pNets: an Expressive Model for Parameterised Networks of Processes

Ludovic Henrio, Eric Madelaine, Min Zhang

pNets: an Expressive Model for Parameterised Networks of Processes

Ludovic Henrio, Eric Madelaine
Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271
INRIA Sophia Antipolis Méditérannée,
BP 93, 06902 Sophia Antipolis, France
ludovic.henrio@cnrs.fr, eric.madelaine@inria.fr

Min Zhang
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, P.R. China 200062
mzhang@sei.ecnu.edu.cn

***Abstract*—This article studies Parameterised Networks of Automata (pNets) from a theoretical perspective. We illustrate the expressiveness of pNets by showing how to express a wide range of classical constructs of (value-passing) process calculi, but also how we can easily encode complex interaction patterns used in modern distributed systems. Our framework can model full systems, using (closed) hierarchies of pNets; we can also build (open) pNet systems expressing composition operators. Concerning more fundamental aspects, we define a strong bisimulation theory specifically for the pNet model, prove its properties, and illustrate it on some examples. One of the original aspects of the approach is to relate the compositional nature of pNets with the notion of bisimulation; this is exemplified by studying the properties of a flattening operator for pNets.**

I. Introduction

The pNet model [1] has been used to provide a behavioural specification formalism in different contexts; it is particularly powerful for expressing the behaviour of distributed objects [2] and distributed components [1]. pNets is a very powerful model for expressing the behaviour of distributed applications featuring queues, futures, component systems, one-to-many communications, or fault-tolerance protocols [2], [3], [4]. pNets are used in a verification environment called Vercors.[1] However the theoretical foundations of pNets or their expressiveness have never been studied. This paper provides a formal background for pNets ranging from operational semantics to strong bisimulation.

This paper shows that it is easy to express in pNets many constructs of classical value-passing algebras, including CCS and Lotos; pNets handle e.g. the usual binding mechanism within CCS communication, as well as the *gate negotiation* semantics of Lotos. Then we show that pNets allow the expression of more advanced synchronisation patterns. Then we define a strong (early) bisimulation equivalence for closed pNets, and prove that it is a congruence for pNet contexts. This bisimulation is illustrated by two applications: one inspired from value-passing CCS, the other is a generic proof on an operator flattening a pNet structure.

Many more examples of representation in pNets for various synchronisation artifacts, and the full proofs of the properties, are given in the extended version of this paper [5].

[1] https://team.inria.fr/scale/software/vercors/

II. Parameterised Networks (pNets): definition

This section defines pNets and the notations we will use in this paper together with an operational semantics for pNets.

pNets are tree-like networks of processes: they provide means to represent in a structured and hierarchical way the behaviour of processes, represented as parameterized labelled transition systems (pLTS, similar to finite state machines with value-passing messages). Composition of pNets is realized by *synchronisation vectors*, that relate the actions of (a subset of) the subnets, with a global action that will be exported at the next level. A pNet node may have an unbounded number of subnets, and in practice a synchronisation vector will be encoded as a finite number of (potentialy unbounded) indexed families of subnets.

Last, but not least, at the leaves of a pNet tree, you can also have *holes*, that play the role of process parameters. A hole has a *Sort* defining the set of its possible actions. A pNet tree with at least one hole is called an *open pNet*.

Notations: We extensively use indexed structures over some countable sets, which are equivalent to mapping over the countable set. $a_i^{i\in I}$, or equivalently $(i{\mapsto}a_i)^{i\in I}$ denotes a family of elements a_i indexed over the set I. $a_i^{i\in I}$ defines both I the set over which the family is indexed (called *range*), and a_i the elements of the family. E.g., $a^{i\in\{3\}}$ is the mapping with a single entry a at index 3; also denoted $(3{\mapsto}a)$ in the following. When this is not ambiguous, we shall use notations for sets, and typically write "indexed set over I" when formally we should speak of multisets, and write $x \in a_i^{i\in I}$ to mean $\exists i \in I.\, x = a_i$. An empty family is denoted $\emptyset$. $\uplus$ is the disjoint union on indexed sets (meaning both indices and elements should be distinct). $\mathcal{I}_\mathcal{P}$ is the set of all index sets.

