# Parameterized Partial Orders for Modeling Embedded System Use Cases:
## Formal Definition and Translation to Coloured Petri Nets

Nikola Trčka* and Marc Voorhoeve*† and Twan Basten*‡
* *Eindhoven University of Technology, The Netherlands*
† *Open University Netherlands, The Netherlands*
‡ *Embedded Systems Institute, The Netherlands*

***Abstract*—Model-driven Design-Space Exploration (DSE) for embedded systems has proven to speed up system design and improve quality. Parameterized Partial Orders (PPOs) are a simple yet powerful conservative extension of classical partial orders. They serve as an intermediate representation in our Octopus toolset, allowing to capture applications from different domains and enabling analysis with various tools. We present PPOs, their translation to Coloured Petri Nets, and their use in DSE for a printer case study.**

## I. Introduction

In the early stages of embedded system design, many implementation possibilities need to be considered. Design spaces are typically huge, involving multiple metrics of interest (timing, resource utilization, energy usage, cost, etc.) and multiple design parameters (e.g. amount and type of processing units, memory size and organization, interconnection, scheduling and arbitration policies). The relation between design choices on the one hand and metrics of interest on the other is often difficult to establish, due to aspects such as concurrency, dynamic application behavior, and resource sharing.

To achieve quick, yet high-quality results, most vendors currently adopt some form of model-driven design. Formal models are developed and subsequently verified against various requirements. Many tool-based modeling and analysis approaches do exist, each with its merits and demerits. However, no single tool addresses all challenges of modern embedded-system design.

In order to reduce the dependency upon a single tool, we recently proposed an integrated Design Space Exploration (DSE) framework, called the *Octopus toolset* [1], [2], that tries to leverage the modeling and analysis power of various tools. The Octopus toolset supports provides support to model, analyze in different ways, and finally select appropriate design alternatives in the early phases of product development. It has been implemented within the context of the Octopus project [3], a joint project of a consortium of industrial and academic partners, with Océ Technologies B.V. as the main industrial partner and case-study provider.

This work was carried out as part of the Octopus project with Océ Technologies B.V. under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the BSIK program.

The approach to connect different tools requires a generic format that can be supported by any known tool. This intermediate representation must provide the constructs to facilitate translation to the different target analysis tools, yet it must be powerful and expressive enough to accommodate developers. The Octopus toolset introduces DSEIR (Design Space Exploration Intermediate Representation), a powerful, yet simple formalism to facilitate reuse of models across tools. The added value of DSEIR is in providing consistency between analysis results, in allowing for quicker model development and adaptation, and in making the toolset less dependent on modifications in the back-end tools.

DSEIR adopts the Y-chart philosophy [4] popular in hardware design. It is based on the observation that design space exploration process typically involves the co-development of an *application* (system use-cases), a *platform* (system resources), and the *mapping* of the application onto the platform (defining which resources are involved in which tasks). DSEIR specifications fully respect this subdivision, allowing different parts of design to be explored in isolation.

In this paper we focus on the modeling of use cases only, i.e., the application side of DSEIR. We introduce *parameterized partial orders* (PPOs) to model sets of causally dependent tasks in a declarative way. In PPO models, tasks are parameterized according to their task class, while the causal dependencies are modeled by predicates.

As indicated in [5], any distributed computation is essentially a partial order (PO). This justifies their use as an intermediate representation within DSEIR. Adding parameters allows a compact and convenient description (the PPO in Fig. 2, e.g., describes a partial order containing more than $2{\cdot}S{\cdot}C$ individual tasks, where $S$ and $C$ represent the number of scan and copy pages). Despite its power, the parameter system of PPOs is designed to be simple and easily supported by tools that process DSEIR models.

It is important to note that we do not aim at defining yet-another model of computation. We see PPOs only as a convenient extension of classical partial orders (or tasks graphs), motivated by a rather pragmatic need for efficiency and compactness. In fact, as shown in the paper, the addition of parameters is conservative and every PPO can in principle be converted to a PO. Using this approach in translations, however, is rarely desirable. It typically results in a huge

number of tasks, and does not take advantage of the (usually very efficient) data support of the target language.

One of the analysis back-ends in the Octopus toolset is CPN Tools [6], a well-known simulation and verification framework based on (coloured) Petri nets (CPNs). This tool enables performance evaluation of DSEIR models, and was chosen for its efficient, yet formally specified and user-friendly simulator. The other Octopus back-end is UPPAAL [7], used for model-checking and scheduling synthesis.

In this paper we define the PPO to CPN translation that has been implemented in Octopus, and we formally prove the correctness of this translation. Interesting enough, the semantics of Petri nets is highly operational, which makes the generic translation inefficient and far from trivial. To solve this problem, we identify several (syntactical) restrictions of the PPO model, all very common in practice, that allow for much simpler translations and consequently result in nets that can be more efficiently analyzed.

The results of this paper are illustrated on an example from the domain of professional printers. Although simplified, this example exhibits the features of several realistic case studies we have successfully conducted at Océ.

The paper is structured as follows. Section II contains some preliminaries. Section III defines the PPO model with the running example. Section IV addresses syntax and semantics of CPN. Section V tackles the problem of PPO to CPN translation. The last section concludes the paper.

## II. Preliminaries

In this section we define some useful constructs and introduce notation that will be used in the sequel.

