Distributed Conflicts in Communicating Systems

Nadia Busi **Roberto Gorrieri**

Technical Report UBLCS-94-8

April 1994
Revised February 1995

Department of Computer Science
University of Bologna
Piazza di Porta S. Donato, 5
40127 Bologna (Italy)

The University of Bologna Department of Computer Science Research Technical Reports are available in gzipped PostScript format via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` or via WWW at URL `http://www.cs.unibo.it/`. Plain-text abstracts organized by year are available in the directory `ABSTRACTS`. All local authors can be reached via e-mail at the address *last-name*`@cs.unibo.it`. Written requests and comments should be addressed to `tr-admin@cs.unibo.it`.

UBLCS Technical Report Series

93-23 *A Scalable Architecture for Reliable Distributed Multimedia Applications*, F. Panzieri, M. Roccetti, October 1993.

93-24 *Wide-Area Distribution Issues in Hypertext Systems*, C. Maioli, S. Sola, F. Vitali, October 1993.

93-25 *On Relating Some Models for Concurrency*, P. Degano, R. Gorrieri, S. Vigna, October 1993.

93-26 *Axiomatising ST Bisimulation Equivalence*, N. Busi, R. van Glabbeek, R. Gorrieri, December 1993.

93-27 *A Theory of Processes with Durational Actions*, R. Gorrieri, M. Roccetti, E. Stancampiano, December 1993.

94-1 *Further Modifications to the Dexter Hypertext Reference Model: a Proposal*, W. Penzo, S. Sola, F. Vitali, January 1994.

94-2 *Symbol-Level Requirements for Agent-Level Programming*, M. Gaspari, E. Motta, February 1994.

94-3 *Extending Prolog with Data Driven Rules*, M. Gaspari, February 1994.

94-4 *Schedulability Checking of Data Flow Tasks in Hard-Real-Time Distributed Systems*, R. Davoli, L. A. Giachini, March 1994.

94-5 *A Shared Dataspace Language and its Compilation*, M. Gaspari, March 1994.

94-6 *Exploring the Coordination Space with LO*, S. Castellani, P. Ciancarini, April 1994.

94-7 *Comparative Semantics of LO*, S. Castellani, P. Ciancarini, April 1994.

94-8 *Distributed Conflicts in Communicating Systems*, N. Busi, R. Gorrieri, April 1994 (Revised February 1995).

94-9 *Pseudopolar Array Mask Algorithm for Spherical Coordinate Grid Surfaces*, A. Amoroso, G. Casciola, May 1994.

94-10 *MPA: a Stochastic Process Algebra*, M. Bernardo, L. Donatiello, R. Gorrieri, May 1994.

94-11 *Describing Queueing Systems with MPA*, M. Bernardo, L. Donatiello, R. Gorrieri, May 1994.

94-12 *Operational GSPN Semantics of MPA*, M. Bernardo, L. Donatiello, R. Gorrieri, May 1994.

94-13 *Experiments in Distributing and Coordinating Knowledge*, P. Ciancarini, May 1994.

94-14 *A Comparison of Parallel Search Algorithms Based on Tree Splitting*, P. Ciancarini, May 1994.

94-15 RELACS: *A Communications Infrastructure for Constructing Reliable Applications in Large-Scale Distributed Systems*, Ö. Babaoğlu, M.G. Baker, R. Davoli, L.A. Giachini, June 1994.

94-16 *Replicated File Management in Large-Scale Distributed Systems*, Ö. Babaoğlu, A. Bartoli, G. Dini, June 1994.

94-17 *Parallel Symbolic Computing with the Shared Dataspace Coordination Model*, P. Ciancarini, M. Gaspari, July 1994.

94-18 *An Algorithmic Method to Build Good Training Sets for Neural-Network Classifiers*, F. Tamburini, R. Davoli, July 1994.

94-19 *On Group Communication in Large-Scale Distributed Systems*, Ö. Babaoğlu, A. Schiper, July 1994.

94-20 *Dynamic Allocation of Signature Files in Multiple-Disk Systems*, P. Ciaccia, August 1994.

94-21 *Parallel Independent Grid Files Based on a Dynamic Declustering Method Using Multiple Error Correcting Codes*, P. Ciaccia, November 1994.

95-1 *Performance Preorder and Competitive Equivalence*, F. Corradini, R. Gorrieri, M. Roccetti, January 1995.

95-2 *Clepsydra Methodology*, P. Ciaccia, O. Ciancarini, W. Penzo, January 1995.

