A Calculus of Petri Net Components

N. Sidorova, M. Voorhoeve [1], J.C.S.P. van der Woude
Eindhoven University of Technology

Abstract

A framework is developed for modeling concurrent systems. The modeled systems are "open" in the sense that they can interact in prescribed ways with an unspecified environment. Systems are represented both as labeled Petri nets (allowing validation) and algebraic terms (allowing analysis).

1 Introduction

The engineering of systems is often performed by combining *requirements* and *models*. Requirements can be formulated in both natural and formal languages. It is almost impossible to describe a system by requirements alone. Their soundness can be validated by end users, but not their completeness: most likely systems can be constructed satisfying the requirements that are nonetheless unacceptable to the end user. This is the reason why requirements are complemented by models.

The systems we consider in this paper consist of many cooperating, yet independent asynchronous components. Petri Nets are well suited for modeling such systems. Advantages are its graphical nature and the decomposition of both states and events into local places and transitions. These properties make Petri nets an excellent tool for the validation of models by non-technical end users. Many state-of-the-art modeling techniques use Petri nets (or some derivative) for describing concurrent activities.

It stands to reason that Petri nets have been combined with hierarchy by isolating subnets and combining them to form large supernets. Examples are CPN [9] and ExSpect [8]. The common way of defining the semantics for these hierarchical approaches is to "flatten" the hierarchy and present the semantics of the complete system. So subnets obtain a semantics only in the context they were used in, which means that - in spite of efforts to make them reusable - they cannot be regarded as true components. True Petri net components must have a compositional semantics of their own. Compositionality means that components with the same semantics (i.e. mapped to the same mathematical object) must behave the same in any context.

In this paper we develop a framework for Petri net components, inspired by the theory behind process algebra [1]. Components are regarded as algebraic terms specifying subcomponents, their connection and the external interface. These terms are "open"; the interface fixes their possible uses by any environment. This results in a calculus with operators for connecting components, like fusion, relabeling and hiding of places and transitions, very similar to [12]. Each algebraic term can be normalized, resulting in a term containing the merge of basic components (places and transitions) that are connected by arc addition and then relabeled. These normalized

[1]email: wsinmarc@win.tue.nl

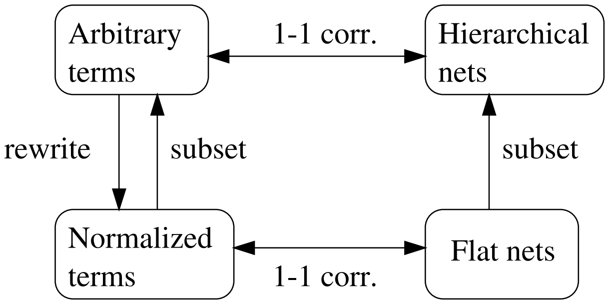

Figure 1: Correspondences between nets and terms

terms can be represented as "flat" Petri nets with partially labeled places and transitions (compare Figure 1). The state/event duality typical of Petri nets has been preserved: one can choose to label only places (reasoning only about states and their successions) or transitions (reasoning about possible and actual events) or combine the two.

The semantics of our component terms is indeed compositional: the (strong) bisimilarity equivalence is a congruence w.r.t. our operators. We can use it to speed up verifications of closed hierarchical nets by exploiting the hierarchy. However, the most important contribution is the possibility to define "open" subnets (components) that can be used and reused in arbitrary contexts.

This paper is a development from [12] (the calculus) and [2] (the semantics: branching instead of linear time). Both predecessor papers address P/T nets only, whereas the present paper applies to colored nets as well, thus providing a foundation for full-fledged component specification languages.

We have structured the paper as follows. In section 2, after some preliminary notations, we give some general definitions and results about transition systems with events having a baglike structure (i.e. they can occur simultaneously). In section 3, the events are pinned down further to consumption-firing-production triples typical of Petri nets. We define atoms (unconnected marked places and transitions) and operators for connecting them. In section 4 this calculus is specialized still further to cover "standard" colored nets. An example script is given in section 5 to illustrate its use as a specification and design language, after which some concluding remarks are given in section 6.

Acknowledgements: We thank our colleagues Tim Willemse, Twan Basten and Sjouke Mauw for their support and suggestions, as well as the anonymous referees.

2 Basic Notions

Let A, B be sets. A binary relation between A and B is a subset of $A \times B$. For relations R, S, R^{-1} denotes the converse of R and $R \mathbin{.} S$ denotes the composition of R and S. Subclasses of binary relations are the sets of total ($A \rightarrow B$) and partial ($A \nrightarrow B$) functions. Function application of f to a is denoted as $f.a$. We omit the dot operator if this does not cause confusion, writing e.g.