### A. Sets and functions

The set of natural numbers is denoted $\mathbb{N}$. If $A$ is a set, then $\mathcal{F}(A)$ is the set of all finite subsets of $A$. The symbol $\hookrightarrow$ denotes partial mappings. A (partial) mapping from $A$ to $B$ is seen as a (special) subset of $A \times B$. If $f, g$ are mappings satisfying $\mathrm{dom}(f) \cap \mathrm{dom}(g) = \emptyset$, then $f \cup g$ is also a mapping.

### B. Lists and multisets

For $n \geq 0$, the set $A^n$ consists of $n$-tuples $\langle a_1 \dots a_n \rangle$ with $\forall 1 \leq i \leq n : a_i \in A$. The set $A^*$ of $A$-lists equals $\bigcup_{n \geq 0} A^n$ ($A^0$ equals the set $\{\langle\rangle\}$, containing the empty list $\langle\rangle$ as its sole element). If $\ell = \langle a_1 \dots a_n \rangle$ is an $A$-list and $m : A \to B$, then $m(\ell)$ denotes the $B$-list $\langle m(a_1) \dots m(a_n) \rangle$. If $A$ is a set of sets and $\ell = \langle \alpha_1 \dots \alpha_n \rangle$ an $A$-list, then $\prod \ell$ denotes the set of lists $\{\langle a_1 \dots a_n \rangle \mid a_i \in \alpha_i \text{ for all } 1 \leq i \leq n\}$.

A *bag* (or *multiset*) $m$ over set $A$ is a mapping $m : A \to \mathbb{N}$. We denote the set of all bags over $A$ by $\mathrm{bag}(A)$. We use $+$ and $-$ for the sum and the difference of two bags, defined in a standard way. For example, we write $m = 2[p] + [q]$ for a bag $m$ with $m(p) = 2$, $m(q) = 1$ and $m(x) = 0$ for $x \notin \{p, q\}$. We write $[]$ for the empty bag. We overload $\in$ to denote bag inclusion: $a \in A$ iff $A(a) > 0$.We also use the shortcut notation $[p_1, \dots, p_n]$ for the bag $[p_1] + \dots [p_n]$. If $S = \{s_1, \dots s_n\}$ is a set, then $[S]$ denotes the bag $[s_1, \dots s_n]$. In expressions $[\{f(x_1, \dots, x_n) \mid C(x_1, \dots, x_n)\}]$, we omit the braces.

### C. Data operations

The sets $\mathsf{VAL}, \mathsf{TYPE}, \mathsf{VAR}$ and $\mathsf{EXP}$ are respectively the set of *values*, *types*, *variables* and *expressions*. Every type is a set of values. The set of all values, i.e. the set $\bigcup \mathsf{TYPE}$, is denoted $\mathsf{VAL}$. We assume that $\mathsf{TYPE}$ contains standard types like $\mathsf{Bool}$, $\mathsf{Nat}$, and $\mathsf{Int}$ and that it also allows n-ary cartesian products and bag formation. Tupling is effectuated by enclosing the comma-separated list of its elements between brackets. So if $A, B$ are types containing values $a, b$ respectively, then $(a, b)$ is a value of type $A \times B$.

The function $\mathsf{typ} : \mathsf{VAR} \to \mathsf{TYPE}$ assigns a type to every variable. The set $\mathsf{EXP}$ is built from values, variables, and standard operators. It is strongly typed, i.e. we can extend typ to $\mathsf{EXP}$ to determine $\mathsf{typ}(e) \in \mathsf{TYPE}$ for every $e \in \mathsf{EXP}$. We define *predicates* as expressions of boolean type: $\mathsf{BoolEXP} = \{e \in \mathsf{EXP} \mid \mathsf{typ}(e) = \mathsf{Bool}\}$. The function $\mathsf{var} : \mathsf{EXP} \to \mathcal{F}(\mathsf{VAR})$ returns the set of variables in an expression. An expression $e$ without variables is always implicitly *evaluated*, yielding a value in $\mathsf{typ}(e)$.

A *valuation* is a partial function $\sigma : \mathsf{VAR} \hookrightarrow \mathsf{VAL}$ satisfying $\sigma(x) \in \mathsf{typ}(x)$ for every $x \in \mathrm{dom}(\sigma)$. The set of all valuations is $\Sigma$. If $e \in \mathsf{EXP}$ and $\mathsf{var}(e) \subseteq \mathrm{dom}(\sigma)$, then $\sigma(e)$ is a value obtained by substituting in $e$ every occurrence of a variable $x \in \mathrm{dom}(\sigma)$ by $\sigma(x)$ and evaluating the result.

## III. Parameterized Partial Orders

In this section we introduce parameterized partial orders (PPOs). We first define partial orders (of tasks) and their operational semantics. Next, we explain the extension with parameters and the visualization of PPOs. Finally, we define the PPO model of a printer use case.

A *partial order* (PO) is a pair $(T, P)$, where $T$ is a set of *tasks* and $P \subseteq T \times T$ is the *precedence relation*. Intuitively, if $(t, u) \in P$, then task $u$ can be executed only if task $t$ has completed.