95-3 *Debugging Distributed Executions Using Language Recognition*, Ö. Babaoğlu, E. Fromentin, M. Raynal, January 1995.

95-4 *Effective Applicative Structures*, A. Asperti, A. Ciabattoni, January 1995.

95-5 *An Open Framework for Cooperative Problem Solving*, M. Gaspari, E. Motta, A. Stutt, February 1995.

95-6 *Considering New Guidelines in Group Interface Design: a Group-Friendly Interface for the CHAOS System*, D. Bottura, C. Maioli, S. Mangiaracina, February 1995.

95-7 *Modelling Interaction in Agent Systems*, A. Dalmonte, M. Gaspari, February 1995.

Distributed Conflicts in Communicating Systems[1]

Nadia Busi[2] **Roberto Gorrieri**[3]

Technical Report UBLCS-94-8

April 1994
Revised February 1995

Abstract

We argue that the alternative composition operator of CCS not only lacks expressiveness, but also provides a too abstract description of conflicting activities. Hence, we propose to replace it with a unary conflict operator and a conflict restriction operator, yielding the process algebra DiX. We show that DiX is a semantic extension of CCS. Moreover, DiX is equipped with a simple distributed semantics defined in terms of nets with inhibitor arcs, where the set of transitions is generated by three axiom schemata only. This net semantics is the main motivation for the present proposal.

1. In *Proceedings of the pre-ECOOP'94 Workshop on Models and Languages for Coordination of Parallelism and Distribution*, Bologna (Italy), July 1994. To appear in LNCS, Springer. Research partially supported by EC BRA n. 9102 COORDINATION.
2. Dipartimento di Matematica, Università di Siena, Via del Capitano 15, I-53100 Siena, Italy.
3. Dipartimento di Matematica, Università di Bologna, Piazza di Porta S. Donato 5, I-40127 Bologna, Italy

1 Introduction

The operator of alternative composition, denoted by +, is one of the most usual operators in process algebras. Its success probably depends on the intuitively clear and simple mathematical description it receives in interleaving semantics. Unfortunately, as soon as one is interested in causality based semantics, one immediately discovers a lot of intriguing problems concerning +.

These are mainly due to the fact that an execution of $E_1 + E_2$ consists of two different activities: the choice of the process to be executed (say E_1) and the performance of the first action of the selected E_1. In order to preserve correctness, the foregoing two operations must be performed atomically [18]. Hence, troubles arise for its distributed-choice net-based modeling [9, 10, 12, 14, 27], centralized-choice net-based description [16, 17] and CHAM-like semantics [2].

The interleaving description of + gives a very abstract representation of "rivalry" because in a distributed environment the competition between two activities a and b, represented by $a + b$, is usually not resolved *before* the actions start, rather it is resolved *by* the completion of one of these actions. In other words, a and b both start their execution and the choice is made by the action which ends first, hence preventing the completion of the other one (*a posteriori* choice). Thus, in our opinion, $a + b$ should be replaced by the parallel composition $a|b$, enhanced with some mechanism for resolving the rivalry.

In this paper we replace + with a more flexible and expressive conflict mechanism. This is achieved by marking processes with conflicts. That is, given a process p and a set I of conflicts, we write Ip to represent the process which can perform the same actions of p. The evaluation of Ip causes the "broadcasting" of the conflicts in I to the other processes. Moreover, a conflict restriction operator $p \backslash n$ specifies that the conflict name n is local to p, hence the broadcasting of n is confined inside p.

Clearly, the execution of actions of Ip may influence the execution of other processes in several ways. Here, we choose a simple discipline: when performing an action, Ip "kills" all its competitors, i.e., all the other processes which possess "contrasting" conflict names. This choice is for simplicity sake, in order to keep simple the process algebra – we call DiX (Distributed Conflicts in Communicating Systems: DiCCS = DiX) – which implements the alternative composition of CCS. To be more precise, the CCS expression $a + b$ is translated into the DiX term $(\{1\}a|\{\bar{1}\}b)\backslash 1$, where $\bar{1}$ is the contrasting conflict of 1 and the external conflict restriction makes 1 local to the inner parallel process. The rules for parallel composition have the duty of broadcasting the conflict set I of the active subagent, thus possibly killing some of the other processes. Note that when the above process performs an action, e.g. a, the elimination of the conflicting subprocess, namely $\{\bar{1}\}b$, is done after the completion of a.