$f(a, b)$ instead of $f.(a, b)$. If r is a relation and X a set, then $r[X] = \{a \mid \exists x \in X : x\ r\ a\}$.

The set of bags with base A is denoted as $\mathbb{B}.A$. For $\alpha \in \mathbb{B}.A$, $a \in A$, $w_\alpha a$ denotes the weight of a in α. The carrier $C.\alpha$ of $\alpha \in \mathbb{B}.A$ is the set $\{a \mid w_\alpha a > 0\}$. We identify bags $\alpha \in \mathbb{B}.A$ and $\beta \in \mathbb{B}.B$ when $C.\alpha = C.\beta$ and $\forall c \in C.\alpha : w_\alpha c = w_\beta c$. If $\alpha \in \mathbb{B}.A$, $\beta \in \mathbb{B}.B$, then $\alpha + \beta$ is the element of $\mathbb{B}(A \cup B)$ satisfying $w_{\alpha+\beta} x = w_\alpha x + w_\beta x$. The empty bag is denoted by 0 (its carrier is the empty set). If no confusion can arise, we can interpret a set A as a bag with carrier A and weights 1 throughout. Note that $(\mathbb{B}.A, +, 0)$ is a commutative monoid.

If $r \in A \nrightarrow B$, then $\bar{r} \in \mathbb{B}.A \to \mathbb{B}.B$ satisfies $w_{\bar{r}.\alpha} b = \Sigma(w_\alpha a \mid r.a = b)$. So if r is the empty function, the range of $\bar{r}$ is $\{0\}$.

We shall define transition systems and the bisimilarity equivalence relation on them. Bisimilarity abstracts as much as possible from states; two states are equivalent iff they allow the same actions leading to equivalent states. We presuppose a universe of *event labels* $\mathcal{E}$.

Definition 1 *A* transition system *is a pair (S, E), where S is a state space and $E \in \mathcal{E} \to \mathbb{P}(S \times S)$ a function assigning state relations to events. The relations $E.e$ are denoted by $\xrightarrow{e}$. If $C \xrightarrow{e} C'$, there exists an event with label e leading from C to C'.*

Given a transition system (S, E), a relation $Q \subseteq (S \times S)$ is a simulation *iff for every $e \in \mathcal{E}$ we have $Q^{-1} \centerdot \xrightarrow{e} \subseteq \xrightarrow{e} \centerdot Q^{-1}$. A simulation Q is a* bisimulation *iff Q^{-1} is a simulation too. States C, D are called* bisimilar *iff there exists a bisimulation Q such that $C\ Q\ D$.*

A component *is an equivalence class of bisimilar states in a transition system.*

Note that we use a single transition system S in the definition, which is enough for our purposes. A definition with several transition systems is not hard, but somewhat more cumbersome. A simulation is a relation Q that satisfies the transfer property: if $C\ Q\ D$ and $C \xrightarrow{e} C'$, then there exists a D' such that $D \xrightarrow{e} D'$ and $C'\ Q\ D'$. A weaker equivalence relation is *branching bisimilarity* [6]. This equivalence relation is most appropriate for comparing implementations to their specification. The bisimilarity notion above is often called "strong" bisimilarity in contrast.

The state spaces of our transition systems will consist of algebraic terms describing the events that can occur to its states. Those terms are given via SOS production rules (structured operational semantics [11]). These rules define the events that can occur in a given state; it is implicitly assumed that no event can occur in a given state unless there is a production rule defining it (no junk property). The syntax is $\dfrac{p, q, \ldots}{r}$, where $p, q, \ldots$ are the premisses and r the conclusion.

We suppose that the set of event labels $\mathcal{E}$ is a commutative monoid with addition operator + and neutral element 0. The interpretation of $e + f$ is the simultaneous occurrence of events e and f. The neutral element 0 indicates the absence of any visible event. Every component C will satisfy $C \xrightarrow{0} C$. Note that it is possible that $C \xrightarrow{0} C'$ with $C \neq C'$: the state may change without any event being visible.

The binary free merge operator $C \mid\mid D$ allows the combination of components; there exist various transformation operators Θ_R, with R a binary or ternary relation between events. In the sections to come, we will specialize these transformation operators in various ways.

