation and administration*. This results in a change of the data attribute *trade use*. The execution is then split into two parallel threads, which is modeled by an **and** split. The left thread models the check whether the confirmation of the trade has been received electronically or manually by means of an **xor** split and two condition events: *Received trade confirmation (manual)* and *Received trade confirmation (electronic)* that are linked to a data attribute *Trade use*. The second thread models the check whether the trade is administered for manual or electronic use by means of conditions on the attribute *trade use*.

The two **and** join connectors make sure that the matching process continues either manually or electronically. The visual (manual) check is performed by the function *visual trade check* and results in the event *trade checked*. The result of the electronic matching of the internal information with BLIM messages has 40% probability to succeed. In case the trade has been matched either manually or electronically, which is modeled by an **xor** join, the process is released by *start release process*. If the data registered internally does not match the information contained in the BLIM message, the message is rejected.

3 Semantics of eEPCs

We introduce first the notions of a *process folder* and a *state* of an eEPC necessary to define the semantics of eEPCs.

A **process folder** is an object that resides at a node (function or start event) or at an arc. Furthermore, it carries a folder number and a timestamp denoting the value of the timer associated to the folder. Timestamps are either nonnegative numbers indicating the delay after which the folder may be used, or \perp , denoting that the timer of the process folder is off and the folder can be used directly. A **state** of an eEPC is defined by a multiset of process folders together with a valuation of data attributes and capacity resources.

For the rest of the paper, we consider the discrete time domain $\mathbb{N}_{\perp} = \mathbb{N} \cup \{\perp\}$ and discrete probability distributions for durations. We denote the domain of a discrete probability distribution $pdf \in PDF$ by $Dom(pdf) \subseteq \mathbb{N}$. The same approach can be applied for continuous time and continuous probability distributions. We consider process folder numbers to take values from \mathbb{N} . Formally:

Definition 3. Let $G_e = (G, \mathcal{A}, \mathcal{R}, \mathbf{Type}, \mathbf{Expr}, PDF, \mathbf{Pr})$ be an eEPC. A process folder is a tuple $p = (n, (i, t))$ where $n \in E_s \cup F \cup A$ is a start event,

a function or an arc, $i \in \mathbb{N}$ is a process folder number and $t \in \mathbb{N}_\perp$ is a timestamp. A state of G_e is a tuple $s = (m, \mathbf{Val})$, where m is a multiset of process folders, i.e. $m: ((F \cup A \cup E_s) \times (\mathbb{N} \times \mathbb{N})) \rightarrow \mathbb{N}$, and \mathbf{Val} is a valuation function that maps every resource $r \in \mathcal{R}$ into some value $\mathbf{Val}(r) \in \mathbf{Type}(r)$ and every data attribute $a \in \mathcal{A}$ to some value $\mathbf{Val}(a) \in \mathbf{Type}(a)$.

We denote the timestamp of a process folder p by p_t . We will say that a process folder has its timer *off* when $p_t = \perp$ and its timer *on* when $p_t \geq 0$. We call a process folder with the timer on *active* if it has the timestamp 0. If $p_t > 0$, the process folder is *waiting* to become active.

Every start event of a start event set has initially an active process folder with the index of the start event set as its folder number. The initial state is thus $s_0 = (m_0, \mathbf{Val}_0)$ and contains one active process folder on every start event so that for every start event set $(e_s, (i, 0)) \in m_0$ for all $e_s \in E_s^i$, and every resource and data attribute has an initial value according to the specification.

Probability distributions are used in eEPCs to model the behavior of the environment or to describe nondeterminism in the system (when decisions need to be made), and for performance analysis. Since all events that have $\mathbf{Pr}(e) > 0$ can occur and the errors in a model (eEPC) having probabilistic events can thus be detected on the model without probabilities, we do not take probabilities further into consideration, as they are irrelevant to our verification purposes. In order to express nondeterminism without probabilities, we extend the mapping \mathbf{Expr} to non-condition events and set $\mathbf{Expr}^b(e) = \text{true}$, for every event $e \in (C_{\text{os}} \cup C_{\text{xs}})^\bullet - E_c$, and subsequently extend the set of *condition events* to all events following an **or** and **xor** split connector, i.e. $E_c = (C_{\text{os}} \cup C_{\text{xs}})^\bullet$.