We provide DiX with an interleaving semantics, proving that strong bisimulation is a congruence. Then, we define the embedding of CCS in DiX, which is sound and complete w.r.t. strong bisimulation.

Then we show that DiX has a simple distributed description in terms of Place/Transition nets *with inhibitor arcs*. As a byproduct, we obtain a net semantics for CCS, which implements nondeterminism and restriction by means of inhibitor arcs. In particular,

(i) every conflict name n is represented by a place in the net. This place inhibits all the transitions of the (sub)processes having $\bar{n}$ in its conflict set;

(ii) every restriction $\backslash a$ is modelled by a place on the net inhibiting all the transitions labelled by a.

Hence, this net semantics moves the focus from broadcasting information (as in the interleaving case) to "asking" information whenever needed, i.e., checking whether the transition is not inhibited by any conflict or restriction it might be subject to. The semantics gains in simplicity because

(i) the set of net transitions is generated by only three axiom schemata;

(ii) it gives a description of distributed choice which is linear in the size of the components.

We prove that the interleaving marking graph associated to the net for DiX tightly corresponds to DiX interleaving transition system. We also define the multistep marking graph associated to the net for DiX.

2 The Language

Let $\mathcal{A}$ be a denumerable set of names and $a, b, c, \ldots$ range over $\mathcal{A}$. Let $\bar{\mathcal{A}} = \{\bar{a} \mid a \in \mathcal{A}\}$ be the set of conames. Let $Act = \mathcal{A} \cup \bar{\mathcal{A}} \cup \{\tau\}$ be the set of *actions*, ranged over by α, where $\tau \notin \mathcal{A}$ is the *internal* action. Let λ range over $\mathcal{A} \cup \bar{\mathcal{A}}$, with the assumption that $\bar{\bar{\lambda}} = \lambda$.

Let $\mathcal{C}$ be a denumerable set of symbols disjoint from $\mathcal{A} \cup \{\tau\}$. In the following we will always take the set $\mathbb{N}$ of natural numbers as $\mathcal{C}$. Let $\bar{\mathcal{C}} = \{\bar{n} \mid n \in \mathcal{C}\}$. A *conflict* is any element of $Con = \mathcal{C} \cup \bar{\mathcal{C}}$, ranged over by n (we assume again that $\bar{\bar{n}} = n$). Finite sets of conflicts will be denoted by $I, J, H, \ldots$. We define $\bar{I} = \{\bar{n} \mid n \in I\}$. A conflict set I is *consistent* if and only if $I \cap \bar{I} = \emptyset$. I and J *overlap* when $I \cap J \neq \emptyset$.

The process terms are those generated by the following grammar:

$$p ::= nil \mid \alpha.p \mid Ip \mid p|p \mid p\backslash a \mid p\backslash n \mid x \mid \mathrm{rec}\, x.p$$

where the main difference w.r.t. CCS is that + is missing (as well as relabeling; see the concluding remarks for a discussion of this issue), and two new operators, Ip and $p\backslash n$, have been added. The set $\mathcal{P}$ of DiX agents are those process terms which are *closed* (w.r.t. recursion variables) and (strongly) *guarded* [23]. $\mathcal{P}$ will be ranged over by p and q. The set of the action names occurring in p is denoted by $Sort_a(p)$, while $Sort_c(p)$ is the set of the conflict names in p. For notational convenience, we often omit the terminal nil; e.g., a stands for $a.nil$.

Let us explain informally the meaning of Ip and $p\backslash n$ (we omit the discussion about the other operators, since they are well–known). With a set of conflicts I and a process p, we construct a new process Ip. We say that Ip is *self-conflicting* if I is not consistent. If I and $\bar{J}$ overlap, we say that Ip and Jq are conflicting processes. A consequence of the execution of any of the actions of Ip is the elimination of all the processes that are in conflict with it. When I is a singleton, we drop the set brackets for notational convenience; e.g., $1p$ stands for $\{1\}p$. The conflict restriction operator $p\backslash n$ is used to make the conflict n local to p. Sometimes we use the convenient shorthand expression $p\backslash I$ to say that p is restricted w.r.t. all the conflict names in I.

3 The Operational Semantics

The operational semantics for DiX is a labeled transition system $LTS_{DiX} = (\mathcal{P}, \mathcal{L}, \rightarrow)$, where $\mathcal{L} = Act \times \wp_{fin}(Con)$ and the transition relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{L} \times \mathcal{P}$ is defined as the least relation generated by the set of inference rules listed in Table 1.