Term algebra: Our models rely on the notion of parameterised actions. We leave unspecified the constructors of the algebra that will allow building actions and expressions. We denote by Σ the signature of those constructors and by $\mathcal{T}_\mathcal{P}$ the term algebra of Σ over the set of variables $\mathcal{P}$. We suppose that we are able to distinguish inside $\mathcal{T}_\mathcal{P}$ a set of *action terms* denoted $\mathcal{A}_\mathcal{P}$ (*parameterised actions*), a set of *expression terms* denoted $\mathcal{E}_\mathcal{P}$, and, among expressions, a set of *Boolean expressions* (guards) denoted $\mathcal{B}_\mathcal{P}$.[2] For each term $t \in \mathcal{T}_\mathcal{P}$ we define *fv*(t) the set of free variables of t. For $\alpha \in \mathcal{A}_\mathcal{P}$, *iv*$(\alpha)$ returns a subset of *fv*(α) which are the input variables of α,

[2] $\mathcal{E}_P \cap \mathcal{A}_P = \emptyset$ $\quad$ $\mathcal{B}_\mathcal{P} \subseteq \mathcal{E}_\mathcal{P}$ $\quad$ $\mathcal{A}_P \cup \mathcal{E}_\mathcal{P} \subseteq \mathcal{T}_\mathcal{P}$

i.e., the variables newly defined by reception of their value during the action α.

pNets can be used with any term algebra. In our examples we will mainly use classical input-output interactions with parameters, *à la* value-passing CCS [6]. These will be written $?a(x_1, ..., x_n)$ for inputs, $!a(v_1, .., v_n)$ for outputs, or $a(v_1, .., v_n)$ and τ for synchronised actions; in that case we have $fv(?a(x)) = iv(?a(x)) = \{x\}$, and for a value v $iv(!a(v)) = \emptyset$. In other cases we use Lotos-style offers, with specific δ and $accept(x)$ action terms, or Meije-SCCS action monoids, like in $a.b$, $a^{f(n)}$. The expressiveness of the synchronisation constructs will depend on the action algebra.

A. The pNets Model

The first definition of pNets was published in [1].

Definition II.1 (pLTS)**.** *A pLTS is a tuple $pLTS \triangleq \langle\!\langle S, s_0, \rightarrow \rangle\!\rangle$ where S is a set of states, with s_0 the initial one. $\rightarrow \subseteq S \times T \times S$ is the transition relation with labels T of the form: $\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J}\rangle$ where α are actions, e_b are guards, and variables are assigned when the transition is performed.*

A *sort* is a set of parameterized actions. It can be viewed as the signature of the pNet. In the case of a pLTS, it is the set of actions within the labels appearing on the transitions:

$$\mathrm{Sort}(\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle) = \{\alpha | \exists s, s', e_b, e_j^{j\in J}, t, x_j^{j\in J}. (s, t, s') \in \rightarrow \wedge\ t = \langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J}\rangle\}$$

pNets are constructors for hierarchical behavioural structures: a pNet is formed of other pNets, or pLTSs at the bottom of the hierarchy tree.

Definition II.2 (pNets)**.** *A pNet is a hierarchical structure where leaves are pLTSs:*
$pNet \triangleq pLTS \mid \langle\!\langle pNet_i^{i\in I}, S_h^{h\in H}, SV_k^{k\in K} \rangle\!\rangle$ where

- *$pNet_i^{i\in I}$ is the family of sub-pNets where $I \in \mathcal{I}_\mathcal{P}$ is the set over which sub-pNets are indexed.*
- *$H \in \mathcal{I}_\mathcal{P}$ is the set over which holes are indexed, each hole h is of sort S_h. I and H must be* disjoint*: $I \cap H = \emptyset$*
- *$SV_k^{k\in K}$ is a set of synchronisation vectors ($K \in \mathcal{I}_\mathcal{P}$). $\forall k \in K, SV_k = \alpha_{lk}^{l\in L_k} \rightarrow \alpha'_k$ where $\alpha'_k \in \mathcal{A}_\mathcal{P}$, $L_k \in \mathcal{I}_\mathcal{P}$, $\emptyset \subset L_k \subseteq I \uplus H$, $\forall l \in L_k \cap I. \alpha_{lk} \in \mathrm{Sort}(pNet_l)$, and $\forall l \in L_k \cap H. \alpha_{lk} \in S_l$. We denote $\mathrm{Label}(SV_k) = \alpha'_k$.*
- *The set of holes of the sub-pNets, and of local holes of the pNet are all disjoint:*

$$\forall i \in I.\ \mathrm{Holes}(pNet_i) \cap H = \emptyset$$
$$\forall i_1, i_2 \in I. i_1 \neq i_2 \Rightarrow \mathrm{Holes}(pNet_{i_1}) \cap \mathrm{Holes}(pNet_{i_2}) = \emptyset$$

The preceding definition relies on the two functions defined below.