Let *eval* be the *evaluation function* for expressions, such that:

- $\text{eval}(\mathbf{Expr}^b(e), \mathbf{Val}) \in \mathbb{B}$ is the evaluation function of the boolean expression of condition events $e \in E_c$ in the valuation \mathbf{Val} .
- $\text{eval}(\mathbf{Expr}_a(f), \mathbf{Val}) \in \mathbf{Type}(a)$ is the evaluation function of the expression $\mathbf{Expr}_a(f)$ on some data attribute $a \in \mathcal{A}_f$ in the valuation \mathbf{Val} that computes a new value for a from the right hand side expression of $\mathbf{Expr}_a(f)$.

For any $r \in \mathcal{R}$, we introduce a variable \bar{r} such that $\mathbf{Val}_0(r) = \mathbf{Val}_0(\bar{r})$ in order to keep track of resources that would be modified by several functions. We denote the set of variables newly introduced by $\bar{\mathcal{R}}$.

We describe the semantics of an eEPC by means of a transition relation between states as follows:

Definition 4. Let $G_e = (G, \mathcal{A}, \mathcal{R}, \mathbf{Type}, \mathbf{Expr}, PDF, \mathbf{Pr})$ be an eEPC. The semantics of an eEPC is given by a transition system $TS = (\Sigma, s_0, \rightarrow)$, where Σ is the set of states of G_e , s_0 is the initial state, and $\rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation described by the rules (a) – (j) below. Let $s = (m, \mathbf{Val})$ and $s' = (m', \mathbf{Val}')$ be two states in Σ .

- (a) **start event set rule** Let E_s^d be a start event set such that there is a process folder on each of its start events and all the folders have the same folder number and are active. Then a step can be taken that results in removing all the folders on the events of E_s^d and producing a process folder on each of the outgoing arcs of E_s^d which have the same folder number as the original process folders and their timers are set to off. Furthermore, a new process folder is generated on each event of E_s^d ; all these new process folders have the same newly generated folder number and the same timestamp which is drawn from the probability distribution of the start event set. Formally, if $(e, (i, 0)) \in m$ for all $e \in E_s^d$, $d \in I_s$ and $i \in \mathbb{N}$, then $s \rightarrow s'$, with $m' = m - \sum_{e \in E_s^d} (e, (i, 0)) + \sum_{e \in E_s^d} (a_e^{out}, (i, \perp)) + \sum_{e \in E_s^d} (e, (i + |I_s|, t))$, for some $t \in \text{Dom}(\text{pdf}_d)$.
- (b) **event rule** Let a_e^{in} be the incoming arc for an event e such that there is a process folder on a_e^{in} . Then, the process folder can be removed from a_e^{in} and placed on the outgoing arc of the event a_e^{out} . Formally, if $(a_e^{in}, (i, \perp)) \in m$ for some $i \in \mathbb{N}$ and $e \in E - E_s - E_c - E_f$, then $(m, \mathbf{Val}) \rightarrow (m', \mathbf{Val})$ with $m' = m + (a_e^{out}, (i, \perp)) - (a_e^{in}, (i, \perp))$.
- (c) **function rule** Let a_f^{in} be the incoming arc of a function f such that there is a process folder on a_f^{in} . The function rule consists of two steps:
- if the resources may be used, consumed or produced, the process folder is removed from the incoming arc of the function, and a new process folder having the same folder number as the original folder and a timestamp from the time distribution interval of the function is placed on the function, and resources are consumed. Formally, if $(a_f^{in}, (i, \perp)) \in m$, for some $f \in F$ and $i \in \mathbb{N}$, $\mathbf{Val}(r) - \mathbf{Expr}_r(f) \leq r_{\min}$ for all $r \in \mathcal{R}_f^u \cup \mathcal{R}_f^c$, $\mathbf{Val}(\bar{r}) + \mathbf{Expr}_r(f) \leq r_{\max}$ for all $r \in \mathcal{R}_f^p$, then $s \rightarrow s'$, where $s' = (m', \mathbf{Val}')$, $m' = m - (a_f^{in}, (i, \perp)) + (f, (i, t))$, where $t = 0$ if $t \in F - F_t$ and $t \in \text{Dom}(\text{pdf}_f)$ if $t \in F_t$, $\mathbf{Val}'(r) = \mathbf{Val}(r) - \mathbf{Expr}_r(f)$ for all $r \in \mathcal{R}_f^c \cup \mathcal{R}_f^u$, $\mathbf{Val}'(\bar{r}) = \mathbf{Val}(\bar{r}) - \mathbf{Expr}_r(f)$ for all $r \in \mathcal{R}_f^c$, $\mathbf{Val}'(\bar{r}) = \mathbf{Val}(\bar{r}) + \mathbf{Expr}_r(f)$ for all $r \in \mathcal{R}_f^p$ and $\mathbf{Val}'(x) = \mathbf{Val}(x)$ for all $x \in (\mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}) - (\mathcal{R}_f^c \cup \mathcal{R}_f^u \cup \bar{\mathcal{R}}_f^c \cup \bar{\mathcal{R}}_f^p)$.
 - Let f be a function such that there is an active process folder on it. Then, the active process folder is removed from the function, and a new process folder is produced on the outgoing arc of the function; this new folder has the same folder number as the removed one and the timer off; attributes are evaluated and resources are produced or released. Formally, if $(f, (i, 0)) \in m$, for some $f \in F$ and $i \in \mathbb{N}$, then $s \rightarrow s'$, with $s' = (m', \mathbf{Val}')$, $m' = m - (f, (i, 0)) + (a_f^{out}, (i, \perp))$, where $\mathbf{Val}'(a) = \text{eval}(\mathbf{Expr}_a(f), \mathbf{Val})$ for all $a \in \mathcal{A}_f$, $\mathbf{Val}'(r) = \mathbf{Val}(r) + \mathbf{Expr}_r(f)$ for all resources $r \in \mathcal{R}_f^p \cup \mathcal{R}_f^u$ produced or used, and $\mathbf{Val}'(x) = \mathbf{Val}(x)$ for all $x \in (\mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}) - (\mathcal{A}_f \cup \mathcal{R}_f^p \cup \mathcal{R}_f^u)$.
- (d) **and split rule** Let a_c^{in} be the ingoing arc of an **and split** connector c such that there is a process folder on a_c^{in} . The rule results in removing the process folder from a_c^{in} and placing a process folder on each outgoing arc of the **and split** connector such that all new process folders have the same folder number

