High-Level Petri Nets as Type Theories in the Join Calculus

Maria Grazia Buscemi and Vladimiro Sassone

DMI, Università di Catania, Italy
buscemi@dmi.unict.it, vs@dmi.unict.it

Abstract. We study the expressiveness of the join calculus by comparison with (generalised, coloured) Petri nets and using tools from type theory. More precisely, we consider four classes of nets of increasing expressiveness, Π_i, introduce a hierarchy of type systems of decreasing strictness, Δ_i, $i = 0, \ldots, 3$, and we prove that a join process is typeable according to Δ_i if and only if it is (strictly equivalent to) a net of class Π_i. In the details, Π_0 and Π_1 contain, resp., usual place/transition and coloured Petri nets, while Π_2 and Π_3 propose two natural notions of high-level net accounting for dynamic reconfiguration and process creation and called reconfigurable and dynamic Petri nets, respectively.

1 Introduction

The join calculus [5,7] is an algebra of mobile processes with asynchronous name-passing communication that simplifies the π-calculus by enforcing the hypothesis of *unique receptors*. This means that there is at most one process receiving messages on a name, and is a distinctive feature of the join calculus that makes it suitable for distributed implementations, as channels may be allocated at their receptor process. The present work focuses on a version of the join calculus, studying its expressiveness by establishing a tight link to Petri nets.

Petri nets [16,17] are a fundamental model in concurrency, representing basic distributed machines that, although rudimentary, may exhibit complex interaction behaviours when processes (transitions) compete for shared resources (tokens). Operational in nature, Petri nets have been studied extensively from the semantic viewpoint (see [18] for some references).

The analogy between join terms and nets is relatively simple, and was first noticed in [1] and, recently, in [14]. Names, messages, and elementary definitions of the join calculus (cf. §2) correspond respectively to places, tokens, and transitions of Petri nets (cf. §3). The correspondence, however, runs short soon because nets are *not* a value-passing formalism and can express *no* mobility. Regrettably, they have a static, immutable network topology. This suggests to look for suitable extensions of Petri nets – in particular nets with mobility – that might be profitably applied to the study of mobile networks.

Supported by MURST project *TOSCA*. The authors wish to thank **BRICS**, Basic Research in Computer Science and project MIMOSA, INRIA Sophia Antipolis.

In the present paper we consider three extensions of place/transition Petri nets ($\mathit{\Pi}_0$) obtained by adding one by one, in a hierarchical fashion, the features needed to achieve the full expressiveness of join calculus. Namely, *value-passing*, accounted for by (a version of) *coloured nets* ($\mathit{\Pi}_1$), mobility as *network reconfigurability*, achieved by introducing *reconfigurable nets* ($\mathit{\Pi}_2$), and dynamically growing, *open networks*, modelled by the notion of *dynamic nets* ($\mathit{\Pi}_3$).

High level nets exist in several shapes and versions. The distinctive features of the mobile nets presented here is that input places of transitions are private and presets are immutable. A reconfigurable net can deliver tokens in different places at each firing, according to the input it receives, while a dynamic net can spawn a new net to life. But they *cannot* change the input arcs of any transitions. As in [13], where an algebra with such properties has been considered, we believe that this is the key to a tractable compositional semantic framework, the key to control generality. Most noticeably, it corresponds to unique receptiveness in the join calculus. A further naturality criteria in our view is that, as it turns out, dynamic nets can be proved *equivalent* to the join calculus.

The technical bulk of this work is characterising classes of join terms corresponding to the classes of the high-level nets discussed above. More precisely, our approach consists of designing four type systems $\mathit{\Delta}_i$, $i = 0, \ldots, 3$, that single out the terms corresponding to the nets in $\mathit{\Pi}_i$, respectively for $i = 0, \ldots, 3$. In particular, for a fixed, semantic-preserving, well-behaved translation $[\![\_]\!]$, a join term P is typeable in $\mathit{\Delta}_i$ if and only if $[\![P]\!]$ belongs to $\mathit{\Pi}_i$. System $\mathit{\Delta}_0$ is aimed at constraining terms to place/transition nets and, therefore, forbids dynamic process creation and nontrivial messages, $\mathit{\Delta}_1$ relaxes the latter limitation, but enforces a strict distinction between channels and values, supporting value-passing but forbidding any mobility. Such distinction is then relaxed in $\mathit{\Delta}_2$. System $\mathit{\Delta}_3$ relaxes the dynamicity constraint and, therefore, turns out to be trivial, i.e., each term is typeable in $\mathit{\Delta}_3$. That is, dynamic nets *coincide* with join terms. Finally, we provide a system $\mathit{\Delta}_4$ that sums the features of all the others.

It is worth remarking that these systems are very rudimentary from the type theoretic viewpoint: there are no complex types or rules, nor sophisticated issues such as polymorphism [8,15] or similar. However, we believe that our formalisation is quite interesting, suggestive, and worth pursuing, because the nature of join-terms makes it natural to express our conditions in systems of rules. Nevertheless, this paper remains a paper about comparing models. As a matter of fact, relating join calculus and Petri nets may be beneficial for both. On one of the edges of the connection, in fact, the join calculus may provide Petri nets with a compositional framework, together with behavioural semantics such as testing [10] and bisimulation [3]. Also, it may suggest interesting, semantically well-founded extensions of Petri nets, such as reconfigurable and dynamic nets here, mobile nets in [1], and functional nets in [14]. On the other edge, it opens the join calculus to an entire body of results on the semantics of noninterleaving concurrency, such as those supported by monoidal categories (see, e.g., [4]).

Here we actually consider a generalisation of the join calculus in that join patterns may not be linear, a choice that we discuss at length in §2.

Related Work. Although focus and approach here differ radically from [1], our models are clearly related to those of Asperti and Busi. While we aim at representing the join calculus precisely, the *dynamic nets* of [1] are more loosely inspired by it and are more general than ours. In particular, they do not enforce privateness of input places – that is unique receptors – as our nets do. The analogy between Petri nets and join calculus has also provided the inspiration for Odersky's *functional nets* [14]. Although *loc. cit.* is quite different in spirit from the present paper, the relationships between Odersky's framework and our ideas here are worth of further investigation. This, together with the fine comparison with [1], we leave to a later paper.

Structure of the Paper. The paper is organised as follows. In §2 we recall the basic definitions of the join calculus, while §3 introduces Petri nets, Δ_0 and relates typeability to PT nets. The following sections repeat the same pattern for coloured, reconfigurable, dynamic nets and the respective type systems. Finally, §7 introduces Δ_4. Due to space limitations all proofs are omitted.