Definition II.3 (Sorts)**.** *The sort of a pNet is the set of actions that a pNet can perform:*

$$\mathrm{Sort}(\langle\!\langle \overline{pNet}, \overline{S}, SV_k^{k\in K} \rangle\!\rangle) = \{Label(SV_k) | k \in K\}$$

Definition II.4 (Holes)**.** *The set of holes of a pNet is defined inductively as follows:*

$$\mathrm{Holes}(\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle) = \emptyset$$
$$\mathrm{Holes}(\langle\!\langle pNet_i^{i\in I}, S_h^{h\in H}, \overline{SV} \rangle\!\rangle) = H \cup \bigcup_{i\in I} \mathrm{Holes}(pNet_i)$$

A pNet Q is *closed* if it has no hole: $\mathrm{Holes}(Q) = \emptyset$; else it is said to be *open*.

When $I \cup H = [0..n]$ we denote synchronisation vectors: $< \alpha_1, .., \alpha_n > \!\!\rightarrow \alpha$, and elements not taking part in the synchronisation are denoted $-$ as in: $< -, -, \alpha, -, - > \!\!\rightarrow \alpha$.

Definition II.5 (pNet composition)**.** *An open pNet: $pNet = \langle\!\langle pNet_i^{i\in I}, S_h^{h\in H}, \overline{SV} \rangle\!\rangle$ can be (partially) filled by providing a family of pNets $(pNet'_l)^{l\in L}$ of the right sort to fill its holes.*

Suppose $\forall l \in H \cap L.\ \mathrm{Sort}(pNet'_l) \subseteq S_l, then$[3]:

$$pNet\left[(pNet'_l)^{l\in L}\right] = \langle\!\langle \left(pNet_i\left[(pNet'_l)^{l\in L}\right]\right)^{i\in I} \uplus (pNet'_l)^{l\in H\cap L}, S_h^{h\in H\setminus L}, \overline{SV} \rangle\!\rangle$$

Note that L can be larger than H. In such a case, the elements of $(pNet'_l)^{l\in L\setminus H}$ can fill holes at an arbitrary depth inside the tree structure of *pNet*.

We use graphical representations of pNets (see e.g. the $SV_{Barrier}$ in sectionIII.3), in which a pNet is drawn by a box with subnets represented as circles on the left of the box. When a subnet is a hole, the sort is given in the circle and there is a line leading to nothing, decorated with the index below the circle. If the subnet is given, the line is connected to a pNet. On the right part we put the synchronisation vectors, often as a reference to a term outside the box.

B. An operational semantics for pNets

Now we give an operational semantics for the pNet model; it is based on a valuation domain for the variables of the pNet, that can be finite or infinite. The operational semantics is defined for *closed* pNets only; the semantics for a pNet with a hole is more complicated, it should be a function of the pNet put in the hole. We suppose the existence of a unique *valuation domain* $\mathcal{D}$. We require that for a boolean expression e, if all the variables of e are given a value in $\mathcal{D}$ it is possible to decide whether e is true or false. $\phi = \{x_j \rightarrow V_j | j \in J\}$ is a valuation function, mapping x_j, which range over variables of the considered pNet, to $V_j \in \mathcal{D}$, values. For a term $t \in \mathcal{T}_\mathcal{P}$, $t\phi \in \mathcal{D}$ is the value of the term obtained by replacing each variable by their values given by ϕ. A valuation can also be applied to indexed sets. Φ is the set of valuations that can be explored. $\phi_1 + \phi_2$ replaces some of the values defined in ϕ_1 by the ones in ϕ_2; ϕ_2 might also define new entries.

The semantics of a pNet is denoted $\| - \|_\Phi$, which is a LTS[4]. It relies on a set of states for closed pNets $S_\Phi(-)$ that are hierarchical composition of states reflecting the structure of the pNet. $[\![-]\!]_\Phi$ defines transitions between states.