We assume the existence of a set A of atoms with a set of relations $\xrightarrow{e}_A$ between them. Our state space S is the smallest set satisfying

1. $A \subseteq S$
2. $C, D \in S \Rightarrow C \,||\, D \in S$
3. $(R \in \mathcal{E}^2 \cup \mathcal{E}^3 \wedge C \in S) \Rightarrow \Theta_R.C \in S.$

The relations $\xrightarrow{e}$ are the smallest subsets of S^2 containing the $\xrightarrow{e}_A$ or produced from the SOS rules given below.

$$\frac{C \xrightarrow{e} C',\, D \xrightarrow{f} D'}{C \,||\, D \xrightarrow{e+f} C' \,||\, D'} \qquad \frac{C \xrightarrow{e} C',\, e\ R\ f}{\Theta_R.C \xrightarrow{f} \Theta_R.C'} \qquad \frac{C \xrightarrow{d} \centerdot \xrightarrow{e} C',\, (d, e, f) \in R}{\Theta_R.C \xrightarrow{f} \Theta_R.C'}$$

From the structure of these rules, it can be deduced that the merge and transformation operators are congruences w.r.t. both strong and branching bisimilarity (c.f. [5]).

Theorem 1 *Let B, C, D be components and $R, T \subseteq \mathcal{E}^2$. Then*
$C \,||\, D = D \,||\, C, \quad (B \,||\, C) \,||\, D = B \,||\, (C \,||\, D), \quad \Theta_R \centerdot \Theta_T = \Theta_{T.R}$

Proof: We shall derive the first of these equations in order to illustrate the type of reasoning used here. We construct the relation $Q = \{(C \,||\, D, D \,||\, C) \mid C, D \in S\}$ between terms. It is sufficient to prove that Q is a bisimulation. We must prove two transfer properties; the first is that $X\ Q\ Y$ and $X \xrightarrow{e} X'$, there exists an Y' such that $Y \xrightarrow{e} Y'$ and $X\ Q\ Y'$. Since $X\ Q\ Y$, X must be of the form $C \,||\, D$ and Y of the form $D \,||\, C$. We have $C \,||\, D \xrightarrow{e} X'$ and as the only SOS rule matching the lhs is the first, X' must have the form $C' \,||\, D'$ for some C', D' satisfying $C \xrightarrow{f} C'$, $D \xrightarrow{g} D'$ and $e = f + g$. It is now easy to find a Y', namely $D' \,||\, C'$ satisfying the transfer property. Since Q is symmetric, the other transfer property is identical to the first, so Q is indeed a bisimulation.
The other two equations are derived similarly. □

3 Global Petri Nets

As a first step towards formalizing Petri net components, we decompose events into consumption, firing and production. Unlike colored or P/T nets, firings are not associated to distinct transitions nor consumptions/productions to distinct places, whence the term "global".

In Figure 2, the main idea is illustrated for defining the semantics of net components, namely the use of consumption/firing/production triples. Three components C, D, E are shown in the figure. The component C can be transformed into D by external production of a token in place p. By externally consuming the same token, D becomes C. By a firing step, D becomes E.

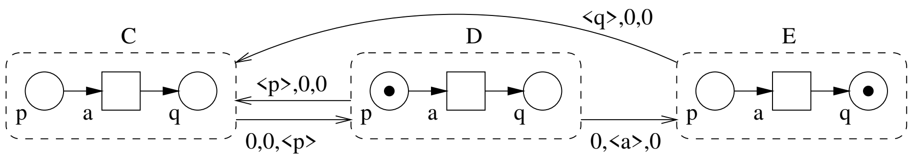

Figure 2: Illustration of events

Internal consumptions and productions (i.e. associated to a firing step) go with the firing and are not separately indicated.

This way of associating a transition system to nets allows both state-based and action-based reasoning. We can discover that there is one and only one token in p-labeled places of D, since one p-token (and not more) can be externally consumed from it. We can also discover that C contains an a-labeled transition that is not enabled but becomes so by externally adding a p-token, and which internally consumes a p token and produces a q token, since the p-token that was there in D has gone and a q-token has appeared after firing.

So we have indeed disguised state-based properties (the presence and absence of tokens) as events (possible and impossible consumptions). Of course, we can have several external consumptions, firings and external productions concurrently, if there are enough tokens to allow the consumptions and firings. Tokens that are produced (either directly or by the firings) have to wait for the next step to be used.

We will indicate how this idea has been transformed into a calculus. Our atoms are unconnected marked places, that allow production and consumption only, and unconnected transitions that allow firing only. The atoms can be *merged* and by *arc addition* they can become connected, associating internal productions and/or consumptions to firings. By *relabeling*, we can then redefine the component's interface. This relabeling allows e.g. to hide tokens (disabling external consumptions and productions), hide firings (relabeling them to 0) and introduce nondeterminism (giving different tokens/firings the same label).