2 The Join Calculus

We shall focus on a monadic version of the join calculus [5,7], writing its operational semantics in terms of a reduction system as in [15]. For a thorough introduction the reader is referred to the literature. Let $\mathcal{N}m$ be an infinite set of names and let $x, y \ldots, u, v, \ldots$ range over $\mathcal{N}m$. Join terms, definitions, and patters are given by the following grammar.

$$\begin{array}{ll}
\textsc{Terms} & P, Q ::= 0 \mid x\langle y\rangle \mid P \mid Q \mid \texttt{def}\ D\ \texttt{in}\ P \\[1ex]
\textsc{Definitions} & D, E ::= J \rhd P \mid D \wedge E \\[1ex]
\textsc{Join Patterns} & J, K ::= x\langle y\rangle \mid J \mid K
\end{array}$$

Thus, a join term P is either the 'null' term, an emission $x\langle y\rangle$ of message y on channel x, a parallel composition of terms, or a local definition. A definition D is a set of elementary definitions $J \rhd P$ matching join patterns J to terms P.

The sets of defined names *dn*, received names *rn*, and free names *fn* are defined below, and then extended to sets of terms in the obvious way.

$$\boxed{\begin{array}{llll}
rn(x\langle y\rangle) & = \{y\} & rn(J \mid K) & = rn(J) \cup rn(K) \\[1ex]
dn(x\langle y\rangle) & = \{x\} & dn(J \mid K) & = dn(J) \cup dn(K) \\
dn(D \wedge E) & = dn(D) \cup dn(E) & dn(J \rhd P) & = dn(J) \\[1ex]
fn(x\langle y\rangle) & = \{x\} \cup \{y\} & fn(P \mid Q) & = fn(P) \cup fn(Q) \\
fn(D \wedge E) & = fn(D) \cup fn(E) & fn(J \rhd P) & = dn(J) \cup (fn(P) \smallsetminus rn(J)) \\
\multicolumn{4}{c}{fn(\texttt{def}\ D\ \texttt{in}\ P) = (fn(P) \cup fn(D)) \smallsetminus dn(D)}
\end{array}}$$

A *renaming* σ is a map from names to names that is the identity except on a finite number of names. We indicate by $dom(\sigma)$ the set of names on which σ

is not the identity, and by $cod(\sigma)$ its image, using function application for the renaming of free names in terms.

Definition 1. The *structural congruence* $\equiv$ is the smallest substitutive (i.e., closed for contexts) equivalence relation on terms that satisfies the following rules, where θ and σ are *one-to-one* renamings such that $dom(\theta) \subseteq rn(D)$, $cod(\theta) \cap fn(D) = \varnothing$ and $dom(\sigma) \subseteq dn(D)$, $cod(\sigma) \cap fn(\theta D, P) = \varnothing$.

$$\texttt{def}\ D\ \texttt{in}\ P \equiv \texttt{def}\ \sigma\theta D\ \texttt{in}\ \sigma P \tag{1}$$

$$P \mid \texttt{def}\ D\ \texttt{in}\ Q \equiv \texttt{def}\ D\ \texttt{in}\ P \mid Q \qquad {\scriptstyle fn(P)\cap dn(D)=\varnothing} \tag{2}$$

$$\texttt{def}\ D\ \texttt{in}\ \texttt{def}\ E\ \texttt{in}\ P \equiv \texttt{def}\ D \wedge E\ \texttt{in}\ P \qquad {\scriptstyle dn(E)\cap(dn(D)\cup fn(D))=\varnothing} \tag{3}$$

plus equations stating that 0 is the unit for $\mid$ and that the operators $\mid$ and $\wedge$ are associative and commutative.

Taking $dom(\sigma) = \varnothing$, rule (1) expresses the α-equivalence of definitions up to a renaming of their received names, while considering $dom(\theta) = \varnothing$ we obtain α-equivalence of terms up to renaming of defined names. Rule (2) formalises the scope extrusion of names, and rule (3) states that, under conditions that avoid name clashes, definitions can equivalently be gathered together by the $\wedge$ connective. Interestingly, rule (3) is problematic in the design of polymorphic type systems for the join calculus [8,15], essentially because it may introduce polymorphic (mutual) recursion. This, however, is not an issue in the present setting. On the contrary, the law is instrumental in establishing our results.

The operational semantics adopted here corresponds to the original one based on the chemical abstract machine [2]. Terms within a reduction context play the role of 'molecules', definitions determine the 'reactions'. The reduction context determines which join pattern is matched in a definition and binds its received names. The reduction rule replaces in the context the matched pattern by the right-hand side of its definition.

Definition 2 (cf. [15]). The *reduction* $\longrightarrow$ is the smallest substitutive relation on $\equiv$-classes that satisfies the following rule:

$$\texttt{def}\ D \wedge J \triangleright P\ \texttt{in}\ \mathcal{C}[\sigma J] \ \longrightarrow\ \texttt{def}\ D \wedge J \triangleright P\ \texttt{in}\ \mathcal{C}[\sigma P] \qquad \text{if } dom(\sigma) \subseteq rn(J),$$

where D may be absent (if $J \triangleright P$ is the only definition), $\mathcal{C}$ is a *reduction context*, i.e.,

$$\mathcal{C} ::= [\,] \mid \texttt{def}\ D\ \texttt{in}\ \mathcal{C} \mid P \mid \mathcal{C},$$

and $\mathcal{C}[P]$ denotes the reduction context $\mathcal{C}$ with its hole filled by P.

On Linearity. Here we no longer require join patterns to be linear, i.e., we allow names to occur several times in patterns. There are two aspects to this. The first one consists of renouncing the linearity of defined names in join patterns. It presents no further difficulties for implementations [11], and we see no reason not to consider it. In terms of Petri nets it corresponds to adding multiplicities to arcs. More problematic in distributed implementations is relaxing the linearity of received names, because this amounts to equating values on different channels. Nevertheless we decided to adopt it, as it corresponds to a form of pattern matching essential in coloured Petri nets. It is worth remarking that our work is largely independent of this choice: we could as well keep linearity of received names at the price of disallowing matching in our coloured nets, and our results would stay, *mutatis mutandis*.

3 Place Transition Nets

A *multiset* on a set P is a function $\mu: P \to \mathbb{N}$. We shall use $\mu(P)$ to denote the set of finite multisets, i.e., markings, on P. The *sum* of $\mu_0, \mu_1 \in \mu(P)$ is the multiset $\mu = \mu_0 \oplus \mu_1$ such that $\forall p \in P.\ \mu(p) = \mu_0(p) + \mu_1(p)$.

Petri nets consists of places, transitions, and tokens. In view of relating nets and join terms, we identify places with names. In addition to $\mathcal{N}m$ – that will play the role of *public* places (*free* names) – we shall consider an infinite set ω of distinguished *private* places (*bound* names). In the following, $\mathcal{N}m_\omega$ stands for the union $\mathcal{N}m + \omega$.