as the removed folder. Formally, if $(a_c^{in}, (i, \perp)) \in m$, for some $i \in \mathbb{N}$ and $c \in C_{as}$, then $s \rightarrow s'$ with $s' = (m, \mathbf{Val})$ and $m' = m + \sum_{a' \in A_c^{out}} (a', (i, \perp)) - (a_c^{in}, (i, \perp))$.

- (e) **xor split rule** Let a_c^{in} be the incoming arc of an **xor** split connector c such that there is a process folder on a_c^{in} . Then the process folder is removed from the arc a_c^{in} and
- if the **xor** split leads to a non-final condition event whose boolean expression is evaluated to true, a process folder with the same number as the removed one is placed on the outgoing arc of the event;
 - if there is an outgoing arc of the **xor** split leading to a final condition event whose boolean expression is evaluated to true or to another connector, then a process folder with the same number as the removed one is placed on the respective arc.

Formally, if $(a_c^{in}, (i, \perp)) \in m$, for some $i \in \mathbb{N}$ and $c \in C_{xs}$, then $s \rightarrow s'$, with $s' = (m', \mathbf{Val})$ and $m' = m + (a', (i, \perp)) - (a_c^{in}, (i, \perp))$, where

- $a' = a_e^{out}$ if $\text{eval}(\mathbf{Expr}^b(e), \mathbf{Val}) = \text{true}$ for some event $e \in c^\bullet - E_f$, or
- $a' = (c, e)$, if $\text{eval}(\mathbf{Expr}^b(e), \mathbf{Val}) = \text{true}$ for some final event $e \in c^\bullet \cap E_f$, or
- $a' = (c, c')$ for some $c' \in C$.

- (f) **or split rule** Let a_c^{in} be the ingoing arc of an **or** split connector c such that there is a process folder on a_c^{in} . The rule results in removing this folder from a_c^{in} and placing a process folder with the same folder number on at least one of the outgoing arcs of the non-final condition events that have a true boolean expression or the outgoing arcs of the **or** split connector leading to final condition events with boolean expression evaluated to true or to connectors. Formally, if $(a_c^{in}, (i, \perp)) \in m$, for some $i \in \mathbb{N}$ and $c \in C_{os}$, then $s \rightarrow s'$, with $s' = (m', \mathbf{Val})$ and $m' = m - (a_c^{in}, (i, \perp)) + \sum_{a \in A' \cup A''} (a, (i, \perp))$, where $A' \subseteq \{a_e^{out} | e \in c^\bullet \cap (E_c - E_f) \wedge \text{eval}(\mathbf{Expr}^b(e), \mathbf{Val}) = \text{true}\}$ and $A'' \subseteq \{(c, e) \in A_c^{out} | e \in c^\bullet \cap (E_c \cap E_f) \wedge \text{eval}(\mathbf{Expr}^b(e), \mathbf{Val}) = \text{true}\} \cup \{(c, c') \in A_c^{out} | c' \in C\}$ and $A' \cap A'' \neq \emptyset$.