The above atoms and operators suffice to model terms that have a direct representation as partially labeled Petri nets. Their application to arbitrary components causes a notion of hierarchy. Finally, the operators are supplemented with a *place fusion* operator, extending the options for connecting components. We can rewrite any term to a normal form that has a flat net representation.

We shall now formalize the notions above. We first set $\mathcal{E} = \mathcal{O} \times \mathcal{A} \times \mathcal{O}$, where $\mathcal{O}, \mathcal{A}$ are commutative monoids (place label, resp. action bags). We assume the set of labels to be infinite, so "fresh" labels are always available. We set $(a, b, c) + (d, e, f) = (a + d, b + e, c + f)$ so $\mathcal{E}$ is a commutative monoid.

We call a relation r between commutative monoids *right linear* iff (1) $0\ r\ 0$, (2) $(a\ r\ b \wedge c\ r\ d) \Rightarrow (a + c)\ r\ (b + d)$ and (3) $a\ r\ (b + c) \Rightarrow \exists a_1, a_2 : a = a_1 + a_2 \wedge a_1\ r\ b \wedge a_2\ r\ c$. It is *right normal* if $0\ r\ x \Rightarrow x = 0$ and *left normal* if r^{-1} is right normal.

From now on, let $A \subseteq \mathcal{A}$, $I \in \mathcal{O}$, $r \in \{s \cup t \mid s \subseteq \mathcal{O}^2 \wedge t \subseteq \mathcal{A}^2\}$ right linear and right normal, $v, w \subseteq (\mathcal{O} \times \mathcal{A})$ both right linear and left normal, $U \subseteq \mathcal{O}$ with $0 \in U$.

Definition 2 *The components P_I (place), T_A (transition) and the operators λ_r (renaming), $\gamma_{v,w}$ (arc addition) and ϕ_U (place fusion) are defined by the SOS rules below.*

$$
1 \quad \frac{x \in A}{T_A \stackrel{0,x,0}{\longrightarrow} T_A} \qquad 2 \quad \frac{c \leq I}{P_I \stackrel{c,0,p}{\longrightarrow} P_{I-c+p}} \qquad 3 \quad \frac{C \stackrel{c,s,p}{\longrightarrow} C',\, c\ r\ d,\, p\ r\ q,\, s\ r\ t}{\lambda_r.C \stackrel{d,t,q}{\longrightarrow} \lambda_r.C'}
$$

$$
4 \quad \frac{C \stackrel{d+c,s,p+q}{\longrightarrow} C',\, d\ v\ s,\, q\ w\ s}{\gamma_{v,w}.C \stackrel{c,s,p}{\longrightarrow} \gamma_{v,w}.C'} \qquad 5 \quad \frac{C\ (\stackrel{x,0,x}{\longrightarrow} \centerdot \stackrel{c,s,p}{\longrightarrow})\ C',\, x \in U}{\phi_U.C \stackrel{c,s,p}{\longrightarrow} \phi_U.C'}
$$

, Note the non-atom operators being special cases of Θ_R, e.g. $\gamma_{v,w} = \Theta_R$ with $(c, s, p)\ R\ (c', s', p')$ iff $s = s' \wedge (c - c')\ v\ s \wedge (p - p')\ w\ s$. The operator ϕ_U is the one with a ternary R.
We will discuss the notions behind this formal definition.

1. Unconnected transitions (T_A) can execute any step from its parameter set A without any state change.

2. Unconnected marked places (P_I) allow an arbitrary consumption of tokens c contained in the marking I ($c \leq I$) and an arbitrary production p, resulting in a new state $I - c + p$.

3. The renaming operator λ_r allows to rename steps, consumptions and productions by relation r. In most cases, the relation r will be a partial function. Hiding of steps is done by renaming them to 0. The right linearity of q ensures that hidden steps cannot become visible again. To allow the hiding of tokens, we can use a partial function r to shield tokens outside dom.r from external production or consumption.

4. The arc addition operator, when used with functions instead of relations, states that an event $(0, s, 0)$ (i.e. a step) between $\gamma_{v,w}.C$ and $\gamma_{v,w}.C'$ will be possible iff the event $(v.s, s, w.s)$ existed between C, C' before the arc addition. A special choice of the v, w relations yields the blocking operator ξ_E for certain $E \subseteq \mathcal{A}$ with the production rule $\dfrac{C \stackrel{c,s,p}{\longrightarrow} C',\, s \in E}{\xi_E.C \stackrel{c,s,p}{\longrightarrow} \xi_E.C'}$.