Definition 3 (PT Nets). A *place/transition net* is a tuple $N = (T, \partial_0, \partial_1, \mu_0)$, where T is a finite set of transitions, $\partial_0, \partial_1: T \to \mu(\mathcal{N}m_\omega)$ are the pre- and post-set functions, and $\mu_0 \in \mu(\mathcal{N}m_\omega)$ is the initial marking. We shall use Π_0 to refer to PT Petri nets.

Thus, to simplify notations, we include in every net all the names of $\mathcal{N}m_\omega$ with the understanding that *only* marked places, i.e., those carrying tokens, and places connected to some transitions are effectively to be considered in the net. In particular, the *empty net*, that we denote by $\varnothing$, formally consists of all the places in $\mathcal{N}m_\omega$, but no tokens and no transitions. Analogously, the net consisting of a distribution of tokens on a *multiset* of places μ, has all places but only $\mu(x)$ tokens in each $x \in \mathcal{N}m_\omega$. With abuse of notations, we shall use μ to denote both the multisets and the corresponding net. Also, for $\mu_0, \mu_1 \in \mu(\mathcal{N}m_\omega)$, $\mathbf{tran}\,\mu_0\,\mu_1$ stands for the net with a unique transition t with $\partial_0(t) = \mu_0$ and $\partial_1(t) = \mu_1$.

Following our intuition, isomorphisms preserve free names. This is detailed in the definition below, where we use *id* for identities, + for function coproducts, and take the liberty of identifying functions with their extensions to multisets.

Definition 4. Nets $(T, \partial_0, \partial_1, \mu_0)$ and $(T', \partial'_0, \partial'_1, \mu'_0)$ are *isomorphic* if there exist isomorphisms $f_t: T \to T'$ and $f_p: \omega \to \omega$ such that $\partial'_i \circ f_t = (id_{\mathcal{N}m} + f_p) \circ \partial_i$, for $i = 0, 1$, and $(id_{\mathcal{N}m} + f_p)(\mu_0) = \mu'_0$.

Observe that nets that only differ for the use of names in ω are isomorphic, yielding a form of α-conversion. In the following, isomorphic nets will be regarded as equal. In particular, as we shall see, net isomorphism corresponds to structural congruence of join terms.

Definition 5. Let $N = (T, \partial_0, \partial_1, \mu_0)$ and $N' = (T', \partial'_0, \partial'_1, \mu'_0)$ be nets. Denoted by $\otimes$, *parallel composition* juxtaposes nets, merging public places without confusing the private ones. Formally,

$$N \otimes N' = (T + T', \partial_0 + \partial'_0, \partial_1 + \partial'_1, \mu_0 \oplus \mu'_0), \qquad \text{if } \omega(N) \cap \omega(N') = \varnothing,$$

where $\omega(N)$ denotes the names in ω that are used in N, i.e., that are either marked or connected to some transition. Observe that the side condition on private names can always be achieved up to isomorphism. Clearly, $\otimes$ is commutative and associative.

Denoted by ν_x, *restriction* 'hides' the place $x \in \mathcal{N}m$ replacing it by a *fresh* name $i \in \omega$, i.e., a name not occurring in $\omega(N)$. Formally, for $N = (T, \partial_0, \partial_1, \mu_0)$

$$\nu_x N = N[x \leftrightarrow i], \qquad i \text{ is fresh in } N$$

where $N[x \leftrightarrow i]$ is the net $(T, \partial'_0, \partial'_1, \mu'_0)$ whose μ'_0 and ∂'_j coincide respectively with μ_0 and ∂_j $(j = 0, 1)$, but for the values they yield on x and i, that are exchanged, i.e., for $\delta \in \{\partial_0, \partial_1, \mu_0\}$, we have that $\delta'(k)$ is equal to $\delta(i)$ if $k = x$, to $\delta(x)$ if $k = i$, and to $\delta(k)$ otherwise.

For $X = \{x_1, \ldots, x_n\} \subseteq \mathcal{N}m$, we use $\nu_X N$ to mean $\nu_{x_1}(\cdots \nu_{x_n} N)$. This is a correct definition because $\nu_x \nu_y N = \nu_y \nu_x N$ for each x, y, up to isomorphism.

The evolution of nets is described in terms of the 'firing' of its transitions. As usual, the firing of t consumes and produces resources, as prescribed by the pre- and post-set functions ∂_i. In the present setting, using $\otimes$ and ν_X to form net contexts, this can be expressed as the least *substitutive* relation $[\cdot\rangle$ such that

$$\mathbf{tran}\, \mu_0\, \mu_1 \otimes \mu_0 \; [\cdot\rangle \; \mathbf{tran}\, \mu_0\, \mu_1 \otimes \mu_1 .$$

This should be read as saying that, in any context, a marking matching the preset of a transition can be replaced by the associated post-set.

Observe that we stop at the single transition semantics of nets, rather than considering also the usual step semantics. We do so in order to match the standard reduction semantics of the join calculus, and it is not a hard limitation. We could easily extend the results to the classical step semantics, provided we equip join terms with multiple concurrent reductions.

The Type System Δ_0

The purpose of Δ_0 is to single out those join terms that are PT Petri nets. In order to achieve this we need to prevent any form of name passing and mobility, imposing a static network structure to terms. In particular, since definitions represent transitions, it is fundamental that their right-hand sides consist of messages only. Also, corresponding to the fact that there is only one atom (*token*) of information delivered in PT nets, we enforce that only empty messages may be exchanged. For notational convenience, we assume in the following the existence of a distinguished name $\bullet \in \mathcal{N}m$ that represents the empty tuple.

The types of Δ_0 are $\diamond$ and $\diamond\diamond$, with τ ranging over them. There are three kinds of type judgement:

$$\vdash P : \diamond\diamond \quad (P \text{ is ok and no } \mathtt{def}\ \_\ \mathtt{in}\ \_ \text{ occurs in it})$$
$$\vdash P : \diamond \quad (P \text{ is ok}) \qquad \vdash D : \diamond \quad (D \text{ is ok}).$$

The typing rules are the following.

$$\text{(P-Zero)}\ \vdash 0 : \diamond\diamond \qquad \text{(P-Mess)}\ \vdash x\langle\bullet\rangle : \diamond\diamond \qquad \text{(P-Par)}\ \frac{\vdash P : \tau \quad \vdash Q : \tau}{\vdash P \mid Q : \tau}$$

$$\text{(P-Def)}\ \frac{\vdash D : \diamond \quad \vdash P : \diamond}{\vdash \mathtt{def}\ D\ \mathtt{in}\ P : \diamond} \qquad \text{(P-Sub)}\ \frac{\vdash P : \diamond\diamond}{\vdash P : \diamond}$$

$$\text{(D-Pat)}\ \frac{\vdash P : \diamond\diamond}{\vdash J \triangleright P : \diamond}\ rn(J)=\{\bullet\} \qquad \text{(D-And)}\ \frac{\vdash D : \diamond \quad \vdash E : \diamond}{\vdash D \wedge E : \diamond}$$

The rules are elementary. The key is (D-Pat) that allows *only* simple processes as right-hand sides of definitions, while (P-Mess) ensures that only emissions of $\bullet$ are well typed and, together with (P-Par), that only parallel composition of messages can be used inside definitions. We use $\Delta_0 \vdash P$ as a shorthand for typeability, i.e., $\vdash P : \diamond$, in the system Δ_0 above.