- (g) **and join rule** Let A_c^{in} be the set of all incoming arcs of an **and** join connector c . If all arcs of A_c^{in} have process folders with the same folder number, the rule can be applied resulting in the removal of these process folders from A_c^{in} , and the production of a process folder with the same process folder number as the original folders on the outgoing arc of the connector. Formally, if there is a $c \in C_{aj}$ such that $(a, (i, \perp)) \in m$, for all arcs $a \in A_c^{in}$ and some $i \in \mathbb{N}$, then $s \rightarrow s'$ with $s' = (m', \mathbf{Val})$, and $m' = m + (a_c^{out}, (i, \perp)) - \sum_{a \in A_c^{in}} (a, (i, \perp))$.

- (h) **xor join rule** Let a be an incoming arc of an **xor** join connector such that there is a process folder on a . Then, the folder is removed from a and placed on the outgoing arc of the connector. Formally, if $(a, (i, \perp)) \in m$ for some $i \in \mathbb{N}$, $c \in C_{xj}$ and $a \in A_c^{in}$, then $s \rightarrow s'$ with $s' = (m', \mathbf{Val})$, $m' = m + (a_c^{out}, (i, \perp)) - (a, (i, \perp))$.

- (i) **or join rule** Let A' be the set of incoming arcs of an **or** join connector that have process folders with timers off and the same folder number. The rule consists of one or more steps (depending on whether the connector has a synchronization time or not):
- (**or** join unsynchronized) In case the **or** connector does not have a synchronization timeout, the rule results in removing all the folders with timers off and the same folder number on A' and producing a process folder with the same folder number as the original folders and the timer off on the outgoing arc of the connector. Formally, if there is a $c \in C_{\mathbf{oj}} - C_s$ such that $A' = \{a \in A_c^{\text{in}} \mid (a, (i, \perp)) \in m\} \neq \emptyset$, for some $i \in \mathbb{N}$, then $s \rightarrow s'$ with $s' = (m', \mathbf{Val})$ and $m' = m + (a_c^{\text{out}}, (i, \perp)) - \sum_{a' \in A'} (a', (i, \perp))$.
 - (**or** join synchronized waiting) Let A'' be a set of incoming arcs of a synchronized **or** join connector that have waiting process folders on them with the same folder numbers as the folders on A' . The rule results in removing the process folders with timers off and the same folder number on A' and producing new process folders on A' having the same folder number as the removed ones and a timestamp that is either the synchronization time of the **or** connector in case A'' is empty or the timestamp of the folders in A'' if A'' is non-empty. Formally, let $A' = \{a = (x, c) \in A \mid x \in N \wedge (a, (i, \perp)) \in m\} \neq \emptyset$ and $A'' = \{a \in A_c^{\text{in}} \mid (a, (i, t'')) \in m \wedge t'' > 0\}$ for some $i \in \mathbb{N}$ and $c \in C_s$. If $A'' = \emptyset$ then let $t' = t$ be the timestamp of the process folders $p = (a, (i, t)) \in m$, where $a \in A''$, otherwise let $t \in \text{Dom}(\text{pdf}_f)$. Then, $s \rightarrow s'$ with $s' = (m', \mathbf{Val})$ and $m' = m + \sum_{a' \in A'} (a', (i, t')) - \sum_{a' \in A'} (a', (i, \perp))$.
 - (**or** join synchronized firing) Let A'' be the set of incoming arcs of an **or** connector that have an active process folder on them and all these folders on A'' have the same folder number. Then these process folders are removed from A'' and a new process folder is produced on the outgoing arc of the connector, so that it has the same folder number as the original folders and the timer off. Formally, if for some $c \in C_{\mathbf{oj}} - C_s$ and $i \in \mathbb{N}$, $A'' = \{a \in A_c^{\text{in}} \mid (a, (i, 0)) \in m\} \neq \emptyset$, then $s \rightarrow s'$ with $s' = (m', \mathbf{Val})$, where $m' = m + (a_c^{\text{out}}, (i, \perp)) - \sum_{a' \in A''} (a', (i, 0))$.
- (j) **time step rule** This rule has the lowest priority, i.e. the rule is applied if no other rule can be applied. Time passage is applied when all process folders with timers on in a state are waiting (have a strictly positive timestamp) and results in decreasing the timestamp of all process folders of the state by the minimal timestamp of the folders with timers on. Formally, if $P' = \{p \in m \mid p_t > 0\} \neq \emptyset$ and $s_t = \min\{p_t \mid p \in P'\} > 0$ and there is no other state $s'' \neq s'$ such that $s \rightarrow s''$, then $s \rightarrow s'$, with $s' = (m', \mathbf{Val})$, where $m' = m + \sum_{(x, (i, t)) \in P'} (x, (i, t - s_t)) - \sum_{p \in P'} p$.