The *state* of the PO $(T, P)$ corresponds to a subset $\theta$ of $T$ satisfying $(u \in \theta \wedge (t, u) \in P) \Rightarrow t \in \theta$. The PO $(T, P)$ can move from state $\theta$ to state $\theta'$ by executing task $u \in T$ (denoted $\theta \xrightarrow{u} (\theta \cup \{u\})$)iff $u \in (T \setminus \theta) \wedge (\forall t \in T : (t, u) \in P \Rightarrow t \in \theta)$. Initially, a PO is in state $\emptyset$. State $\theta = T$ is considered final, indicating *successful termination*.

The PO $(T, P)$ is called *consistent* if there is no $t \in T$ such that $(t, t) \in P^+$, where $P^+$ is the transitive closure of $P$. For finite POs, this is equivalent to saying that the final state is reachable from the initial state.

### A. Adding parameters

As explained before, to facilitate the specification of large (even infinite) POs in which the sets of tasks follow a certain syntactical pattern, we introduce the concept of parameters. Every task is associated a set of variables (called

the parameters) that is used to compactly represent instances and precedence relations of this task.

We assume that VAR is decomposed into $\mathsf{VAR} = \mathsf{UVAR} \cup \mathsf{DVAR}$, where UVAR is a set of *undecorated* variables and $\mathsf{DVAR} = \{x' \mid x \in \mathsf{UVAR}\}$ is the set of variables *decorated* with a prime.We require that $\mathsf{typ}(x') = \mathsf{typ}(x)$ for every $x \in \mathsf{UVAR}$.

Given a valuation $\sigma \in \Sigma$ with $\mathrm{dom}(\sigma) \subseteq \mathsf{UVAR}$, we define $\sigma' : \mathsf{DVAR} \hookrightarrow \mathsf{VAL}$ by $\sigma'(x') = \sigma(x)$ for each $x \in \mathrm{dom}(\sigma)$. If $\mathsf{var}(e) \subseteq \mathsf{UVAR}$ then $e'$ is the expression obtained from $e$ by decorating each variable. We extend $\sigma'$ to such expressions $e'$ by setting $\sigma'(e') = \sigma(e)$.

*Definition 1 (Parameterized Partial Order - PPO):* A PPO is a tuple $(T, V, S, P)$, where

- $T$ is a finite set of *task classes*;
- $V : T \to \mathcal{F}(\mathsf{UVAR})$, associates a finite set of undecorated index variables to every task class. For $t \in T$, we define $V'(t) = \{x' \mid x \in V(t)\}$;
- $S : T \to \mathsf{BoolEXP}$, with $\mathsf{var}(S(t)) \subseteq V(t)$ for all $t \in T$, determines the *scope* of parameters for every task class (in the form of a predicate);
- $P : (T \times T) \hookrightarrow \mathsf{BoolEXP}$, with $\mathsf{var}(P(t,u)) \subseteq V(t) \cup V'(u)$ for $(t,u) \in \mathrm{dom}(P)$, gives a *condition* under which a precedence between two task class instances exists (variables in $u$ are decorated to avoid ambiguity).

$\mathsf{UVAR} = \{n, m, k\}$, $\mathsf{typ}(n) = \mathsf{typ}(m) = \mathsf{typ}(k) = \mathsf{Nat}$,
$T = \{A, B, C\}$,
$V(A) = \{n\}, V(B) = \{n, k\}, V(C) = \{m, n\}$,
$S(A) \;=\; \text{"}1 \leq n \leq 10\text{"}$,
$S(B) \;=\; \text{"}1 \leq n \leq 10 \wedge (n-k) \bmod 3 = 0\text{"}$,
$S(C) \;=\; \text{"}1 \leq n \leq 10 \wedge m+n < 20\text{"}$,
$P(A,B) \;=\; \text{"}n = n'+k'\text{"}$,
$P(A,C) \;=\; \text{"}m' \geq n\text{"}$

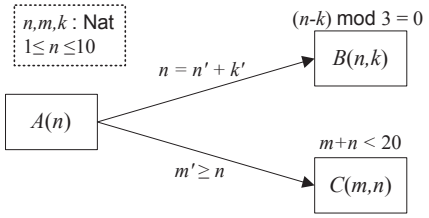


Figure 1. A PPO and its visualization

Fig. 1 gives an example of a PPO and shows its visualization. PPOs are visualized similarly to regular POs but with several extensions. All the variables that are connected to a certain task class by means of function $V$ are shown as (functional) parameters of this class. The scope condition of a class $x$ (specified by $S(x)$) is shown above the node representing this class (omitted if constant true). In addition, global conditions that hold for every $S(x)$ are indicated and shown in combination with the set UVAR. Arc labels contain the conditional expression determined by $P$ (the label is omitted if the expression equals constant true and the arc is not shown if $P$ is false).

As mentioned, PPOs are only used to define large POs concisely; they add no expressive power to regular POs, i.e., every PPO can be converted to a (possibly inifinite) PO.

*Definition 2 (Parameter-Unfolding):* Let $(T, V, S, P)$ be a PPO and let $t \in T$. We define $\Sigma_t = \{\sigma \in \Sigma \mid \mathrm{dom}(\sigma) = V(t) \wedge \sigma(S(t)) = \mathsf{true}\}$ as the set of $t$-valuations and $t_V = \{(t, \sigma) \mid \sigma \in \Sigma_t\}$ as the set of all *instances* of $t$. We define $\bar{T} = \bigcup_{t \in T} t_V$ and $\bar{P} = \{((t, \sigma_t), (u, \sigma_u)) \in (\bar{T} \times \bar{T}) \mid (\sigma_t \cup \sigma'_u)(P(t,u)) = \mathsf{true}\}$. The PO $(\bar{T}, \bar{P})$ is called the *parameter-unfolding* of $(T, V, S, P)$. The *state* of a PPO is the state of its parameter-unfolding. Similarly, a PPO is *consistent* iff its parameter-unfolding is consistent.