Proposition 1 (Subject Reduction). *If $\Delta_0 \vdash P$ and $P \longrightarrow Q$, then $\Delta_0 \vdash Q$.*

Translating Δ_0 Terms to PT Nets

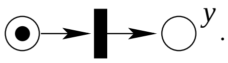
We intend to define a translation from Δ_0-typed join terms to PT nets that can be extended to other type systems and other classes of nets. Therefore, we first give a general map on all join terms, and then prove that it restricts to a well defined encoding of terms typeable in Δ_0 into PT nets. By induction on the structure of terms, $[\![\_]\!]$ is defined as follows.

$$\begin{array}{llll}
[\![0]\!] & = \varnothing & & \\
[\![x\langle v\rangle]\!] & = (x,v) & [\![J \triangleright P]\!] & = \mathbf{tran}[\![J]\!]\,[\![P]\!] \\
[\![P \mid Q]\!] & = [\![P]\!] \otimes [\![Q]\!] & [\![D \wedge E]\!] & = [\![D]\!] \otimes [\![E]\!] \\
[\![\mathtt{def}\ D\ \mathtt{in}\ P]\!] & = \nu_{dn(D)}([\![D]\!] \otimes [\![P]\!]) & &
\end{array}$$

where $[\![J]\!]$ is J seen as a multiset, i.e., $[\![x\langle v\rangle]\!] = (x,v)$ and $[\![J \mid K]\!] = [\![J]\!] \oplus [\![K]\!]$.

Roughly speaking, we map names to places, messages to markings, and definitions to (groups of) transitions. Since we shall soon consider nets whose tokens carry information, we use (x,v) to denote a marking of place x with value v. Of course, in the current case v always equals $\bullet$, and $(x,\bullet)$ should be read as an exotic way to mean x. In more details, $J \triangleright P$ is mapped into a transition whose pre-set is essentially J and whose post-set is the encoding of P, while $|$ and $\wedge$ are mapped homomorphically to $\otimes$. Similarly for $\texttt{def}\ D\ \texttt{in}\ P$, where the use of restriction ensures the correct treatment of local definitions. Here it is important to understand the different roles of $x\langle v\rangle$ because, depending on the context, $(x, \bullet)$ is treated as the net consisting of a single token in place x, or as an arc connecting place x to a transition. We shall get back to this in §4.

Example 1. Let P be the Δ_0-typeable term $\texttt{def}\ x\langle\bullet\rangle \triangleright y\langle\bullet\rangle\ \texttt{in}\ x\langle\bullet\rangle$. Then, using the standard representation of Petri nets enriched with labels that decorate *public* places with their name, the translation of P is

$$[\![P]\!] = \nu_x\big(\mathbf{tran}\{x\}\{y\} \otimes (x, \bullet)\big) =$$  .

Observe that, in force of the restriction ν_x, the name x does not appear.

As announced, the encoding is well defined on well-typed terms.

Proposition 2 (Correctness). *If $\Delta_0 \vdash P$, then $[\![P]\!]$ is a PT net.*

In addition to the previous proposition, we can prove a completeness theorem for Δ_0, expressed as follows.

Theorem 1 (Completeness). *The translation $[\![\_]\!]$ is an isomorphism between the set of terms typeable in Δ_0 and PT nets in Π_0.*

The rest of this section is devoted to illustrating that the $[\![\_]\!]$ is very tight a connection, preserving both structural congruence and dynamic behaviour.

Proposition 3. *Let P and Q be join terms typeable in Δ_0. Then,*

$$P \equiv Q \quad \textit{if and only if} \quad [\![P]\!] = [\![Q]\!].$$

Observe that there is a one-to-one correspondence between transitions in $[\![P]\!]$ and simple definitions $J \triangleright Q$ in P. Here and in the following we use $N\ [t\rangle\ N'$ to mean that N becomes N' by firing transitions t, and use $P \xrightarrow{t} Q$ to signify that P reduces to Q by means of the simple definition from which the transition t of $[\![P]\!]$ was generated.

Proposition 4. *Let P be a join term typeable in Δ_0. Then,*

$$P \xrightarrow{t} Q \quad \textit{if and only if} \quad [\![P]\!]\ [t\rangle\ [\![Q]\!]$$

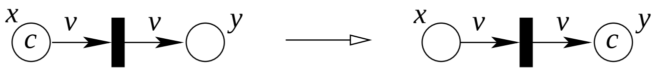
This implies that $P \longrightarrow^* Q$ if and only if there exist transitions $t_1, \ldots, t_n$ such that $[\![P]\!]\ [t_1\rangle \cdots [t_n\rangle\ [\![Q]\!]$. Also, P has a barb $x \in \mathit{fn}(P)$, i.e., $P \equiv \mathcal{C}[x\langle v\rangle]$ for $\mathcal{C}$ a context that does not bind x, if and only if x is marked in $[\![P]\!]$. Together with surjectivity and compositionality of $[\![\_]\!]$, this describes a very strong relationship, allowing to identify (the system represented by) P and (that represented by) $[\![P]\!]$. In other words, it allows us to identify typeability in Δ_0 with Π_0.

4 Coloured Nets

In coloured nets, tokens carry information: the 'colours'. Arcs between places and transitions are labelled by *arc expressions*, which evaluate to multisets of coloured tokens after binding free variables to colours. Expressions on input and output arcs describe respectively the resources needed for the firing and those generated by it. Although our treatment below attempts at providing an essential version of coloured Petri nets, we maintain one of their fundamental features. Namely, we permit a name to label several of the input arcs of a transition, with the intended meaning that firing is allowed only if the same colour can be fetched along all those arcs. This form of pattern matching is the reason why we relaxed the linearity requirement of join patterns. Technically, our model is obtained simply by allowing PT Nets to circulate tokens other than '•'. The obvious choice in the present context is to identify colours with names.