In these rules we exploit the auxiliary operator $\mathsf{kill}(p,I)$, whose definition is listed in Table 2. The operation $\mathsf{kill}(p,I)$ is used to propagate the conflict information of a transition to the whole agent and to erase all those subterms having conflict set overlapping with $\bar{I}$. This effect is the elimination of each subterm of p with the form Jq if the set I contains the opposite of some element in J. Remark that $\mathsf{kill}(p,I)$ also eliminates from p every conflict which is contained in I. Indeed these conflicts are superfluous since every subprocess Jq with $I \cap \bar{J} \neq \emptyset$ is erased.

With $LTS_{DiX}(p)$ we denote the transition system rooted at the initial state p. The transition $p \xrightarrow{\alpha,I} q$ means that the process p may perform α and broadcast the conflicts I, and then behaves like the process q.

By (act), the process $\alpha.p$ evolves to p with a label $\alpha, \emptyset$.

The rule (conf) describes the behavior of the conflict operator. If p performs an action α with a set of conflicts I, then Jp must declare that the new set of "killing" conflicts for the other processes is now $I \cup J$ and also must terminate all its subagents which are in conflict with J. For example, when the process $1a.\bar{1}b.nil$ performs the action a, its subagent $\bar{1}b.nil$ must be killed: we have $1a.\bar{1}b.nil \xrightarrow{a,\{1\}} nil$. Moreover, an action cannot take place if it is performed by a subterm which is in conflict with the agent in which it is contained. For example, in the agent $1(\bar{1}a.nil)$ the action a can never take place. This constraint is imposed by requiring that Jp is not self-conflicting and that I and $\bar{J}$ are not overlapping; these are expressed by the side-condition $(I \cup J) \cap \bar{J} = \emptyset$. These two examples show that we can express even the form of conflict of flow event structures [3], where the flow relation and the conflict relation are not required to be separate.

Table 1. Operational rules

(act)	$\alpha.p \xrightarrow{\alpha,\emptyset} p$	
(conf)	$\dfrac{p \xrightarrow{\alpha,I} p'}{Jp \xrightarrow{\alpha,I\cup J} \mathsf{kill}(p',J)}$	$(I \cup J) \cap \bar{J} = \emptyset$
(par1)	$\dfrac{p \xrightarrow{\alpha,I} p'}{p \mid q \xrightarrow{\alpha,I} p' \mid \mathsf{kill}(q,I)}$	
(par2)	$\dfrac{q \xrightarrow{\alpha,I} q'}{p \mid q \xrightarrow{\alpha,I} \mathsf{kill}(p,I) \mid q'}$	
(sync)	$\dfrac{p \xrightarrow{\lambda,I} p' \qquad q \xrightarrow{\bar{\lambda},J} q'}{p \mid q \xrightarrow{\tau,I\cup J} \mathsf{kill}(p',J) \mid \mathsf{kill}(q',I)}$	$I \cap \bar{J} = \emptyset$
(ract)	$\dfrac{p \xrightarrow{\alpha,I} p'}{p\backslash a \xrightarrow{\alpha,I} p'\backslash a}$	$\alpha \neq a, \bar{a}$
(rconf)	$\dfrac{p \xrightarrow{\alpha,I} p'}{p\backslash n \xrightarrow{\alpha,I\backslash\{n,\bar{n}\}} p'\backslash n}$	
(rec)	$\dfrac{p[\mathrm{rec}\, x.p/x] \xrightarrow{\alpha,I} p'}{\mathrm{rec}\, x.p \xrightarrow{\alpha,I} p'}$	

Table 2. Auxiliary function

$$
\begin{array}{rclrcl}
\mathsf{kill}(nil,I) & = & nil & \mathsf{kill}(\alpha.p,I) & = & \alpha.\mathsf{kill}(p,I) \\
\mathsf{kill}(p\backslash a,I) & = & \mathsf{kill}(p,I)\backslash a & \mathsf{kill}(p\backslash n,I) & = & \mathsf{kill}(p,I \setminus \{n,\bar{n}\})\backslash n \\
\mathsf{kill}(p \mid q,I) & = & \mathsf{kill}(p,I) \mid \mathsf{kill}(q,I) & \mathsf{kill}(\mathsf{rec}\, x.p,I) & = & \mathrm{rec}\, x.\mathsf{kill}(p,I) \\
\mathsf{kill}(x,I) & = & x & & &
\end{array}
$$

$$
\mathsf{kill}(Jp,I) = \begin{cases} (J \setminus I)\mathsf{kill}(p,I) & \text{if } J \cap \bar{I} = \emptyset \\ nil & \text{otherwise} \end{cases}
$$