Remark 5 (Uniqueness of newly generated folder numbers). Folder numbers generated at some start event set E_s^d ($d \in I_s$) have the form $d + k \cdot n$, where $n = |I_s|$ and $k \in \mathbb{N}$. Therefore, all folder numbers produced at E_s^d are equal modulo the

index number of the start event set, i.e. d . As a result, all process folder numbers that are generated are unique.

Remark 6 (Time progress). The time step rule decreases the timestamp of a waiting process folder until it becomes active (its timer expires). Note that all other steps have the same higher priority than the time progress and their semantics is interleaving. The time progress problem reduces to the situation where no other rule can be applied or to the situation where there is an infinite sequence of (timeless) rules that can be applied. We therefore assume that there is a finite number of process folder numbers such that the states contain a finite number of process folders with the same folder number.

4 Verification of eEPCs Using CPN Tools

To verify the correctness of eEPCs, we use the CPN Tools [3] which are based on colored Petri nets [10] as modeling and analysis language. (Timed) Colored Petri nets (TCPNs) combine the expressive power of classical PNs, which are suitable for modeling the logical behavior of a control flow, with the modeling of data and time by means of timed color sets and a global clock. A state of a TCPN is called a *marking* and represents the distribution of (timed) colored tokens on places. Tokens belonging to a timed color set have a timestamp and they can only be used if the value of the timestamp is less than the value of the global clock.

Let G_e be an arbitrary eEPC and $TS = (\Sigma, s_0, \rightarrow)$ the transition system describing the semantics of G_e . We denote the timed integer color corresponding to the set of process folders numbers by **PF**, and for each capacity resource and data attribute, we define a place having the corresponding untimed color type. The locations at which process folders can reside correspond to TCPN places having type **PF** and local steps in TCPNs are depicted by transitions. A step that can be taken in the eEPC corresponds to a transition firing, given preconditions and postconditions expressed as expressions on arcs or guards (boolean expressions) on transitions. The global clock (model time) advances the timestamps of all timed tokens with the smallest amount of time needed to make at least one token active, which corresponds to the time step rule in eEPCs in which timers decrease their values. Token delays are positioned on transitions or on outgoing arcs of transitions.

4.1 Transformation of eEPCs into TCPNs

For mapping of eEPCs into TCPNs, we first identify generic eEPC patterns and provide their translation into TCPN patterns, and then we show how the obtained TCPN patterns can be fused together to form a TCPN.

We define eEPC patterns taking into account the rules of the semantics given in Section 3 (except for the time step rule), covering all patterns that would allow us to build an arbitrary eEPC. An eEPC pattern consists of incoming and outgoing arc(s) and other elements necessary for the rule to occur.

Figures 3(a-i) and 4 show instances of these patterns having the incoming (outgoing) arcs dotted and their corresponding TCPN pattern. In what follows we describe these transformations in more detail.

Start event set pattern. Let E_s^d be a start event set for some $d \in I_s$ and let $n = |E_s^d|$. The corresponding TCPN pattern is represented by a place which is initially marked with a token $d@0$ (with timestamp 0) and where a newly generated token with a delay is deposited; a transition that generates a new token, adds a delay to its timestamp and puts the token from the former place on the places corresponding start events of the start event set. Figure 3(a) shows the eEPC pattern and the TCPN pattern for a start event set with two start events.

Event pattern. Let $e \in E - E_s - E_f - E_c$ be an event. The corresponding TCPN pattern is shown in Figure 3(b). Note that final events are not translated.