Definition 6 (CNets). A *coloured net* is a tuple $(T, \partial_0, \partial_1, \mu_0)$, where T is a finite set of transitions, $\partial_0, \partial_1 : T \to \mu(\mathcal{N}m_\omega \times \mathcal{N}m)$, are the pre and post-set functions, and $\mu_0 \in \mu(\mathcal{N}m_\omega \times \mathcal{N}m)$ is the initial marking. We use Π_1 to denote the class of coloured Petri nets.

Example 2. In the above definition we interpret the pairs (p, c) of $\mathcal{N}m_\omega \times \mathcal{N}m$ as representing a place p that carries a colour (name) c. The coloured net $\mathbf{tran}\{(x, v)\}\{(y, v)\} \otimes (x, c)$ is represented graphically as below on the left.



As usual, the role of names is twofold: in markings they represent colours, e.g., the constant c in x; in transitions they represent bound variables. The net above fetches any token from x, binds it to v, and then delivers the actual value of v to y, as illustrated by the picture above and formalised by the definition below.

Definition 7. The set of *received colours* of a marking $\mu \in \mu(\mathcal{N}m_\omega \times \mathcal{N}m)$ is

$$\mathtt{rc}(\mu) = \{x \in \mathcal{N}m \mid \; x \neq \bullet, \; \exists p. \, (p, x) \in \mu\}.$$

A *binding* for μ is a function $\mathtt{b} : \mathcal{N}m \to \mathcal{N}m$ which is the identity on $\mathcal{N}m \setminus \mathtt{rc}(\mu)$. We shall let $\mu\langle\mathtt{b}\rangle$ denote the multiset $\{(p, \mathtt{b}(c)) \mid (p, c) \in \mu\}$.

The firing rule for coloured nets is the *substitutive* relation generated by the reduction rule below, where $\mathtt{b}$ is a binding for μ_0

$$\mathbf{tran}\, \mu_0\, \mu_1 \otimes \mu_0\langle\mathtt{b}\rangle \; [\cdot\rangle \; \mathbf{tran}\, \mu_0\, \mu_1 \otimes \mu_1\langle\mathtt{b}\rangle.$$

Isomorphisms of coloured nets are the obvious extension of those of PT nets given in Definition 4. They allow renaming of private places, but map public ones identically. It is important to remark that coloured nets are considered up to α-conversion, that is renaming of received colours. In particular, coloured nets are isomorphic if there is an isomorphism between them after possibly renaming received names.

The Type System $\varDelta_1$

The purpose of type system $\varDelta_1$ is to characterise coloured Petri nets among join terms. Having introduced names, we now face the issue of distinguishing among two kind of names: channels and parameters. In fact, coloured nets are a strict value-passing formalism. They are not allowed to use values received along channels as channels themselves, nor to send private names as messages. In order to enforce this, we consider typing environments holding assumptions on free names, *viz.*, if they are channels or messages. Type environments are therefore pairs of *disjoint* sets $\varGamma$ (the channels) and ∇ (the messages). Type judgements are exactly as before, but for the presence of type environments.

(P-Zero)
$$\varGamma;\nabla \vdash 0 : \diamond\diamond$$

(P-Mess)
$$\varGamma;\nabla \vdash x\langle y\rangle : \diamond\diamond \quad \binom{x\notin\nabla}{y\notin\varGamma}$$

(P-Par)
$$\frac{\varGamma;\nabla \vdash P : \tau \quad \varGamma;\nabla \vdash Q : \tau}{\varGamma;\nabla \vdash P \mid Q : \tau}$$

(P-Def)
$$\frac{\varGamma, dn(D);\nabla \vdash D : \diamond \quad \varGamma, dn(D);\nabla \vdash P : \diamond}{\varGamma;\nabla \vdash \texttt{def}\ D\ \texttt{in}\ P : \diamond}$$

(P-Sub)
$$\frac{\varGamma;\nabla \vdash P : \diamond\diamond}{\varGamma;\nabla \vdash P : \diamond}$$

(D-Pat)
$$\frac{\varGamma;\nabla, rn(J) \vdash P : \diamond\diamond}{\varGamma;\nabla \vdash J \triangleright P : \diamond}$$

(D-And)
$$\frac{\varGamma;\nabla \vdash D : \diamond \quad \varGamma;\nabla \vdash E : \diamond}{\varGamma;\nabla \vdash D \wedge E : \diamond}$$

The structure of the rules matches exactly those of $\varDelta_0$, the only difference being the use of $\varGamma$ in (P-Def) and of ∇ in (D-Pat), so to be able to control the use of names in (P-Mess).

Proposition 5. *$\varDelta_0 \vdash P$ implies $\varDelta_1 \vdash P$.*

Proposition 6 (Subject Reduction). *If $\varDelta_1 \vdash P$ and $P \longrightarrow Q$, then $\varDelta_1 \vdash Q$.*

A simple inspection of the rules shows that, as $\varDelta_0$, system $\varDelta_1$ does not allow processes of the form `def _ in _` to appear inside definitions. The other fundamental properties of $\varDelta_1$ are expressed in the proposition below.

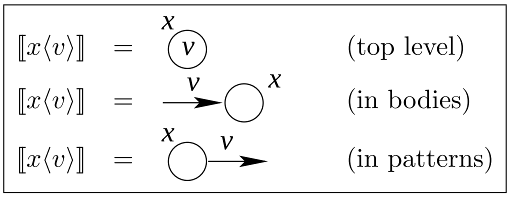
Proposition 7. *If $\varDelta_1 \vdash P$, then P never emits messages on received names nor emits bound names out of the scope of their definitions.*

Translating $\varDelta_1$ Terms to CNets

It follows as a consequence of Proposition 7 that the translation $[\![\_]\!]$ extends to a well defined map from $\varDelta_1$ typeable terms to coloured Petri nets.

Proposition 8 (Correctness). *If $\varDelta_1 \vdash P$, then $[\![P]\!]$ is a coloured net.*

Here more than before it is important to understand the three different roles played by $x\langle v\rangle$ in the translation. First, when $x\langle v\rangle$ appears outside all definitions, at top level, it is translated as the net consisting of a token v in the place x. Secondly, when it appears in a join pattern J, it is translated in $[\![J]\!]$ as (x, v), representing the input from place x on a bound variable v. Finally, when $x\langle v\rangle$ appears in the body of a definition, it is translated as (x, v) by the same clause above but, considered as a marking, it represents the output of a free or bound variables in the place x. This is summarised by the pictures below.