[3] else the composition is undefined

[4] A (classical) LTS is indeed a pLTS with no parameter

Definition II.6 (Operational semantics of closed pNets)**.** *Let $\phi_0 \in \Phi$ be an initial valuation associating a value to each variable of $\mathcal{P}$. The semantics of pNet is an LTS $\|pNet\|_\Phi = \langle\!\langle S_\Phi(pNet), S_0(pNet), \rightarrow \rangle\!\rangle$ where:*

- *The set of states $S_\Phi(pNet)$ is*[5]*:*

$$S_\Phi(\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle) = \{(s,\phi) | s \in S \wedge \ \phi \in \Phi\}$$
$$S_\Phi(\langle\!\langle pNet_i^{i\in I}, \emptyset, SV_k^{k\in K} \rangle\!\rangle) = \{\lhd s_i^{i\in I\phi} \rhd \,|\, \phi \in \Phi \wedge \forall i \in I\phi.\, s_i \in S_\Phi(pNet_i)\}$$

- *The initial state $S_0(pNet)$ is:*

$$S_0(\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle) = (s_0, \phi_0)$$

$$S_0(\langle\!\langle pNet_i^{i\in I}, \emptyset, SV_k^{k\in K} \rangle\!\rangle) = \lhd S_0(pNet_i) \rhd^{i\in I\phi_0}$$

- *labels are $\{\alpha\phi | \alpha \in \mathrm{Sort}(pNet) \wedge \phi \in \Phi\}$;*
- *and transitions are defined as $[\![pNet]\!]_\Phi$, the smallest set of transitions satisfying the rules below.*

$$\frac{\begin{array}{c}\phi, \phi', \phi'' \in \Phi \qquad s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J}\rangle} s' \in \rightarrow \\ \phi' = \phi + \{x_i' \rightarrow V_i | x_i' \in iv(\alpha) \wedge V_i \in \mathcal{D}\} \\ e_b\phi' = True \qquad \phi'' = \phi' + \{x_j \rightarrow e_j\phi' | j \in J\}\end{array}}{(s,\phi) \xrightarrow{\alpha\phi'} (s', \phi'') \in [\![\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle]\!]_\Phi} \textbf{Tr1}$$

$$\frac{\begin{array}{c}\phi, \phi' \in \Phi \qquad \alpha_l^{l\in L} \rightarrow \alpha \in \overline{SV} \qquad \forall i \in I\phi \setminus L\phi.\, s_i' = s_i \\ \forall l \in L\phi.\, \phi_l \in \Phi \wedge s_l \xrightarrow{\alpha_l\phi_l} s_l' \in [\![pNet_l]\!]_\Phi \qquad \phi' = \phi \uplus \{\phi_l\}^{l\in L\phi}\end{array}}{\lhd s_i^{i\in I\phi} \rhd \xrightarrow{\alpha\phi'} \lhd s_i'^{i\in I\phi'} \rhd \in [\![\langle\!\langle pNet_i^{i\in I}, \emptyset, \overline{SV} \rangle\!\rangle]\!]_\Phi} \textbf{Tr2}$$

Only states of pLTSs are associated with a valuation, but states of the pNets still use the valuation to decide (expand) the set of sub-pNets embedded in the pNet. Rule **Tr1** deals with transitions between states of the pLTS; only input variables and assigned variables are allowed to change value in the valuation; the resulting valuation is obtained by the successive updates from the input variable assignments, and the explicit assignments from the transition. Remark that the predicate in a transition is evaluated in a valuation that includes the values carried by the communication action, allowing for expressing non-local decisions as in Lotos *gate negotiation*. Note also that there is *a priori* no predefined direction for the flow of data because it is dependent on the chosen term algebra. Rule **Tr2** deals with transitions between states of the pNet; A given synchronisation is picked, it involves the set $L \subseteq I$ of sub-pNets. The resulting valuation is obtained by the combination of all updates happening in the sub-pNets involved in the synchronisation ($\phi \uplus \{\phi_l\}^{l\in L\phi}$ in the rule); sub-pNets not involved in the transition keep the same state.

The fact that each variable has a complete instantiation ensures that the semantics is a fully instantiated LTS that has no more variable. This condition is crucial for the decidability of strong bisimilarity presented in Section IV.