Function pattern. Let f be a (timed) function connected to the set of resources \mathcal{R}_f and to the set of data attributes \mathcal{A}_f . The corresponding TCPN pattern has two transitions and an intermediate place depicting the two steps of the function rule. Figure 3(c) shows the pattern and its translation operating on an attribute, a resource that is used, a resource that is produced and a resource that is consumed. The operations on the resources and attributes are described on the arc inscriptions.

and split, and join, and xor join patterns. These patterns are parameterized by the number of the outgoing and incoming arcs, respectively. Figure 3(d-f) shows the respective patterns with two incoming, respectively outgoing arcs;

xor split pattern. Let c be an **or** split connector. The TCPN pattern consists of a place with outgoing transitions. The boolean expressions on the condition events of c (on resources or attributes) become guards for the transitions. Figure 3(g) shows an **or** split connector having a condition event linked to an attribute or resource, a condition event with condition *true* and a connector on the outgoing arcs and the corresponding TCPN pattern.

or split pattern. Let c be an **or** split connector. The TCPN pattern contains a transition and a non-deterministic choice is implemented in the code segment of the transition by means of generation of boolean variables for each outgoing arc of the transition corresponding to outgoing arcs leading to non-conditional events. Figure 3(h) shows an instance of the pattern and its translation. In case at least one of the boolean expressions corresponding to conditions on attributes or resources is true, a boolean variable generated for each arc that does not lead to a condition event with conditions on attributes or resources. In case all the boolean expressions on attributes or resources are false or there are no condition events on data attributes or resources, the boolean variables generated for the rest of the arcs must have at least one *true* value. The boolean expressions on the condition events and the boolean values generated become conditions in the arc inscriptions.