$[\![x\langle v\rangle]\!]$ = x v (top level)
$[\![x\langle v\rangle]\!]$ = v x (in bodies)
$[\![x\langle v\rangle]\!]$ = x v (in patterns)

Example 3. Let P be the Δ_1 typeable term $\texttt{def}\ x\langle v\rangle \triangleright y\langle v\rangle\ \texttt{in}\ x\langle c\rangle$. Then $[\![P]\!]$ is the coloured net of Example 2.

Our results about Δ_1 match exactly those for Δ_0. They are listed below.

Theorem 2 (Δ_1 vs Π_1). *The translation $[\![\_]\!]$ is an isomorphism between the set of terms typeable in Δ_1 and coloured nets in Π_1. Moreover, if P and Q are typeable in Δ_1, then*

$$\begin{aligned} P \equiv Q \quad &\textit{if and only if} \quad [\![P]\!] = [\![Q]\!] \\ P \xrightarrow{t} Q \quad &\textit{if and only if} \quad [\![P]\!]\ [t\rangle\ [\![Q]\!] \end{aligned}$$

5 Reconfigurable Nets

Coloured Petri nets have a static structure, i.e., their firings only affects the marking. While this accounts faithfully for value-passing theories, it is completely inadequate to represent dynamically changing structures. Reconfigurable nets generalise coloured ones by adding precisely one ingredient: each firing of a transition may deliver to a different set of output places. This equips nets with a mechanism to model networks with reconfigurable topologies, that is networks in which the set of components is fixed, but the connectivity among them may change in time. Formally, this is achieved by a very smooth alteration of the definitions of coloured nets – one that allows markings and messages to contain private names – and of their firing rule.

Definition 8 (RNets). A *reconfigurable net* is a tuple $(T, \partial_0, \partial_1, \mu_0)$, where T is a finite set of transitions, $\partial_0 \colon T \to \mu(\mathcal{N}m_\omega \times \mathcal{N}m)$ and $\partial_1 \colon T \to \mu(\mathcal{N}m_\omega \times \mathcal{N}m_\omega)$, are, respectively, the pre and post-set functions, and $\mu_0 \in \mu(\mathcal{N}m_\omega \times \mathcal{N}m_\omega)$ is the initial marking. We use Π_2 to denote the set of reconfigurable Petri nets.

Restriction ν_x and parallel composition $\otimes$ extends straightforwardly to reconfigurable nets. Also, isomorphisms of reconfigurable Petri nets are obtained extending those of coloured nets in the obvious way.

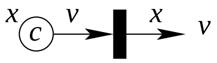
The firing rule is generalised by extending the scope of the binding of received names to include the output places in addition to the names on the output arcs.

Definition 9. Let $\mathtt{b}$ be a binding. For $\mu \in \mu(\mathcal{N}m_\omega \times \mathcal{N}m_\omega)$, let $X\langle\!\langle \mathtt{b} \rangle\!\rangle$ denote the multiset $\{(\mathtt{b}_\omega(p), \mathtt{b}_\omega(c)) \mid (p,c) \in \mu\}$, where $\mathtt{b}_\omega = \mathtt{b} + id_\omega$.

For $\mathtt{b}$ a binding for μ_0, the firing is generated closing by net contexts the rule

$$\mathbf{tran}\, \mu_0\, \mu_1 \otimes \mu_0\langle \mathtt{b} \rangle \;[\cdot\rangle\; \mathbf{tran}\, \mu_0\, \mu_1 \otimes \mu_1\langle\!\langle \mathtt{b} \rangle\!\rangle .$$

Example 4. The simple reconfigurable net represented below binds v to any name found in x – in this case c – and delivers the name x to the place to which v is bound.



The Type System Δ_2

In order to design a type system Δ_2 corresponding to Π_2 we only need to remove the difference between channels and messages upon which Δ_1 is based, which can be done easily by removing Γ and ∇. This yields a system identical to Δ_0, apart from (P-MESS) that allows any message to be transmitted on any name.

(P-ZERO) $\vdash 0 : \diamond\diamond$ (P-MESS) $\vdash x\langle y \rangle : \diamond\diamond$ (P-PAR) $$\frac{\vdash P : \tau \quad \vdash Q : \tau}{\vdash P \mid Q : \tau}$$

(P-DEF) $$\frac{\vdash D : \diamond \quad \vdash P : \diamond}{\vdash \mathtt{def}\ D\ \mathtt{in}\ P : \diamond}$$ (P-SUB) $$\frac{\vdash P : \diamond\diamond}{\vdash P : \diamond}$$

(D-PAT) $$\frac{\vdash P : \diamond\diamond}{\vdash J \rhd P : \diamond}$$ (D-AND) $$\frac{\vdash D : \diamond \quad \vdash E : \diamond}{\vdash D \wedge E : \diamond}$$

Proposition 9 (Subject Reduction). *If $\Delta_2 \vdash P$ and $P \longrightarrow Q$, then $\Delta_2 \vdash Q$.*

Proposition 10. *$\Delta_1 \vdash P$ implies $\Delta_2 \vdash P$.*

Translating Δ_2 Terms to RNets

The property guaranteed by Δ_2 is that there is no process of the kind `def _ in _` inside definitions, i.e., the topology of the net may change by redirecting output edges, but the components of the net are fixed once and for all. In force of this, we can extend to Δ_2 the correspondence between well-typed terms and nets.

Theorem 3 (Δ_2 vs Π_2). *The translation $[\![\_]\!]$ is an isomorphism between the set of terms typeable in Δ_2 and reconfigurable nets in Π_2. Moreover, if P and Q are typeable in Δ_2, then*

$$\begin{aligned} P \equiv Q &\quad \textit{if and only if} \quad [\![P]\!] = [\![Q]\!] \\ P \xrightarrow{\;t\;} Q &\quad \textit{if and only if} \quad [\![P]\!]\, [t\rangle\, [\![Q]\!] \end{aligned}$$

6 Dynamic Nets

The obvious generalisation of reconfigurable nets is to allow the dynamic creation of components, that we achieve by means of the notions of *dynamic Petri nets*. The idea behind such structures is that the firing of a transition allocates a new net parametric in the actual values of the received names. As the net may consists simply of a marking and no transitions, this includes the standard definition of PT nets. Also coloured and reconfigurable nets are, of course, special kinds of dynamic nets. The characteristic feature of dynamic nets, that to our best knowledge distinguishes them by other approaches in the literature and draws a connection to [13], is that, as for reconfigurable nets, input arcs are never modified. While it is possible to modify dynamically the post-set of a transition, and also to spawn new subnets, it is not possible to add places to the presets of transitions. This allows us to formalise our intuition by simply generalising the post-set functions of nets, allowing $\partial_1(t)$ to be a dynamic net. Of course, this means that the definition of nets becomes recursive.