5. The fusion rule states that certain states that *look* the same in fact *become* the same in ϕ_U. So if an event (c, s, p) is possible from a state C'', the same event is possible from C, provided C'' can be reached from C by a token switch: consuming and producing tokens that both are labeled x (but may differ).

Component terms are recursively defined from the places, transitions and merge and transformation operators.

Definition 3 *The set $\mathcal{C}$ of* component terms *is the smallest set satisfying:*

1. *$T_A, P_I \in \mathcal{C}$,*

2. *if $C, D \in \mathcal{C}$, then $C \,||\, D \in \mathcal{C}$,*

3. *if $C \in \mathcal{C}$, then $\lambda_r.C, \xi_E.C, \gamma_{v,w}.C, \phi_U.C \in \mathcal{C}$.*

We shall now derive rules by which the terms in $\mathcal{C}$ can be rewritten to a flat normal form. The place fusion and blocking operators can be eliminated altogether and the other operators can be given a certain order. We start with some basic equations, stating that the application order of most operators can be reversed.

We define the sum of sets and relations $A, B \subseteq \mathcal{A}$ by $A + B = \{a + b \mid a \in A, b \in B\}$. If r, s are both right linear, then so is $r + s$.

Theorem 2 *The following equations hold.*

$$\begin{array}{llll} \lambda_r.T_A = T_{r[A]} & \gamma_{v,w}.T_A = T_{A\cap(v^{-1}\cap w^{-1})[0]} & \phi_U.T_A = T_A & T_A \,||\, T_{A'} = T_{A+A'} \\ \lambda_r.P_I = P_{r[I]} & \gamma_{v,w}.P_I = P_I & \phi_U.P_I = P_I & P_I \,||\, P_J = P_{I+J} \end{array}$$

$$\lambda_r \bullet \lambda_x = \lambda_{r\bullet x} \quad \gamma_{v,w} \bullet \lambda_r = \lambda_r \bullet \gamma_{r\bullet v\bullet r^{-1}, r\bullet w\bullet r^{-1}} \quad \gamma_{v,w} \bullet \gamma_{x,y} = \gamma_{v+x,w+y}$$

$$\phi_U \bullet \lambda_r = \lambda_r \bullet \phi_{r^{-1}[U]} \quad \phi_U \bullet \gamma_{v,w} = \gamma_{v,w} \bullet \phi_U$$

Proof: We construct the obvious bisimulation between terms in each case. □

We can eliminate bijective renaming.

Theorem 3 *If r is a bijection, then λ_r is a bijection on $\mathcal{C}$ with inverse $\lambda_{r^{-1}}$. We also have the equation $\lambda_r \bullet \gamma_{v,w} = \gamma_{r\bullet v, r\bullet w}$ in this case. For all C, D, we have $\lambda_r(C \,||\, D) = \lambda_r.C \,||\, \lambda_r.D$.*

Proof: By Theorem 2, $\lambda_r \bullet \lambda_{r^{-1}} = \lambda_{r\bullet r^{-1}}$, which is the identity. Again, bisimulations are constructed for each equation. □

We can now prove the flat net theorem: each hierarchical net ($\mathcal{C}$ term) can be represented as the renaming of connected places and transitions.

Theorem 4 *Each term $C \in \mathcal{C}$ can be rewritten to the normal form $(\lambda_r \bullet \gamma_{v,w}).(T_A \,||\, P_I)$.*

Proof: We use structure induction to terms. Use the equations in Theorem 2 as rewrite rules from left to right, proving the theorem for all terms of the form $(f \bullet g).C$, with f, g transformations. We are left with terms $C \,||\, D$. By the induction hypothesis, we may assume that C, D are in normal form, so let $C = (\lambda_u \bullet \gamma_{v,w}).(T_A \,||\, P_I)$. With $f = \lambda_u$, $E = \gamma_{v,w}.(T_A \,||\, P_I)$, we can find a fresh set of labels and a bijective renaming r such that $\lambda_r \bullet f = f \bullet \lambda_{r^{-1}} = f$. We then have $C \,||\, D = f.E \,||\, D = f.E \,||\, \lambda_r \bullet \lambda_{r^{-1}}.D = (\lambda_r \bullet f).(E \,||\, \lambda_{r^{-1}}.D)$. Since r was injective, we can eliminate the $\lambda_{r^{-1}}$ by Theorem 3, obtaining the term $(\lambda_r \bullet f).(E \,||\, D')$, with D' in normal form. We repeat the same trick with $f = \gamma_{v,w}$, $E = T_A \,||\, P_I$ to obtain $(\lambda_{r'} \bullet \gamma_{v,w})E \,||\, D''$. Repeating this for D'', we obtain a merge of places and transitions, that can be rewritten to the merge of one place and one transition by Theorem 2. □

4 Local Petri nets

In this section we further specialize the operators of the previous section to arrive at more or less "standard" colored nets. The firings that can occur are localized into distinct transitions and the tokens that are consumed and produced are localized into distinct places.