The PPO from Fig. 1 defines the PO $(\bar{T}, \bar{P})$ satisfying

$$\begin{aligned}\bar{T} \;=\;& \{(A, \{(n,x)\}) \mid \phi(x)\} \\ & \cup \{B, \{(n,x),(k,y)\}) \mid \phi(x) \wedge \psi(x,y)\} \\ & \cup \{(C, \{(n,x),(m,z)\}) \mid \phi(x) \wedge z{+}x < 20\},\end{aligned}$$

$$\begin{aligned}\bar{P} \;=\;& \{((A, \{(n,x)\}), (B, \{(n,x'),(k,y')\})) \mid \\ & \quad \phi(x) \wedge \phi(x') \wedge \psi(x',y') \wedge x = x' + y'\} \\ & \cup \{((A, \{(n,x)\}), (C, \{(n,x'),(m,z')\})) \mid \\ & \quad \phi(x) \wedge \phi(x') \wedge x' + z' < 20 \wedge z' \geq x\},\end{aligned}$$

with $\phi(x) = 1 \leq x \leq 10$ and $\psi(x,y) = (x{-}y) \bmod 3 = 0$.

### B. Example: Modeling a printing use case

Our running example features a simplified *scan-to-print* job where scan pages are reduced in size and pairwise combined into print pages. A scan page is successively *scanned*, *resampled* and *diminished*. Diminished pages are combined into print pages for *printing*. Print pages are uploaded while concurrently a first copy is printed. Next copies are first downloaded and then printed. The print order requires all pages of a copy to be printed before moving to the next copy. The PPO defining this printer use-case is shown in Fig. 2, with a partial unfolding at the bottom.

The PPO parameters are $s$ (scan page), $p$ (print page), and $c$ (copy). The relation between scan and print pages is defined by $p = \mathsf{hlv}(s)$, where $\mathsf{hlv}(s) = (s{+}1) \operatorname{div} 2$. So $p = 1$ corresponds to $s \in \{1, 2\}$, $p = 2$ to $s \in \{3, 4\}$ and so on.

The PPO contains global constants $S$ (the number of scan pages) and $C$ (the number of copies to be printed per page); the parameters satisfy the global constraints $1 \leq s \leq S$, $1 \leq c \leq C$ and $1 \leq p \leq \mathsf{hlv}(S)$.

The scan, resample and diminish tasks have parameter $s$, the combine and upload tasks have parameter $p$, the download and print tasks have parameters $p, c$. The unfolding shows that only task *start scan*$(1)$, having no predecessors can start initially, concurrently enabling tasks *end scan*$(1)$ and *start resample*$(1)$. As the use case progresses, more tasks can run concurrently, exhibiting streaming behaviour. "Loops" in the PPO ensure that scanning and printing is performed in the prescribed order. A third loop ensures that uploading cannot get ahead of printing; while a page is printed, only the next page can be uploaded.

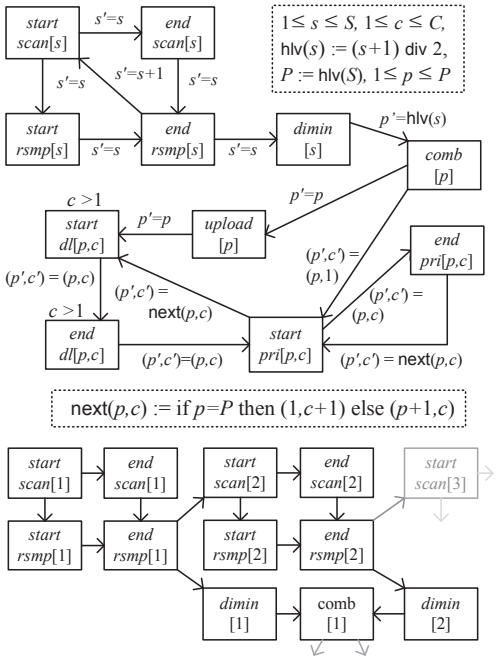


Figure 2. PPO model of a printer use-case

## IV. Coloured Petri nets

In this section we define the syntax and semantics of Coloured Petri nets.

*Definition 3 (Coloured Petri Net - CPN):* A *Coloured Petri Net (CPN)* is a tuple $N = (\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathcal{C}, \mathcal{E}, \mathcal{G})$ where:

- $\mathcal{P}$ is a set of *places*;
- $\mathcal{T}$ is a set of *transitions*, with $\mathcal{P} \cap \mathcal{T} = \emptyset$.
- $\mathcal{A} \subseteq \mathcal{P} \times \mathcal{T} \cup \mathcal{T} \times \mathcal{P}$ is a set of *arcs*. For $t \in \mathcal{T}$, we define $\mathcal{A}_t = ((\mathcal{P} \times \{t\}) \cup (\{t\} \times \mathcal{P})) \cap \mathcal{A}$;
- $\mathcal{C} : \mathcal{P} \to \mathsf{TYPE}$ associates *colours* (types) to places;
- $\mathcal{E} : \mathcal{A} \to \mathsf{EXP}$ is the set of *arc inscriptions* satisfying $\mathsf{typ}(\mathcal{E}(a)) = \mathsf{bag}(\mathcal{C}(p))$, for all $p \in \mathcal{P}$ and $a \in \mathcal{A}_p$;
- $\mathcal{G} : \mathcal{T} \to \mathsf{BoolEXP}$ is a *guard function* satisfying $\mathsf{var}(\mathcal{G}(t)) \subseteq \bigcup_{a \in \mathcal{A}_t} \mathsf{var}(\mathcal{E}(a))$.