Definition 10 (DNets). Let *DNets* be the least set satisfying the equation

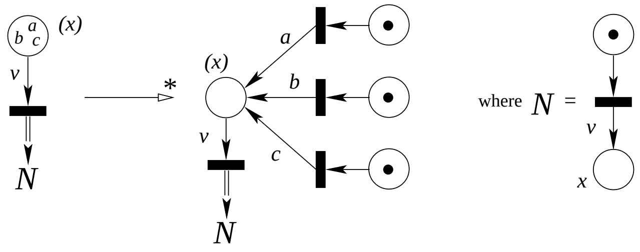
$$\mathit{DNets} \cong \big\{(T, \partial_0 \colon T \to \mu(\mathcal{N}m_\omega \times \mathcal{N}m), \partial_1 \colon T \to \mathit{DNets}, \mu_0)\big\},$$

where T is a finite set and μ_0, the initial marking, is in $\mu(\mathcal{N}m_\omega \times \mathcal{N}m_\omega)$. A *dynamic* net *DN* is an element of *DNets*. We use Π_3 to refer to the set of dynamic nets.

Once again, the operations of parallel composition and restriction, and the notions of α-conversion and isomorphism lift smoothly to this setting.

Definition 11. Dynamic nets $(T, \partial_0, \partial_1, \mu_0)$ and $(T', \partial'_0, \partial'_1, \mu'_0)$ are isomorphic if, up to α-conversion, there exists a pair $(f_t \colon T \to T', f_p \colon \omega \to \omega)$ of isomorphisms such that $(id_{\mathcal{N}m} + f_p \times id_{\mathcal{N}m} + f_p)(\mu_0) = \mu'_0$, $\partial'_0 \circ f_t = (id_{\mathcal{N}m} + f_p \times id_{\mathcal{N}m}) \circ \partial_0$, and, recursively, $\partial_1(t)$ and $\partial'_1(f_t(t))$ are isomorphic for each $t \in T$.

Example 5. For $P = \texttt{def}\ x\langle v\rangle \triangleright (\texttt{def}\ y\langle\bullet\rangle \triangleright x\langle v\rangle\ \texttt{in}\ y\langle\bullet\rangle)\ \texttt{in}\ x\langle a\rangle \mid x\langle b\rangle \mid x\langle c\rangle$, the structure $[\![P]\!]$ is the dynamic net illustrated below together with one of its possible firing sequences. The transition may fire three times, binding v in turn to a, b, and c, and spawning three new nets instantiating v appropriately in N. The free name x of N maintains its meaning across instantiations. This rules are formalised below, and amounts to a recursive extension of binding.



Definition 12. Let $\mathtt{b}$ be a binding. For $X = (T, \partial_0, \partial_1, \mu_0) \in \mathit{DNets}$, let $X\langle\!\langle\!\langle \mathtt{b} \rangle\!\rangle\!\rangle$ denote the dynamic net obtained from X by substituting free names according to $\mathtt{b}$ in μ_0, $\partial_0(t)$ and, recursively, applying $\langle\!\langle\!\langle \mathtt{b} \rangle\!\rangle\!\rangle$ to $\partial_1(t)$, for all $t \in T$.

The firing for dynamic nets is generated as follows, for $\mathtt{b}$ a binding for μ_0.

$$\mathbf{tran}\,\mu_0\, X \otimes \mu_0\langle \mathtt{b} \rangle\ [\cdot\rangle\ \mathbf{tran}\,\mu_0\, X \otimes X\langle\!\langle\!\langle \mathtt{b} \rangle\!\rangle\!\rangle$$

The Type System Δ_3

In order to capture dynamic nets in terms of the join calculus, we simply have to relax the restriction that disallows definitions inside definitions. At the level of type system, this can be achieved by removing the distinction between types $\diamond$ and $\diamond\diamond$ from Δ_2. As an obvious consequence, any join term results typeable.

Proposition 11. *For each join term P, $\Delta_3 \vdash P$.*

Thus, extending our results for $[\![\_]\!]$ to Δ_3 amounts to proving that dynamic nets coincide with join terms, which is the content of the following.

Theorem 4 (Δ_3 vs Π_3). *The translation $[\![\_]\!]$ is an isomorphism between join terms (terms typeable in Δ_3) and dynamic nets in Π_3. Moreover, if P and Q are typeable in Δ_3, then*

$$\begin{array}{rcl} P \equiv Q & \textit{if and only if} & [\![P]\!] = [\![Q]\!] \\ P \ {-t\!\!\rightarrow}\ Q & \textit{if and only if} & [\![P]\!]\ [t\rangle\ [\![Q]\!] \end{array}$$

7 The Type System Δ_4

The information contained in Δ_i, $i = 0, \ldots, 3$ can be summarised in a single type system Δ_4 that tags terms with indices that characterise them as nets of Π_i.

Types range over $i^{\diamond\diamond} \in \{0^{\diamond\diamond}, 1^{\diamond\diamond}, 2^{\diamond\diamond}, 3^{\diamond\diamond}\}$ and $i^{\diamond} \in \{0^{\diamond}, 1^{\diamond}, 2^{\diamond}, 3^{\diamond}\}$. For $\tau = i^{?}$ a type, we write $\tau\!\downarrow$ for i and $\tau\!\uparrow$ for ?. By $\tau \leq \tau'$ we mean that $\tau\!\downarrow \leq \tau'\!\downarrow$ and $\tau\!\uparrow \leq \tau'\!\uparrow$, with the convention that $\diamond\diamond \leq \diamond$. Type environments are pairs $\Gamma; \nabla$ as for Δ_1, and type judgements are as follows:

$$\begin{array}{ll}
\Gamma; \nabla \vdash P : i^{\diamond\diamond} & (P \text{ well-typed, in } \Pi_i, \text{ and containing no } \mathtt{def}\ \_\ \mathtt{in}\ \_) \\
\Gamma; \nabla \vdash P : i^{\diamond} & (P \text{ well-typed and in } \Pi_i) \\
\Gamma; \nabla \vdash D : i^{\diamond} & (D \text{ well-typed and containing terms in } \Pi_i)
\end{array}$$

The typing rules are the following.