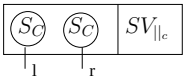
[5]We use the $\lhd \ldots \rhd$ notation to easily identify states of the semantics.

III. Examples and Expressiveness

This section illustrates the expressiveness of pNets by exhibiting pNets for several classical constructs of process calculi. These pNets "behave" similarly to the original construct: the reachable states and traces simulate faithfully the original construct. Proving richer equivalence between the modelled construct and the pNet or providing a general results on expressiveness of pNets is out of the scope of this paper.

We first focus on value passing algebras. The first example models the parallel operator of CCS, synchronising input and output actions, or transmitting those actions unchanged to the outside.

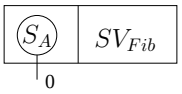
Example III.1 (CCS-like Synchronisation)**.** Let C be a set of channel names, V a set of values. We denote S_C the sort: $S_C = \{\tau\} \cup \{?a(x) | a \in C,\, x \in \mathcal{P}\} \cup \{!a(v) | a \in C,\, v \in V\}$. The pNet of the synchronisation operator, $||_C$, is defined as:



$$\begin{aligned} SV_{||_c} = \{&< !a(x), ?a(x) > \rightarrow \tau\}_{a:C,x:P,xfresh} \\ \cup \{&< ?a(x), !a(x) > \rightarrow \tau\}_{a:C,x:P,xfresh} \\ \cup \{&< b, - > \rightarrow b\}_{b:S_C} \\ \cup \{&< -, b > \rightarrow b\}_{b:S_C} \end{aligned}$$

There are two holes indexed l and r (synchronisation vectors are written taking $l = 0$ and $r = 1$). There are four families of synchronisation vectors: the first two sets of vectors synchronise input and output actions from the process to the right (resp. left) to the one to the left (resp. right), it is visible as a τ action; the two last sets allow any action of a process to be visible at the level of the parallel operator, and potentially to be synchronised by another parallel operator. Value passing is obtained by an adequate definition of the valuation domains of the variables: for each fresh variable x introduced by the synchronisation vector for channel c, the valuation domain should include the set of values that can be transmitted over the channel c. This set can be any over-approximation, and can be refined by typing on channels. Note that input and output actions have a symmetric role in the synchronisation vector, they will be distinguished in the way they can appear in pLTSs, and in the way they are handled by rule **Tr1**.

Example III.2 (Fibonacci.)**.** Here we have a single hole with sort $S_A = \{a^n | a \in A, n \in \mathbb{N}\}$ with A a set of action names. Then the open pNet Fib transforms any action of the subnet by increasing its "intensity".

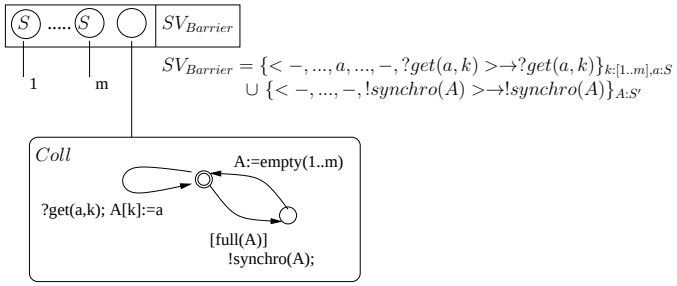


$$SV_{Fib} = \{< a^n > \rightarrow a^{fib(n)}\}_{a:A,n:Nat}$$

Reversely, we define a many-to-one communication acting as a synchronisation barrier. It is a synchronising pattern in an asynchronous system: it waits for events from a set of processes, and when all events have arrived it creates some sort of "global data" from data received, and sends it to an external process.

Example III.3 (Synchronisation Barrier)**.** This pNet collects actions from a set of m pNets. Its holes, of sort S, are indexed over a set $N = [1..m]$. The pNet also contains a collecting process $Coll$ that gathers individual 'a' messages from the

holes, and stores them in an array (if any hole sends 'a' twice, the new value replaces the older one); when the array is full it transmits the resulting array as an argument of the 'synchro' global action. The sort of the SBarrier pNet is $\{?get(a,k)|a \in S \wedge k \in [1..m]\} \cup \{!synchro(A)|A \in S'\}$, where S' is the set of arrays of length m containing elements in S.



This section has illustrated the expressiveness of pNets through examples. We have shown their capacity to express rich composition patterns mimicking both classical synchronisations, and modern many-to-many interactions.