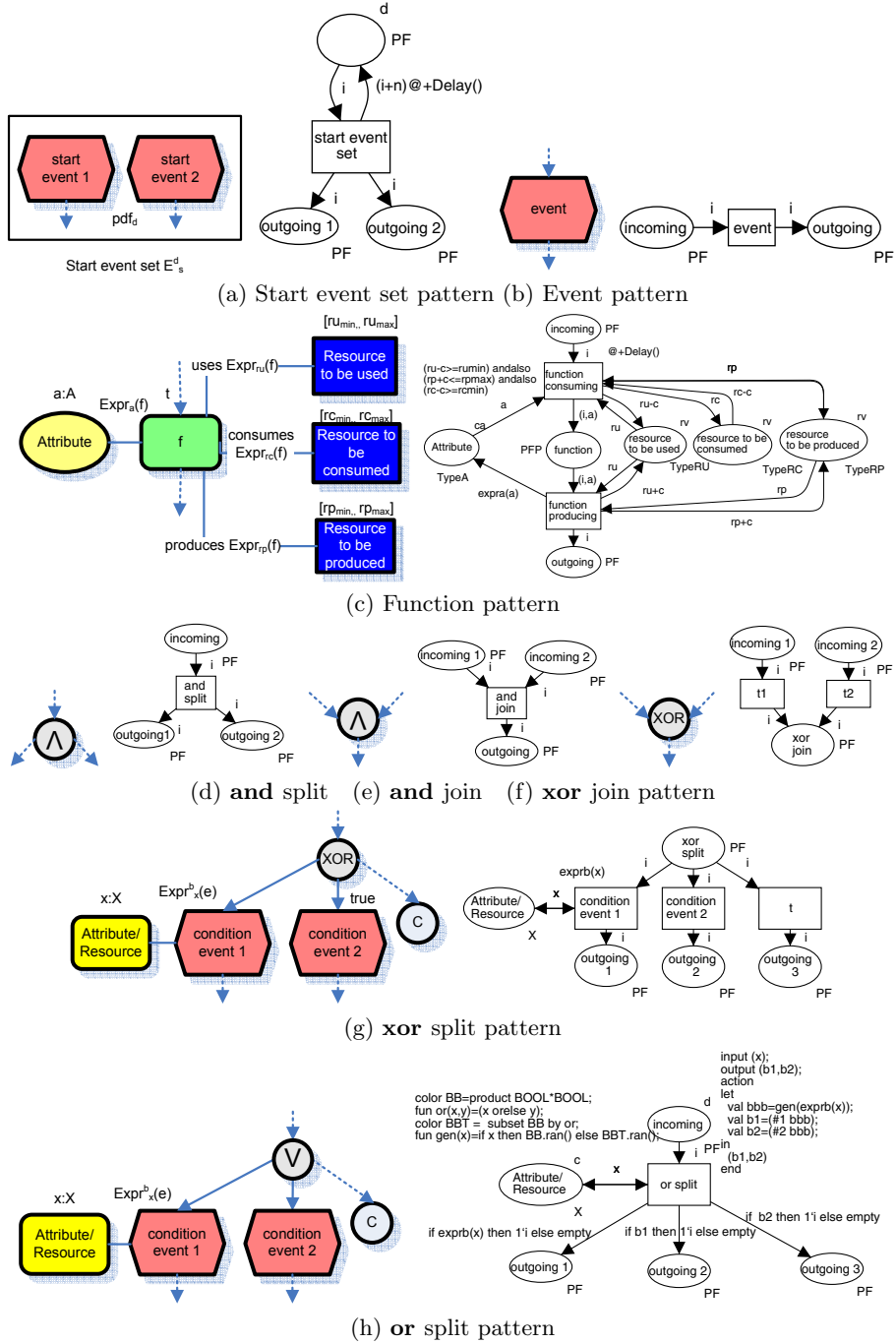


Fig. 3. Translation of the eEPC patterns into TCPN patterns

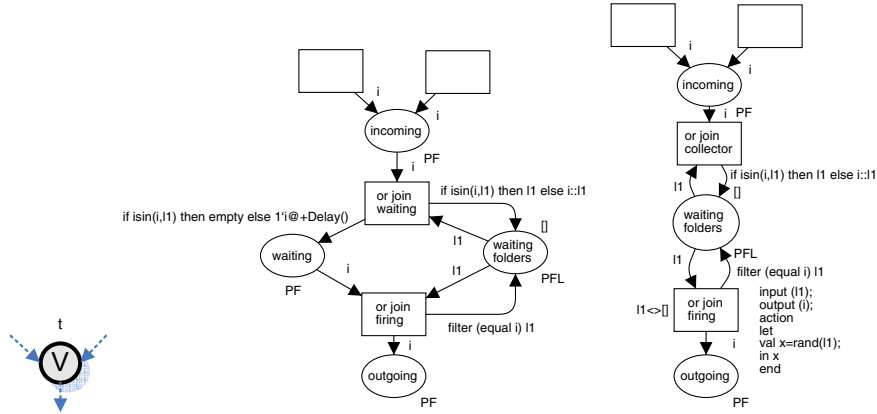


Fig. 4. or join pattern (synchronized and unsynchronized)

or join pattern. Figure 4 shows the pattern with two incoming arcs. In the TCPN pattern, place *incoming* collects all the tokens on the incoming arcs. The **or join** waiting transition decides whether the token will receive a delay by checking if the folder number is already in the list of waiting folders (in the place *waiting folders*). The *or join firing* transition becomes enabled, then the timestamp of the tokens in place *waiting* expires and the firing removes the token from the list of waiting tokens present in the place *waiting folders*. Note that in the untimed version the place *waiting* is eliminated.

Two eEPC patterns are called *adjacent* if an outgoing arc of a pattern coincides with an ingoing arc of another pattern. Two TCPN patterns are called *adjacent* if the corresponding eEPC patterns are adjacent. Adjacent TCPN patterns are fused, i.e. for every two adjacent patterns, if they have the same input and output nodes (e.g. both are either transitions or places), then the two nodes are fused; otherwise, a directed arc is added between them.

4.2 Verification

A TCPN obtained using the translation procedure described in the previous section can be simulated and analyzed by the CPN Tools [3], using state-space analysis (which is basically an exhaustive search through all possible states of the model).

The first check on eEPCs to be performed is whether the semantics of diverse connectors is respected. For **xor** join connectors, this coincides with checking whether there are reachable markings having two tokens in the places corresponding to **xor** join connectors. If there are, we can conclude that the eEPC model is not correct and we can provide a simulation of the TCPN that leads to this error.

If an **or** split has condition events on all its outgoing arcs, at least one of the conditions should be evaluated to *true* at every reachable marking. The violation of this requirement can be easily checked by finding a marking with at least one token on the place corresponding to the incoming arc of the **or** split such that all boolean expressions on resources or data attributes are *false*.

CPN Tools can provide a report on the state space of the constructed TCPN that includes information about dead markings (marking at which no transition is enabled). In case the only dead markings of the TCPN are markings having tokens on the places corresponding to incoming arcs of final events, we can conclude that the original eEPC *terminates properly*. Dead markings can also provide information about deadlocks in the eEPC, e.g. functions that cannot execute due to non-availability of resources or non-synchronization. Furthermore, the CPN Tools can verify properties of the model which are defined as temporal logic formulas in ASK-CTL [4].

5 Related and Future Work

Related work. There are different approaches to the formalization of the syntax and semantics of EPCs.

One approach is to use Petri nets to specify their semantics. An EPC is translated into a PN using a set of transformation rules. The semantics of EPCs is defined as the semantics of resulting Petri nets. Van der Aalst [1] and Dehnert [7] use a subclass of PNs — workflow nets that is suitable to describe EPCs and use specific verification methods developed for PNs in order to verify EPCs. In [1], an EPC is considered to be correct if and only if the workflow obtained as the translation of an EPC is sound. Langner, Schneider and Wehler [13] use a transformation into boolean nets which are colored Petri nets with a single color of type boolean and formulas from the propositional logic as guards. The correctness criterion is the well-formedness of the corresponding boolean net, which is too strict for some practical applications.