Any function $m : P \to \mathsf{EXP}$ assigns to each place $p$ in its domain a variable-free expression $e$ satisfying $\mathsf{typ}(e) = \mathsf{bag}(\mathcal{C}(p))$ is called a *marking*. A *marked CPN* is a pair $(N, m)$ where $N$ is a CPN and $m$ is a marking (called the *initial marking*).

A marked CPN is depicted as a bipartite graph with an expression attached to each arc, a *guard* predicate to transitions and a type attached to each place (cf. Fig. 3). The guard predicate of a transition $t$ may be omitted if equal to true. Marking expressions indicating the initial set-up are shown near their place unless they denote an empty bag. Singleton bags are represented without square brackets.

To define the operational semantics of CP nets, we need the concept of a binding. For $t \in \mathcal{T}$, a valuation $b_t : \mathsf{VAR} \to \mathsf{VAL}$ is called a *t-binding* if $\mathrm{dom}(b_t) = \mathsf{var}(t)$, where $\mathsf{var}(t) = \bigcup_{a \in \mathcal{A}_t} \mathsf{var}(\mathcal{E}(a))$ is the set of variables of $t$. A *binding element* is a pair $(t, b_t)$ where $t \in \mathcal{T}$ and $b_t$ is a $t$-binding.

Let $m, m'$ be markings of $N$ and $(t, b)$ a binding element. We write $m \xrightarrow{(t,b)} m'$ iff $b(\mathcal{G}(t)) = \mathsf{true}$ and $m + O_b(t) = m' + I_b(t)$, with $I_b(t) = \sum_{p:(p,t) \in \mathcal{A}_t} (p, b_t(\mathcal{E}(p,t)))$ and $O_b(t) = \sum_{p:(t,p) \in \mathcal{A}_t} (p, b_t(\mathcal{E}(t,p)))$. This relation defines an one-step semantics for (marked) CPNs. A binding element $(t, b_t)$ is called *enabled* in a marking $m$ if there exists a marking $m'$ such that $m \xrightarrow{(t,b_t)} m'$. A transition $t$ with binding $b_t$ is thus enabled if there are enough input tokens of the right type and $\mathcal{G}(t)$ holds with $b_t$.

In the marked CPN from Fig. 3, the initial marking $m$ satisfies $m(p) = [1,2] \wedge m(q) = m(r) = m(s) = []$. Initially, transitions $t$ and $u$ are enabled with bindings $b_1$ and $b_2$ respectively, where $b_i(n) = [i]$ for $i \in \mathbb{N}$. After executing $(t, b_1)$, the token with value $b_1(n) = 1$ is removed from $p$ and a token with value $b_1(n+2) = 3$ is added to $p$ and another token with value $b_1(n) = 1$ is added to $q$. In this new state, $t$ and $u$ are enabled with respective bindings $b_3, b_2$. Executing $(u, b_2)$ will lead to the state $\{(p, [3,4]), (q, [1]), (r, [2,3]), (s, [])\}$. In this state, $t$, $u$ and $v$ are enabled, with respective bindings $b_3, b_4$ and $b_1$. Executing $(v, b_1)$ will lead to the state $\{(p, [3,4]), (q, []), (r, []), (s, [0])\}$.

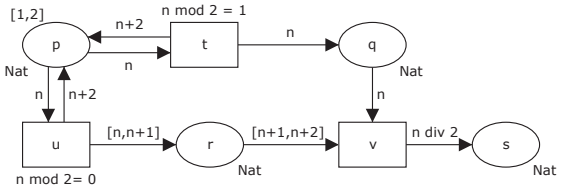


Figure 3. Example of a CP net

## V. From PPOs to Coloured Petri nets

Before defining the correspondence, we introduce some notation. Let $\mathcal{O} = (T, V, S, P)$ be a PPO. Given a set $V = \{x_1, \ldots x_n\}$ of variables, let $\ell(V)$ denote the *linearization* of that set, i.e. the tuple $(x_1, \ldots x_n)$ satisfying $x_1 < \ldots < x_n$ in the lexicographic ordering (decorating a variable does not change the order). Let $S'(t)$ be obtained from $S(t)$ by substituting each variable $x$ in the expression $S(t)$ by $x'$. Also, $P'(t,u)$ is obtained from $P(t,u)$ by simultaneously removing decorations from the decorated variables and adding decorations to undecorated ones.

The main idea of the PPO–to–CPN translation is to introduce a transition for every task class, a place for every task parameter, and a place for each precedence rule. Task parameters are simply taken one by one, while the inscriptions on arcs surrounding precedence-rule places contain parameter-bag expressions to ensure proper dynamics.