IV. Strong Bisimulation for pNets

In this section, we define a notion of (strong) bisimulation for closed pNets, and prove that this equivalence is a congruence w.r.t. any kind of pNet context. We first define a context in the pNet formalism. Defining in pNets a context syntactically as one would do in usual process algebras is possible. However, pNets already provide a way to define expressions with holes. A context pNet is a pNet C with a single hole, we index this hole by $\bullet$. To alleviate proofs, we distinguish contexts featuring a single hole at the top-level of the pNet structure (we call them *top1-contexts*), from contexts having a single hole at any depth.

Definition IV.1 (contexts)**.**
A top1-context *of sort S is a pNet C of the form $\langle\!\langle \overline{pNet}, (\bullet \mapsto S), \overline{SV} \rangle\!\rangle$ where $\overline{pNet}$ is a family of closed pNets.*

A 1-context *of sort S is either: a top1-context pNet of sort S: $\langle\!\langle \overline{pNet}, (\bullet \mapsto S), \overline{SV} \rangle\!\rangle$, or a pNet $\langle\!\langle \overline{pNet}, \emptyset, \overline{SV} \rangle\!\rangle$, in which one (and only one) of the $pNet_i$ is a 1-context of sort S.*

Recall that the operation $pNet[\{P_i\}]$ fills some holes of *pNet*, with $\{P_i\}$ a family of pNets indexed with the indexes of the holes. Then, provided $\mathrm{Sort}(Q) \subseteq S$, $C[(\bullet \mapsto Q)]$ is the closed pNet where the hole is replaced by Q.

a) Strong bisimulation: Strong bisimulation of pNets is relatively classical except that it relates instantiated states, and that the two bisimilar states of pNets can perform different transitions with different valuations provided the application of the two valuations on the two labels give the same result.

Definition IV.2 (Strong Bisimulation)**.** *Let P and Q be two closed pNets. A relation $R \subseteq S_\Phi(P) \times S_\Phi(Q)$ is a* strong simulation *if for each $(s,t) \in R$, for any valuation set $\phi \in \Phi$, whenever $s \xrightarrow{\alpha\phi} s' \in [\![P]\!]_\Phi$, then $\exists \phi' \in \Phi, \beta, t'$, such that $t \xrightarrow{\beta\phi'}_\Phi t' \in [\![Q]\!]_\Phi$ and $(s',t') \in R$ and $\alpha\phi = \beta\phi'$.*

A relation R is a strong bisimulation *if R and R^{-1} are strong simulations.*

Consider two closed pNets P and Q, two states $s \in S_\Phi(P)$ and $t \in S_\Phi(Q)$ are strongly bisimilar*, denoted by $(\!|s,P|\!) \sim (\!|t,Q|\!)$ iff $(s,t) \in R$, for some strong bisimulation $R \subseteq S_\Phi(P) \times S_\Phi(Q)$.*

Two closed pNets P and Q are strongly bisimilar*, denoted by $P \sim Q$, iff their initial states are strongly bisimilar: $(\!|S_0(P), P|\!) \sim (\!|S_0(Q), Q|\!)$*

Observe that the bisimilarity between pNets only implies that their reachable states are equivalent.

Theorem IV.3 (Strong bisimulation is a congruence)**.** *Let P and Q be two bisimilar closed pNets: $P \sim Q$. Then for any 1-context C of sort S, such that $\mathrm{Sort}(P) \cup \mathrm{Sort}(Q) \subseteq S$, we have $C[(\bullet \mapsto P)] \sim C[(\bullet \mapsto Q)]$.*

The details on the proof of this property and the intermediate lemmas (e.g. the fact that bisimulation is a congruence for top1-contexts) can be found in the extended version of this paper [5].

b) Application to Flattening Operator: Finally, this section presents a flattening operation for pNet contexts and proves strong bisimilarity. The purpose of this section is to illustrate further the interaction between bisimulation and pNets structure on an example that is useful both from a practical and from a theoretical point of view. The flattening operator can be used to flatten a pNet hierarchy, it is used when building a model for a large system (to reduce the state-space) and can also be used to ease compositional formal reasoning. We suppose here that the action algebras limits the parameters of the actions to be only variables. This restriction avoids us to have to deal with unification between action terms here; it is possible to add a pNet in the flattened structure dealing with the unification.