We attach a label to every place; tokens possess the label of their place and a value in addition. A marked place is parametrized by a bag of such tokens (with the same label). We attach a function of labels to values (a "record") to every transition firing; a transition is parametrized with a set of such records (with the same domain) (a "table").

The selection of operators is a compromise between power and clarity, with CPN in mind. It is not clear whether a closer approximation of CPN is possible or worthwhile. Arc addition is split into production and consumption. We choose one label-value pair of a firing record to go with a label-value pair of a token.

We define fusion based on labels only: all tokens from places labeled with a fusion label are dumped in the same place (as it were). Relabeling is split into transition and place relabeling and does not change values. Blocking is split into encapsulation: the blocking of any step containing a forbidden label-value pair and synchronization: the blocking of any step that does not combine the label-value pairs to be synchronized. Hiding is split into place hiding and transition hiding and is based on labels.

The formal definition of tha atoms and operators of our global net calculus involves rather awkward notation. To get an impression of them, the reader is invited to compare the example script in the next section. We set the monoids $\mathcal{O}$ and $\mathcal{A}$ to be bags of label-value pairs: $\mathbb{B}.(\mathcal{L} \times \mathcal{V})$, where $\mathcal{L}$ is an infinite set of labels and $\mathcal{V}$ a set of values. The *closure* of a set F of functions is set of bags that can be written alpha as a finite sum σf_i with the $f_i \in F$ are interpreted as bags with weights 1.

Definition 4 *We define*

1. *For $A \subseteq (L \to \mathcal{V})$, with $L \subseteq \mathcal{L}$ finite, we set $T[A] = T_B$, with B the closure of A.*
2. *For $\ell \in \mathcal{L}, \alpha \in \mathbb{B}.\mathcal{V}$, we set $P[\ell, \alpha] = P_I$, where I is the bag with carrier $\ell \times C.\alpha$ such that $w_I(\ell, v) = w_\alpha v$ for $v \in \mathcal{V}$.*
3. *For $f \in (\mathcal{L} \times \mathcal{L}) \nrightarrow \mathbb{P}(\mathcal{V} \times \mathcal{V})$, we set $\chi[f] = \gamma_{v,0}$ and $\pi[f] = \gamma_{0,v}$, with $v = \{((\ell, x), (m, y)) \mid x \ f(\ell, m) \ y\}$ (consumption/production). The function 0 has the empty bag as range.*
4. *For a relation $g \subseteq \mathcal{L} \times \mathcal{V}$, we define $\partial[g] = \xi_B$, with $B = \mathbb{B}.(\mathcal{L} \times \mathcal{V} \setminus g)$ (encapsulation) and $\sigma[g] = \xi_B$, with B the closure of $\{h \subseteq g \mid \mathrm{dom}.h = \mathrm{dom}.g \wedge h \text{ is functional}\}$ (synchronization).*
5. *For $L \subseteq \mathcal{L}$ finite, we define $\phi[L] = \phi_U$, with $U = L \times \mathcal{V}$.*
6. *For $h \in \mathcal{L} \nrightarrow \mathcal{L}$, let $H = \mathrm{dom}.h$. We define $\theta[h] = \lambda_{1_{\mathcal{L}\times\mathcal{V}},\bar{q}}$, with 1_A the identity function on A and $q = (h \cup 1_{\mathcal{L}\setminus H}) \times 1_{\mathcal{V}}$ (transition relabeling, which is total!) and $\eta[h] = \lambda_{\bar{r},1}$, with $r = h \times 1_{\mathcal{V}}$ (place relabeling, which is partial).*
7. *For $H \subseteq \mathcal{L}$, we define $\tau[H] = \lambda_{1,\bar{q}}$, with $q = id_{\neg H} \times \mathcal{V}$ (transition hiding) and $\upsilon[H] = \lambda_{\bar{r},1}$, with $r = id_{\neg H} \times \mathcal{V}$ (place hiding).*

We define labeled components, terms defining hierarchical colored Petri nets.