Another approach is to consider the transition systems-based semantics. In [16], [2] and [12], the dynamic behavior of an EPC is defined in terms of transition systems. In [2] and [12], the state of an EPC is defined as a mapping of the set of arcs to $\{0, 1\}$ and is represented by the presence or absence of process folders on the arcs of the EPC. Moreover, [2] proposes a non-local semantics of the **xor** and **or** join connector that refers to checking certain conditions that depend on the overall behavior of the EPC. An **xor** join has to wait for a folder on one of its input arcs in order to propagate it to its output arc. However, if a process folder is present or could arrive at some other input arc, the **xor** join should not propagate the folder. For an **or** join, the propagation of a process folder from its input arcs is delayed as long as a process folder could possibly arrive at one of the other input arcs. Computing the transition relation of an EPC has been implemented and tested in [5] using symbolic model checking.

In our paper we consider the semantics of extended event-driven process chains, i.e. EPCs extended with data, resources, and time as it is specified in the

ARIS Toolset [9]. We provide a formal definition of their semantics in terms of a transition system. This can be further used as a base for behavioral (functional) verification of eEPCs using different model checkers.

Furthermore, we provide a translation to timed colored Petri nets and formulate some correctness criteria for eEPCs that can be checked on the translated eEPCs using CPN Tools.

Future work. For future research, it would be interesting to consider an automatic translation of eEPCs to colored Petri nets and perform verification experiments on large case studies. Since verification by model checking could lead to a blow-up of the state space, reduction techniques would be beneficial (see [8] for an overview). Another line of investigation is the development of a classification of correctness requirements for business processes like it is already done for software processes [14].

References

1. W. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650, 1999.
2. W. M. P. van der Aalst, J. Desel, and E. Kindler. On the semantics of EPCs: A vicious circle. In *EPK 2002, Proceedings des GI-Workshops und Arbeitskreistreffens (Trier, November 2002)*, pages 71–79. GI-Arbeitskreis Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 2002.
3. The CPN Tools Homepage. <http://www.daimi.au.dk/CPNtools>.
4. A. Cheng, S. Christensen, and K. Mortensen. Model Checking Coloured Petri Nets Exploiting Strongly Connected Components. In M. Spathopoulos, R. Smedinga, and P. Kozak, editors, *Proceedings of the International Workshop on Discrete Event Systems, WODES96*, pages 169–177, 1996.
5. N. Cuntz and E. Kindler. On the semantics of EPCs: Efficient calculation and simulation. In M. Nüttgens and F. J. Rump, editors, *EPK 2004: Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, Gesellschaft für Informatik, pages 7–26, Bonn, 2004.
6. R. Davis. *Business Process Modeling with ARIS: A Practical Guide*. Springer-Verlag, 2001.
7. J. Dehnert. *A Methodology for Workflow Modeling - From business process modeling towards sound workflow specification*. PhD thesis, TU Berlin, 2003.
8. C. Girault and R. Valk. *Petri Nets for Systems Engineering - A Guide to Modeling, Verification, and Applications*. Springer, 2003.
9. IDS Scheer AG. *ARIS Methods Manual*, 2003.
10. K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical*. Springer-Verlag, 1992.
11. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley, 1998.
12. E. Kindler. On the semantics of EPCs: A framework for resolving a vicious circle. In J. Desel, B. Pernci, and M. Weske, editors, *Business Process Management, BMP 2004*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2004.

13. P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event-Driven Process Chains. In *19th Int. Conf. on Application and Theory of Petri Nets*, volume 1420 of *LNCS*, pages 286–305. Springer, 1998.
14. G. S. A. Matthew B. Dwyer and J. C. Corbett. Patterns in Property Specifications for Finite-state Verification. In *Proceedings of the 21st International Conference on Software Engineering*, 1999.
15. K. G. Nüttgens and A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). Technical report, Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken, 1992.
16. M. Nüttgens and F.J.Rump. Syntax und Semantik Ereignisgesteuerter Processketten (EPK). In J. Desel and M. Weske, editors, *Promise 2002- Processorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, volume LNI, pages 64–77, 2002.
17. A.-W. Scheer. *ARIS : business process modeling*. Springer-Verlag, Berlin, 2nd edition, 1998.