*Definition 4 (PPO2CPN):* To the PPO $\mathcal{O} = (T, V, S, P)$ corresponds the marked CPN $\Gamma(\mathcal{O}) =$

$((\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathcal{C}, \mathcal{E}, \mathcal{G}), m)$, where

$$
\begin{array}{l}
\mathcal{T} = T, \\
\mathcal{P} = \{t^p \mid t \in T\} \cup \mathrm{dom}(P), \\
\forall t \in T : \mathcal{C}(t^p) = \prod \ell(V(t)), \\
\forall (t,u) \in \mathrm{dom}(P) : \mathcal{C}(t,u) = \mathcal{C}(t^p) \times \mathcal{C}(u^p), \\
\mathcal{A} = \{(t^p, t) \mid t \in T\} \cup \{(t,(t,u)) \mid (t,u) \in \mathcal{P}\} \\
\qquad \cup \{((t,u),u) \mid (t,u) \in \mathcal{P}\}, \\
\forall (t^p, t) \in \mathcal{A} : \mathcal{E}(t^p, t) = \ell(V(t)), \\
\forall (t,(t,u)) \in \mathcal{A} : \mathcal{E}(t,(t,u)) = \\
\qquad [(\ell(V(t)), \ell(V'(u))) \mid P(t,u) \wedge S'(u)], \\
\forall ((t,u),u) \in \mathcal{A} : \mathcal{E}((t,u),u) = \\
\qquad [(\ell(V'(t)), \ell(V(u))) \mid P'(t,u) \wedge S'(t)], \\
\mathcal{G} = \{(t, \mathsf{true}) \mid t \in T\}, \\
\forall t \in T : m(t^p) = [\ell(V(t)) \mid S(t)], \text{ empty for other places .}
\end{array}
$$

The example PPO in Fig. 1 becomes as shown below.

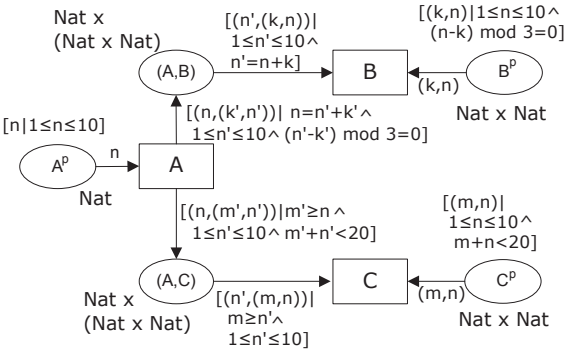


Figure 4. CPN corresponding to the PPO from Fig. 1

*Remark 1:* Note that the above translation assumes that the bags $[\ell(V(t)) \mid S(t)]$ are finite. The implementation in CPN Tools requires to syntactically derive values $\mathsf{frst}_t$, $\mathsf{last}_t$ in $\ell(V(t))$ and a function $\mathsf{nxt}_t : \ell(V(t)) \to \ell(V(t))$ for every task $t$ satisfying $\mathsf{last}_t = \mathsf{nxt}_t^N(\mathsf{frst}_t)$ such that $\forall x \in [\ell(V(t)) \mid S(t)] : (\exists n \leq N : x = \mathsf{nxt}_t^n(\mathsf{frst}_t))$. This allows to recursively generate the bag $[\ell(V(t)) \mid S(t)]$ by starting with $\mathsf{frst}_t$, testing whether $S$ holds and applying $\mathsf{nxt}_t$ until $\mathsf{last}_t$ is reached. By adding special "generator" transitions, we can even omit the $\mathsf{last}_t$ value, allowing for a certain class of infinite PPOs.

### A. Correctness of the translation

We now establish a bisimilarity relation between a PPO $\mathcal{O}$ and its corresponding CPN $\Gamma(\mathcal{O})$, guaranteeing that the two models always behave in the same way. We first define a function $\gamma$, mapping states of $\mathcal{O}$ to their corresponding markings in $\Gamma(\mathcal{O})$.

*Definition 5 (Corresponding marking):* Given a PPO $\mathcal{O} = (T, V, S, P)$, let $\Gamma(\mathcal{O}) = ((\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathcal{C}, \mathcal{E}, \mathcal{G}), m)$ be its corresponding marked CPN. Let $\phi_\sigma(t) = \sigma(\ell(V(t)))$ for $t \in T$ and $\sigma \in \Sigma_t$. The PPO-state $\theta \subseteq (T \times \Sigma)$ corresponds to the CPN marking $\gamma_\theta$ defined by

$$
\begin{array}{l}
\forall t \in T : \gamma_\theta(t^p) = [\phi_\sigma(t) \mid \sigma \in \Sigma_t \wedge (t,\sigma) \notin \theta] \\
\forall (u,t) \in \mathrm{dom}(P) : \gamma_\theta(u,t) = \\
\qquad [(\phi_\sigma(u), \phi_\rho(t)) \mid \sigma \in \Sigma_u \wedge \rho \in \Sigma_t \wedge \\
\qquad\qquad (u,\sigma) \notin \theta \wedge (t,\rho) \in \theta].
\end{array}
$$

Note that $\gamma_\emptyset$ equals the initial marking of the corresponding CPN. The next theorem shows that $\gamma$, seen as a relation between PPO states to markings, is a bisimulation between the states of the PPO and their corresponding CPN markings. Reachable CPN markings are $\gamma$-related to PPO states.