Definition IV.4 (Flattening)**.** *Let $C' = \langle\!\langle \overline{pNet'}, (\bullet \mapsto S'), \overline{SV'} \rangle\!\rangle$, and $C = \langle\!\langle \overline{pNet}, (\bullet \mapsto S), \overline{SV} \rangle\!\rangle$ with $\mathrm{Sort}(C') \subseteq S$; we define $\mathrm{Flat}(C,C')$ as follows:*

$$\mathrm{Flat}(C,C') = \langle\!\langle \overline{pNet} \uplus \overline{pNet'}, (\bullet \mapsto S'), \overline{SV''} \rangle\!\rangle \qquad \textit{where}$$

$$\begin{aligned}\overline{SV''} = \{&(\alpha_l \varphi)^{l \in L \setminus \{\bullet\}} \uplus \beta_l^{l \in L'} \rightarrow \alpha' \varphi \,|\alpha_l^{l \in L} \rightarrow \alpha' \in \overline{SV} \wedge \\ &\beta_l^{l \in L'} \rightarrow \beta \in \overline{SV'} \wedge \bullet \in L \wedge \alpha_\bullet \varphi = \beta \wedge \\ &\varphi = (x_n \rightarrow y_n)^{n \in N} \wedge \forall n \in N. (x_n \in fv(\alpha_\bullet) \wedge y_n \in fv(\beta))\} \\ &\cup \{(\alpha_l)^{l \in L} \rightarrow \alpha' \,|(\alpha_l)^{l \in L} \rightarrow \alpha' \in \overline{SV} \wedge \bullet \notin L\}\end{aligned}$$

This definition uses the fact that action parameters can only be variables, and thus an action β triggered by a subnet can only match the action $\alpha_\bullet$ in the synchronisation vector if there exist φ that assigns to each free variable of $\alpha_\bullet$ a variable of β such that the two actions are the same. φ is then applied to all the actions α_i of the original synchronisation vector. Now we prove the most interesting property of the flattening: it preserves strong bisimulation:

Proposition IV.5 (Flattening and strong bisimulation)**.**

$$\forall P \textit{ closed pNet}, C[\bullet \mapsto C'[\bullet \mapsto P]] \sim Flat(C,C')[\bullet \mapsto P]$$

This property has several consequences, as it shows that it is sufficient to prove congruence for the top-level contexts instead of checking holes occurring deeper in the pNet tree. We can then derive a Flatten operator that flattens a closed pNet P hierarchy, it produces a pNet composing only pLTSs.

V. Related Works

As mentioned in the introduction, this research naturally inherits from the seminal work by Arnold and Nivat on synchronisation vectors [7]. The theoretical expressiveness of (closed) pNets is the same, but their tree structure is essential to search for compositionality, and for the definition of open structures. Moreover, Arnold/Nivat synchronisation vectors did not include explicit data values; in our approach, this ability is very important for usability in verification tools.

Our work on pNets is definitely at a "semantic" level, and we shall not compare it to process algebras. The closest interesting works are with NT-Lotos [8] and BIP [9]. NT-Lotos was developped as an execution language for Lotos, but appeared expressive enough to encode the Π-calculus. Still it is closely bound to specific parallel operators; the direct multi-way and parameterised synchronisation vectors of pNets allow for a much more flexible and direct encoding of other mechanisms. BIP is a formal framework that allows building and analysing complex (synchronous and asynchronous) component-based systems. Its core dialect does not allow for explicit value-passing. On the other hand, the priority functionality of BIP would require to use some kind of "negative premisses", that we cannot do in the current version of pNets. Finally, one concept that seems to fit naturally with the pNet approach, is with formalisms dedicated to the external coordination of components, as e.g. the REO connectors language [10].

The closest work to this paper may be the efforts in the nineties [11] to define the tyft/tyxt format and its extensions. The expressive power of synchronisation vectors is similar to that of SOS rules with positive hypothesis and structured transition labels. Close to the limits of what pNets (and tyft/tytx) can express, you would find replication mechanisms: in pNets this is encoded as the dynamic activation of a new instance inside one (unbounded) parameterised hole. One significant difference is that pNets allow for synchronisation of an unbounded number of subnets, while in the tyft/tyxt format and its variants (to the best of our knowledge), this is not possible. We already had a similar distinction in our introduction section, when comparing with the MEIJE-SCCS format [12]. Our approach with pNets is more constructive and pragmatic, as our main interests are for defining behavioural semantics for various programming languages in the parallel and distributed computation area, and we want our model, its expressiveness, and its limits to be compatible with an efficient implementation in a modeling and verification environment.