In the rule (sync) the communication can happen only if the sender and the receiver are not in conflict with each other. For instance, the process $1a \mid \bar{1}\bar{a}$ cannot perform a τ action. If the side condition is satisfied and the communication takes place, we label the transition with the union of the conflicts of the sender and the receiver, to broadcast the conflicts to the whole system. Moreover, each of the two subagents propagates its conflict set to the other one in order to kill all the conflicting subterms, as in the case of the asynchronous move.

We say that a DiX agent p is *closed w.r.t. conflict names* if every conflict n occurring in p is within the scope of a $\backslash n$ operator. The effect is that p does not "export" conflicts, i.e., for any p' reachable from p we have that $p' \xrightarrow{\alpha,I} q$ implies $I = \emptyset$.

It is not difficult to imagine how a flow event structure [3] can be associated to an agent (closed w.r.t. conflicts), where the conflict names are used to build the conflict relation. Instead of defining the operations on these structures – which is outside the scope of the paper – we would like to give the flavour of our claim with a couple of examples.

Example 3.1 The prime event structure [26] composed of three events – labeled by *non internal* actions a, b and c, respectively – where a and c are concurrent, a causes b and b is in conflict with c can be expressed in DiX by the agent $(a.1b.nil|\bar{1}c.nil)\backslash 1$, but there is no way to write a CCS term which strongly (e.g., modulo strong causal bisimulation [7]) represents it. As a matter of fact, the binary operator + prescribes conflicts between actions which are "ready at the same time", and this is not the case for b and c above.

As another example, consider the event structure composed by four events – labeled by a, b, c and d, respectively – where the causality relation is empty and the conflict relation contains only the pairs in $\{(a,b),(b,c),(c,d)\}$. Even if all the four events are ready, this structure is not expressible in CCS [4] because a conflict can be expressed between two observable actions only if they appear "syntactically close" to each other. The DiX term representing this structure is $(1a.nil|\{\bar{1},2\}b.nil|\{\bar{2},3\}c.nil|\bar{3}d.nil)\backslash\{1,2,3\}$. Note that in both cases the conflict relation is translated in the DiX term at the right syntactical level. □

We now introduce the observational semantics for DiX, based on bisimulation.

Definition 3.2 *A* binary relation R over the set of processes is a *bisimulation* if $(p,q) \in R$ implies:

(i) if $p \xrightarrow{\alpha,I} p'$, there exists then q' such that $q \xrightarrow{\alpha,I} q'$ and $(p',q') \in R$;

(ii) symmetrically for q derivations.

Two processes p and q are *bisimulation equivalent*, written $p \sim q$, if there exists a bisimulation R such that $(p,q) \in R$. □

In order to prove that bisimulation equivalence is a congruence, we exploit a classic result, ensuring the congruence property if all the operational rules fit the $tyft$ format [19]. To this aim, we slightly change our syntax and semantics. The grammar for agents is extended to include also the auxiliary function kill as a first class operator. Hence, $\mathsf{kill}(p,J)$ is an agent provided that p is an agent and J is a (finite) set of conflicts. The set of all these extended agents is denoted by $\mathcal{P}_{kill}$. Then we extend Table 2 with the rule:

$$\text{(kill)} \qquad \frac{p \xrightarrow{\alpha,I} p'}{\mathsf{kill}(p,J) \xrightarrow{\alpha,I\backslash J} \mathsf{kill}(p',J)} \qquad I \cap \bar{J} = \emptyset$$

Observe that the rules in Table 1 and (kill) (i) fit with $tyft$ format, and (ii) are equivalent to the original one on the restricted set of agents $\mathcal{P}$. Indeed, the equations listed in Table 2 are all sound w.r.t. strong bisimulation in the transition system for $\mathcal{P}_{kill}$. This justifies the following

4. This structure, as well as the previous one, can be represented with + if a TCSP-like parallel operator is provided. However, this descriptive capability is mainly due to the other very powerful operator, rather than by the expressiveness of +.

Theorem 3.3 $\sim$ is a congruence on the operators of DiX.