Definition 5 *A labeled component is either a term $T[A]$ or $P[\ell, \alpha]$ or $C \mid\mid D$ with C, D labeled components or a $F.C$ with $F \in \{\chi[f], \pi[f], \partial[g], \sigma[g], \phi[L], \eta[h], \theta[h], \upsilon[H], \tau[H]\}$ and C a labeled component.*

Like in the previous section, labeled components can be normalized. Because of the added complexity, the normal form is less clear-cut. Essentially it consists of the same order: hiding, then renaming, then a bunch of consumptions/productions applied to the merge of several places and transitions. Minor extensions and alterations of our operators may lead to a more or less smooth normal form. The ideas behind the normalization have been introduced in the previous section: we establish the way that the transformations "commute" if they are in the "wrong" order and use the "fresh label renaming trick" to invert the order of transformation and merge.

A language is still needed for representing the functions, sets and relations between values. There exist many languages that will fit the bill nicely.

5 Example

We can represent components textually by scripts that define functions, sets and relations to go with the operators above. A tentative example script is given below.

```
E[action n, action p] :=
 hide[place x, action c] in encap[c with c <> 0] in
  sync[c with c=0 with n] in fuse[x] in
   (C[place y renamed to x, action c] merged to
    D[place x, action n,p]);
C[place y:int, action c:nat] :=
 hide[action d,e] in consume[d from y with d=y] in
   produce[c to y with c=y, e to y with e=y] in
    (trans[d:int, e:int with e-d < 2 and d-e < 2] merged to
     trans[c:nat] merged to place[y:int init <0>])
D[place x:int, action n,p] :=
 consume[p from x with x>0, n from x with x<0] in
   (trans[p] merged to trans[n] merged to place[x:int init empty)])
```

The semantics of this script is as follows:
$E = (\upsilon[\{\mathrm{x}\}] \centerdot \tau[\{\mathrm{c}\}] \centerdot \partial[\{(\mathrm{c}, \mathbb{N} \setminus \{0\})\} \centerdot \sigma[\{(\mathrm{c}, \{0\}), (\mathrm{n}, \mathcal{V})\}] \centerdot \phi[\{\mathrm{x}\}]).(\eta[\{(\mathrm{y}, \mathrm{x})\}].C \mid\mid D)$, with
$C = (\tau[\{\mathrm{d}, \mathrm{e}\}] \centerdot \chi[\{((\mathrm{d}, \mathrm{y}), \{(d, y) \mid d = y\})\}] \centerdot \pi[f]).M$ and
$M = (T[\{\{(\mathrm{d}, d), (\mathrm{e}, e)\} \mid d, e \in \mathbb{Z} \wedge -2 < d - e < 2\}] \mid\mid T[\{\{(\mathrm{c}, c)\} \mid c \in \mathbb{N}\}] \mid\mid P[\mathrm{y}, \langle 0 \rangle]$ and
$f = \{((\mathrm{c}, \mathrm{y}), \{(c, y) \mid c = y\}), ((\mathrm{e}, \mathrm{y}), \{(e, y) \mid e = y\})\}$ and
$D = (\chi[\{((\mathrm{p}, \mathrm{x}), \{(p, x) \mid x > 0\})\}] \centerdot \chi[\{((\mathrm{n}, \mathrm{x}), \{(n, x) \mid x < 0\})\}]).N$ and
$N = T[\{\{(\mathrm{p}, \bullet)\}\}] \mid\mid T[\{\{(\mathrm{n}, \bullet)\}\}] \mid\mid P[\mathrm{x}, 0]$.

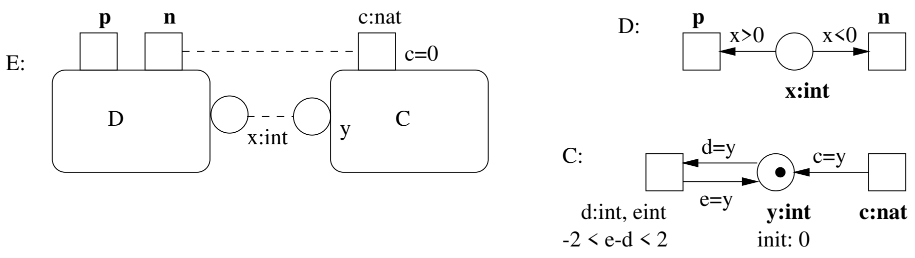

Figure 3: Colored Petri Net of example script

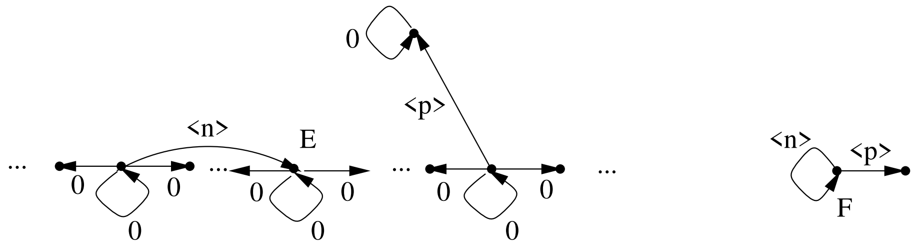

Figure 4: Transition system and reduction of example

Here the • signifies some value that has no importance. The identifiers in `typescript` font are to be interpreted literally (i.e. as labels), the others are placeholders for the definition of sets and functions. We can graphically represent this script by the CPN-like annotated net in Figure 3. There, the identifiers in boldface represent the external labels.