VI. Conclusion and Discussion

pNets have been used to provide a general behavioural specification formalism in different contexts, in particular it proved to be particularly powerful for expressing the behaviour of distributed objects and of distributed components.

In this paper, we illustrate the expressiveness of pNets by showing how they can be used to express a wide range of classical synchronisation patterns, including constructs of value-passing algebras. We have also exhibited more complex interaction and synchronisation patterns used for many-to-many communication schemes in distributed component-based applications. In that case, pNet parameters are used both as data values and as topology indexes.

We propose a behavioural semantics and a bisimulation theory, starting with a strong bisimulation relation for closed pNets. Both the semantics and the equivalence are based on all possible valuations, and can be thought of *early* versions. We prove that the strong bisimulation indeed is a congruence for all pNets contexts, and we provide a more constructive approach for combining pNets using a flattening operator.

Our next goal is to look for appropriate bisimulation relations for *open pNets*. Such equivalences should be considering hypotheses on the behaviour of holes, as was used in so-called FH-bisimulations in [12], but a particular challenge will be to find more constructive equivalences that will give us efficient approaches to build proofs in a compositional (congruent) manner. Such compositional theories and proof methods are of course essential in the analysis of hierarchical component systems. Another promising perspective is to investigate the proof of "symbolic" properties that are valid independently of the parameter valuations and to identify conditions on data and indexes in which one can prove such properties, such an approach could rely on previous works on symbolic reasoning for value passing processes [13]. The properties of congruence and of the flattening operator are promising first steps toward a bisimulation theory for open pNets.

References

[1] T. Barros, R. Boulifa, A. Cansado, L. Henrio, and E. Madelaine, "Behavioural models for distributed Fractal components," *Annals of Telecommunications*, vol. 64, no. 1–2, jan 2009, also Research Report INRIA RR-6491. [Online]. Available: http://hal.inria.fr/inria-00268965

[2] R. Boulifa, L. Henrio, and E. Madelaine, "Behavioural models for group communications," in *WCSI-10: International Workshop on Component and Service Interoperability*, ser. EPTCS, no. 37, 2010, pp. 42–56.

[3] R. A. Boulifa, R. Halalai, L. Henrio, and E. Madelaine, "Verifying safety of fault-tolerant distributed components," in *International Symposium on Formal Aspects of Component Software (FACS 2011)*, ser. Lecture Notes in Computer Science. Oslo: Springer, Sept 2011.

[4] T. Barros, L. Henrio, and E. Madelaine, "Behavioural models for hierarchical components," in *Proceedings of SPIN'05*. Spinger Verlag, 2005.

[5] L. Henrio, E. Madelaine, and M. Zhang, "pNets: an Expressive Model for Parameterised Networks of Processes," INRIA, Research Report RR-8579, Nov. 2014. [Online]. Available: http://hal.inria.fr/inria-00621264/en

[6] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989, ISBN 0-13-114984-9.

[7] A. Arnold, *Finite transition systems. Semantics of communicating sytems*. Prentice-Hall, 1994, iSBN 0-13-092990-5.

[8] R. Mateescu and G. Salaün, "Translating Pi-Calculus into LOTOS NT," in *IFM*, 2010, pp. 229–244.

[9] A. Basu, B. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis, "Rigorous component-based system design using the bip framework," *IEEE Softw.*, vol. 28, no. 3, pp. 41–48, May 2011.

[10] F. Arbab, "A channel-based coordination model for component composition," *Mathematical Structures in Computer Science*, vol. 14, pp. 329–366, 2004.

[11] J. Groote and F. Vaandrager, "Structured operational semantics and bisimulation as a congruence," *Information and Computation*, vol. 100(2), pp. 202–260, october 1992.

[12] R. de Simone, "Higher-level synchronizing devices in Meije-SCCS," *Theoretical Computer Science*, vol. 40, 1985.

[13] A. Ingólfsdóttir and H. Lin, "A symbolic approach to value-passing processes," in *Handbook of Process Algebra, chapter 7. Elsevier Science*. Elsevier, 2001.