(P-Mess$_0$)
$$\Gamma; \nabla \vdash x\langle \bullet \rangle : 0^{\diamond\diamond} \ (x \notin \nabla)$$

(P-Mess$_1$)
$$\Gamma; \nabla \vdash x\langle y \rangle : 1^{\diamond\diamond} \ \begin{pmatrix} x \notin \nabla \\ y \notin \Gamma \end{pmatrix}$$

(P-Mess$_2$)
$$\Gamma; \nabla \vdash x\langle y \rangle : 2^{\diamond\diamond}$$

(P-Def)
$$\frac{\Gamma, dn(D); \nabla \vdash D : i^{\diamond} \quad \Gamma, dn(D); \nabla \vdash P : i^{\diamond}}{\Gamma; \nabla \vdash \mathtt{def}\ D\ \mathtt{in}\ P : i^{\diamond}}$$

(P-Zero)
$$\Gamma; \nabla \vdash 0 : 0^{\diamond\diamond}$$

(P-Par)
$$\frac{\Gamma; \nabla \vdash P : \tau \quad \Gamma; \nabla \vdash Q : \tau}{\Gamma; \nabla \vdash P \mid Q : \tau}$$

(P-Sub)
$$\frac{\Gamma; \nabla \vdash P : \tau \quad \tau \leq \tau'}{\Gamma; \nabla \vdash P : \tau'}$$

(D-Patt$_0$)
$$\frac{\Gamma; \nabla \vdash P : 0^{\diamond\diamond}}{\Gamma; \nabla \vdash J \triangleright P : 0^{\diamond}} \ (rn(J) = \{\bullet\})$$

(D-Patt$_1$)
$$\frac{\Gamma; \nabla, rn(J) \vdash P : i^{\diamond\diamond}}{\Gamma; \nabla \vdash J \triangleright P : i^{\diamond}}$$

(D-Patt$_2$)
$$\frac{\Gamma; \nabla, rn(J) \vdash P : i^{\diamond}}{\Gamma; \nabla \vdash J \triangleright P : 3^{\diamond}}$$

(D-And)
$$\frac{\Gamma; \nabla \vdash D : i^{\diamond} \quad \Gamma; \nabla \vdash E : i^{\diamond}}{\Gamma; \nabla \vdash D \wedge E : i^{\diamond}}$$

(D-Sub)
$$\frac{\Gamma; \nabla \vdash D : i^{\diamond} \quad i < j}{\Gamma; \nabla \vdash D : j^{\diamond}}$$

Here (P-Mess) is split into three rules, each behaving as in the corresponding Δ_i; (P-Mess$_2$) ignores type environments, so achieving the effect of Δ_2. Definitions can be typed by i if the terms they contain are typeable by i; (D-Patt$_2$) constrains to 3 the type of definitions containing a `def _ in _` term, so forcing to 3 the type of enclosing processes by means of rule (P-Def). For Δ_4 we have the following results.

Proposition 12 (Subject Reduction). *If $\Delta_4 \vdash P$ and $P \longrightarrow Q$, then $\Delta_4 \vdash Q$.*

Theorem 5. *$\Delta_4 \vdash P : i^{\diamond}$ if and only if $P \in \Pi_i$, for $i = 0, \ldots, 3$.*

Proof. It follows by proving that $\Delta_4 \vdash P : i^{\diamond}$ if and only if $\Delta_i \vdash P$.

Conclusions and Future Work

We have provided a full correspondence between join calculus and a hierarchy of Petri net classes. It would be interesting to study relevance and adaptability to coloured nets of the existing polymorphic type systems [8,15] and extensional semantic equivalence [10,3,6] for join terms. On the other hand, the body of work on the semantics of nets suggests a noninterleaving semantics based on monoidal categories for the join-calculus.

Our version of coloured nets simplifies considerably those in the literature. It is worth investigating whether it can be interesting for the coloured Petri net community by putting it at work on suitable applications.

Also, we plan to compare our nets to Milner's named nets [12] – clearly closely related to the nets of §3 – to Asperti and Busi's dynamic nets [1], and to Odersky's functional nets [14].

Acknowledgements. We heartily thank Cedric Fournet, Alan Schmitt, and Emilio Tuosto for their useful comments. Special thanks to Roberto Bruni, who made substantial suggestions, and to an anonymous referee, who spotted some technical flaws.

References

1. A. Asperti and N. Busi (1996), *Mobile Petri Nets*, Technical Report UBLCS 96-10, Università di Bologna.
2. G. Berry and G. Boudol (1992), The Chemical Abstract Machine, *Theoretical Computer Science*, 96:217–248.
3. M. Boreale, C. Fournet, and C. Laneve (1998), Bisimulations for the Join-Calculus, In *Proc. PROCOMET'98*, D. Gries and W.P. de Roever (Eds.), 68–86, IFIP, Chapman & Halls.
4. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone (2000), Functorial Models for Petri Nets, *Information and Computation.* To appear.
5. C. Fournet and G. Gonthier (1996), The Reflexive Chemical Abstract Machine and the Join-Calculus, In *Proc. POPL'96*, 372–385, ACM.
6. C. Fournet and G. Gonthier (1996), A Hierarchy of Equivalences for Asynchronous Calculi, In *Proc. ICALP'98*, Lecture Notes in Computer Science 1443, 844–855, Springer.
7. C. Fournet, G. Gonthier, J. Lévy, L. Maranget, and D. Rémy (1996), A Calculus of Mobile Agents, In *Proc. CONCUR'96*, Lecture Notes in Computer Science 1119, 406–421, Springer.
8. C. Fournet, C. Laneve, L. Maranget, and D. Rémy (1997), Implicit Typing à la ML for the Join-Calculus, In *Proc. CONCUR'97*, Lecture Notes in Computer Science 1243, 196–212, Springer.
9. K. Jensen (1992), *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* Monographs on Theoretical Computer Science, Springer.
10. C. Laneve (1996), May and Must Testing in the Join-Calculus, Technical Report UBLCS 96-04, Università di Bologna.
11. F. Le Fessant and L. Maranget (1998), Compiling Join-Patterns, In *Proc. HLCL'98*, Electronic Notes in Computer Science 16(3), Elsevier.

12. R. Milner (1993), Action Calculi, or Syntactic Action Structures, In *Proc. MFCS'93*, Lecture Notes in Computer Science 711, 105-121, Springer.
13. M. Nielsen, L. Priese, and V. Sassone (1995), Characterizing Behavioural Congruences for Petri Nets, in *Proc. CONCUR 95*, Lecture Notes in Computer Science 962, 175–189, Springer.
14. M. Odersky (2000), Functional Nets, In *Proc. ESOP'2000*, Lecture Notes in Computer Science 1782, 1–25, Springer.
15. M. Odersky, C. Zenger, M. Zenger, and G. Chen (1999), A Functional View of Join, Technical Report ACRC-99-016, University of South Australia.
16. C.A. Petri (1962), *Kommunikation mit Automaten.* Ph.D. thesis, Institut für Instrumentelle Mathematik, Bonn.
17. W. Reisig (1985), *Petri Nets: An Introduction.* EATCS Monographs on Theoretical Computer Science, Springer.
18. V. Sassone (2000), On the Algebraic Structure of Petri Nets, *Bulletin of EATCS* 72, 133–148.