We can normalize the above script and arrive at the following script. We will forgo a semantics and graphical representation.

```
E[action n, action p] :=
 hide[place x, action d,e] in
 consume[d from x with d=x, p from x with x>0, n from x with x<0] in
   produce[n to x with x=0, e to x with e=x] in
     (trans[d:int, e:int with e-d < 2 and d-e < 2] merged to
      trans[n] merged to trans[p] merged to place[x:int init <0>])
```

The transition system of E is shown in Figure 4. Its reduction modulo branching bisimilarity is shown as F in the same figure.

6 Conclusion

This paper defines a semantics for colored hierarchical Petri nets, which supports both action-oriented and state-oriented approaches, matching the dual nature of Petri nets. The compositional nature of our framework allows the specification of components that can be implemented independently respecting the specified interface.

Our paper has been inspired by process algebra, more specifically by the wish to be able to reason about and manipulate with Petri nets like process algebra terms. When comparing our framework to a process-algebraic one like μCRL [7], one notices the same expressive power. However, the presence of true sequencing and choice operators in μCRL makes it better suited for specifying technical systems like data communication protocols, whereas the added connection possibilities, graphical nature and the step (instead of interleaving) semantics of our framework seem to give it a slight edge for programming-in-the-large.

We already discussed the relation of this paper to [2], [12]. It is also interesting to compare our approach to the Petri Box calculus originated by [3]. The Petri Box calculus is aimed at giving a Petri net (and thus causal) semantics for the language B(PN)^2 [4], [10], somewhat similar to μCRL. Our aim is to use colored net components as a high-level specification and design tool.

It may be worthwhile to extend our step semantics to more "truly concurrent" ones. This would allow e.g. an action refinement operator in our calculus, which is not compositional in step semantics. However, we hesitate to sacrifice the simplicity of transition systems. Our approach gives the modeler the option of preserving the causal connections between actions by not hiding all places (causes) between the transitions involved.

Much work is still needed to arrive at practical verifications, let alone "best practice" rules for deriving correct models from a set of requirements, but we believe that this paper is a serious step towards this goal.

References

[1] J.C.M. Baeten and C. Verhoef. Concrete Process Algebra. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 149–268. Oxford University Press, Clarendon, UK, 1995.

[2] T. Basten and M. Voorhoeve. An Algebraic Semantics for Hierarchical P/T Nets. In G. De Michelis and M. Diaz, editors, *Proceedings ATPN '95*, volume 935 of *Lecture Notes in Computer Science*, pages 45–65, Torino, Italy, 1995. Springer–Verlag, Berlin.

[3] E. Best, R. Devillers, and J. Hall. The Petri Box Calculus: a New Causal Algebra with Multilabel Communication. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 21–69. Springer–Verlag, Berlin, 1992.

[4] E. Best and R.P. Hopkins. B(PN)^2 - a Basic Petri Net Programming Notation. In A. Bode, M. Reeve, and G Wolf, editors, *Proceedings PARLE '93*, pages 379–390. Springer–Verlag, Berlin, 1993.

[5] Wan Fokkink. Rooted branching bisimulation as a congruence. *Journal of Computer and System Sciences*, 60(1):13–37, 2000.

[6] R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[7] J.F. Groote and M.A. Reniers. Algebraic Process Verification. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 1151–1208. Elsevier Science BV, Amsterdam, 2001.

[8] K.M. van Hee. *Information Systems Engineering: a Formal Approach*. Cambridge University Press, Cambridge, 1994.

[9] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* EATCS monographs on Theoretical Computer Science. Springer–Verlag, Berlin, 1992.

[10] H. Klaudel. Compositional high-level Petri net semantics of a parallel programming language with procedures. *Science of Computer Programming* (to appear).
url: `www.univ-paris12/~lacl/klaudel/proc.ps.zip`.

[11] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Dept., Aarhus University, 1981.

[12] L Priese and H Wimmel. A uniform approach to true-concurrency and interleaving semantics for Petri nets. *Theoretical Computer Science*, 206:219–256, 1998.