*Theorem 1:* Let $\mathcal{O} = (T, V, S, P)$ be a PPO, $\theta, \theta' \subseteq T$ PPO states, and $t \in T, \sigma \in \Sigma_t$ such that $\theta \xrightarrow{(t,\sigma)} \theta'$. Then $\gamma_\theta \xrightarrow{(t,\sigma)} \gamma_{\theta'}$ in $\Gamma(\mathcal{O})$. Conversely, if $\theta \subseteq T$ and $\gamma_\theta \xrightarrow{(t,\sigma)} m'$ in $\Gamma(\mathcal{O})$, then there is a $\theta'$ such that $m' = \gamma_{\theta'}$ and $\theta \xrightarrow{(t,\sigma)} \theta'$ in $\mathcal{O}$.

*Proof 1:* Let $((\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathcal{C}, \mathcal{E}, \mathcal{G}), m)$ be the marked CPN $\Gamma(\mathcal{O})$. Let $\theta, \theta' \subseteq T$ be PPO states and $t \in T, \sigma \in \Sigma_t$ such that $\theta \xrightarrow{(t,\sigma)} \theta'$. From the PPO definition, it follows that $(t,\sigma) \notin \theta$, $\sigma(S(t)) = \mathsf{true}$ and $\theta' = \theta \cup \{(t,\sigma)\}$. Moreover, for every $u$ such that $(u,t) \in \mathrm{dom}(P)$ and every valuation $\rho \in \Sigma_u$ such that $(\sigma' \cup \rho)(P(u,t)) = \mathsf{true}$, we must have $(u,\rho) \in \theta$. By the definition of $\gamma_\theta$, we infer $\phi_\sigma(t) \in \gamma_\theta(t^p)$. In addition, for every $u, t$ such that $(u,t) \in \mathrm{dom}(P)$ and every $\rho \in \Sigma_u$ such that $(\sigma' \cup \rho)(P(u,t)) = \mathsf{true}$, we have $(\rho,\sigma) \in \gamma_\theta(u,t)$, since $(t,\sigma) \notin \theta$ and $(u,\rho) \in \theta$. Thus, there exists an $m$ such that $\gamma_\theta \xrightarrow{(t,\sigma)} m$. By the firing rule, $m(t^p) = \gamma_\theta(t^p) - [\phi_\sigma(t)]$, and for each $u$ such that $(u,t) \in \mathcal{P}$ we have $m(u,t) = \gamma_\theta(u,t) - [(\phi_\rho(u), \phi_\sigma(t)) \mid \rho \in \Sigma_u \wedge (\sigma' \cup \rho)(P(u,t)) = \mathsf{true}]$. This entails $m = \gamma_{\theta'}$, so $\gamma_\theta \xrightarrow{(t,\sigma)} \gamma_{\theta'}$.

Conversely, suppose $m \xrightarrow{(t,\sigma)} m'$, with $m = \gamma_\theta$, so $\sigma(S(t)) = \mathsf{true}$. Thus, $(t,\sigma) \notin \theta$. We set $\theta' = \theta \cup \{(t,\sigma)\}$. By the CPN firing rule, $m(t^p) = m'(t^p) + [\phi_\sigma(t)]$ and for each task $u$ such that $(u,t) \in \mathcal{P}$ we have $m(u,t) = m'(u,t) + [(\phi_\rho(u), \phi_\sigma(t)) \mid \rho \in \Sigma_u \wedge (\sigma' \cup \rho)(P(u,t)) = \mathsf{true}]$. This entails that $(\phi_\rho(u), \phi_\sigma(t)) \in \gamma_\theta(u,t)$ for each $u \in T, \rho \in \Sigma_u$ satisfying $(\sigma' \cup \rho)(P(u,t)) = \mathsf{true}$ and hence $m' = \gamma_{\theta'}$. Clearly, $\theta \xrightarrow{(t,\sigma)} \theta'$.

### B. Special cases and optimizations

The generic PPO to CPN translation sketched above can be reduced if certain precedence relations are represented as functions. The first model in Fig. 5 is a PPO in which the precedence expression has the special form $P(t,u) =$ "$(x_1, \ldots x_n) = f(y'_1, \ldots y'_m)$", written in vector notation "$\bar{x} = f(\bar{y}')$". The second model in the figure shows the generic CPN translation. The third model shows a possible reduction. The type of the place $(t,u)$ and the adjacent arc expressions are simplified. The place $u^p$ is now redundant and is thus removed. Similar reductions are possible for conditions of the form "$\bar{y}' = g(\bar{x})$".

Because of the shape of condition $P(t,u)$ and the fact that place $(t,u)$ is empty in the initial marking, any reachable marking $m$ of the original net satisfies $m(t,u) =$

$\{(f(\bar{y}), \bar{y}) \mid \bar{y} \in B\}$ for some $B \subseteq T_u$. In the reduced net, marking $m$ corresponds to a marking $\mu$ satisfying $\mu(t,u) = B + [\bar{y} \in m(u^p) \mid \neg\exists \bar{x} \in m(t^p) : \bar{x} = f(\bar{y})]$. For all places $p \notin \{(t,u), u^p\}$ we have $\mu(p) = m(p)$. This induces a bisimilarity between the two marked CPNs.

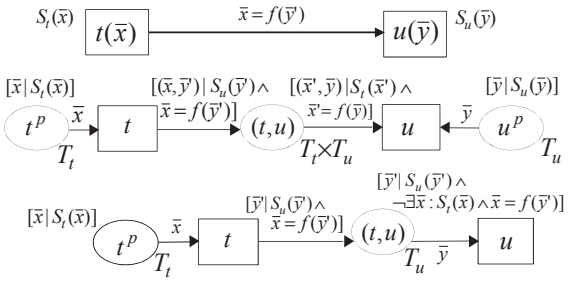


Figure 5. A PPO, its generic CPN translation, and reduction

Further simplification is possible. If the function $f$ is injective (as is the case with conditions such as "$s' = s + 1$") and the implication $S_t(\bar{x}) \Rightarrow S_u(f^{-1}(\bar{x}))$ holds, the expression $[\bar{y}' \mid S_u(\bar{y}') \wedge \bar{x} = f(\bar{y}')]$ on the $(t,u)$ in-arc is reduced to $f^{-1}(x)$. Moreover, if the function $f$ is bijective (as is the case with conditions such as "$s' = s$"), the initialization expression of the $(t,u)$ place is empty. For conditions of the form "$s' = s + 1$", the initialization expression becomes the first $s$, i.e. bag $[1]$. Note that all these special conditions, although semantical in nature, can in many practical cases be syntactically derived from expressions.

### C. Printing example implementation

The printer example from Fig. 2 has been converted into a CPN Tools implementation and the result is shown in Fig. 6. Identifiers have been shortened for clarity. For the CPN ML language, bag expressions are replaced by calls of recursively defined functions using parameter enumeration. For example, the initialization expression $[p \mid 1 \leq p \leq P]$ has become `Allp:=AllpH(1)`, with `AllpH(p)=if p>P then [] else [p]++AllpH(p+1)`. The expression $[s \mid p = \mathsf{hlv}(s)]$ on the `di_co` out-arc has become `hlvs(p)=shlv(1,p)`, with `shlv(s,p) = if s>S then [] else shlv(s+1,p) ++ (if hlv(s) = p then [s] else [])`. Types `NAT` and `NAT2` are defined as `INT` and `product NAT*NAT` respectively.

Although it is already beneficial to be able to study the dynamics of this use-case by executing the CPN model step by step, the main purpose of the model is to serve as a test-case in the process of evaluating different printer designs. This is done by connecting the model with a special CPN template, containing resource specifications and handling procedures, and performing (stochastic) simulations. We refer to [1] for a thorough example.

## VI. Conclusions

We define Parameterized Partial Orders (PPOs), a powerful data-extension of classical partial orders. This model

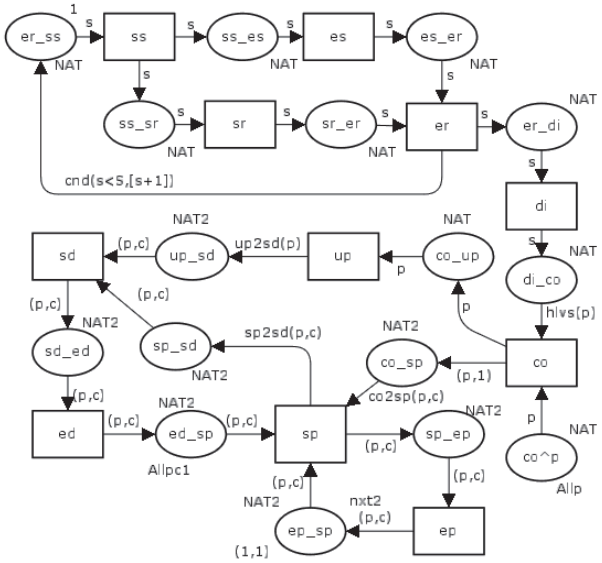


Figure 6. Generated CPN model of the printer use case

serves as an intermediate representation of embedded-system use-cases in the Octopus toolset. We define a translation from PPOs to Coloured Petri nets (CPNs), a target model to enable performance assessment of system designs. Meeting the declarative-to-operational challenge shows that PPOs fulfil the requirements of an intermediate representation. We finally present a practical case study illustrating the described concepts. In future work, we plan to formalize our CPN implementation of the complete Y-chart.

The Octopus toolset is complemented by an UPPAAL converter [8], allowing for schedulability analysis and verification. The two types of analysis can be effectively combined and support engineers in their design decisions.

## References


[1] T. Basten et al., "Model-Driven Design Space Exploration for Embedded Systems: The Octopus Toolset," in *Proc. ISoLA 2010*, ser. LNCS, vol. 6415. Springer, 2010, pp. 90–105.

[2] "Model - Driven Design - Space Exploration : The Octopus Toolset," http://dse.esi.nl/.

[3] "The Octopus project," http://www.esi.nl/projects/octopus.

[4] B. Kienhuis et al., "A Methodology to Design Programmable Embedded Systems - The Y-Chart Approach," in *Embedded Processor Design Challenges*, ser. LNCS, vol. 2268. Springer, 2002, pp. 18–37.

[5] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Comm. ACM*, vol. 21, pp. 558–565, 1978.

[6] K. Jensen and L. Kristensen, *Coloured Petri Nets*. Springer, 2009.

[7] "Uppaal Web Site," 2010, www.uppaal.org, Web Help.

[8] F. Houben, "Design Space Exploration with Generated Timed Automata," Master Thesis Radboud University Nijmegen 2010.