4 Embedding CCS in DiX

In this section we prove that we can embed CCS (without relabeling) within DiX. Essentially, we need to prove that the choice can be expressed as a derived operator. Here, for lack of space, we do not recall the CCS transition system and its bisimulation semantics [22, 23]. We use variables u and v to range over CCS agents, and we also write $u \sim v$ to say that u and v are strongly bisimilar, according to the operational semantics in [22, 23]. The translation $[\![\ ]\!]$ of CCS into DiX is homomorphic w.r.t. all CCS operations, except for +. The definition in this case is:

$$[\![u + u']\!] = (1[\![u]\!]|\bar{1}[\![u']\!])\backslash 1$$

Proposition 4.1 If $[\![p]\!] \xrightarrow{\alpha, I} q$, then $I = \emptyset$.

Namely, the encoding of a CCS agent into DiX never shows up conflicts, i.e. it is a closed process w.r.t. conflicts. The following proposition shows a strong relationship between the transition system for CCS [22, 23] and for DiX: each transition of a CCS term is matched by a corresponding transition of its translation into DiX, and *vice versa*. Then we state that the translation is sound and complete w.r.t. strong bisimulation.

Proposition 4.2 Let u be a CCS term. If $u \xrightarrow{\alpha} u'$, there exists p such that $[\![u]\!] \xrightarrow{\alpha, \emptyset} p$ and $p \sim [\![u']\!]$. Vice versa, if $[\![u]\!] \xrightarrow{\alpha, \emptyset} p$, there exists u' such that $u \xrightarrow{\alpha} u'$ and $[\![u']\!] \sim p$.

Theorem 4.3 Let u and v be CCS terms. We have that $u \sim v$ if and only if $[\![u]\!] \sim [\![v]\!]$.

5 Net Semantics

Now we provide a net semantics for DiX. As a consequence of Theorem 4.3, we get a net semantics for CCS, too. Let us define the nets we shall use.

5.1 P/T nets with inhibitor arcs

We recall simple Place/Transition nets without capacities on places (see, e.g., [28]). Then, we extend it with the so-called inhibitor arcs (see, e.g., [20, 6, 21, 25]). Here we provide a characterization of this model which is convenient for our aims.

Definition 5.1 *G* iven a set S, a *finite multiset* over S is a function $m : S \to I\!N$ such that the set $dom(m) = \{s \in S \mid m(s) \neq 0\}$ is finite. The *multiplicity* of an element s in m is given by the natural number $m(s)$. The set of all finite multisets over S, denoted by $\mathcal{M}_{fin}(S)$, is ranged over by m. A multiset m such that $dom(m) = \emptyset$ is called *empty* . A *finite set* is a special case of finite multiset, when the multiplicity of each s is at most one. The set of all finite sets over S is denoted by $\wp_{fin}(S)$.

We write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$. The operator $\oplus$ denotes *multiset union*: $m \oplus m'(s) = m(s) + m'(s)$. The operator $\setminus$ denotes *multiset difference*: $m \setminus m'(s) = m(s) \dot{-} m'(s)$ where $i \dot{-} j = i - j$ if $i > j$, $i \dot{-} j = 0$ otherwise. Finally, the *scalar product*, $k \cdot m$, of a number k with a multiset m is $(k \cdot m)(s) = k \cdot (m(s))$. □

Definition 5.2 *A* P/T net is a tuple $N = (S, Act, T)$ where S is the set of *places*, Act is the set of *labels* and $T \subseteq \mathcal{M}_{fin}(S) \times Act \times \mathcal{M}_{fin}(S)$ is the set of *transitions*. A finite multiset over the set

S of places is called *marking*. Given a marking m and a place s, we say that the place s contains $m(s)$ *tokens*.

A transition $t = (c, a, p)$ is usually written in the form $c \xrightarrow{a} p$. The marking c, usually denoted with ${}^\bullet t$, is called the *preset* of t and represents the tokens to be "consumed"; the marking p, usually denoted with $t^\bullet$, is called the *postset* of t and represents the tokens to be "produced"; a is called the *label* of t, sometimes denoted by $l(t)$. A transition t is *enabled* at m if ${}^\bullet t \subseteq m$. The execution of a transition t enabled at m produces the marking $m' = (m \setminus {}^\bullet t) \oplus t^\bullet$. This is written as $m[t\rangle m'$.

A *marked* P/T net is a tuple $N(m_0) = (S, Act, T, m_0)$, where (S, Act, T) is a P/T net and m_0 is a nonempty finite multiset over S, called the *initial marking*. □

Definition 5.3 *A* P/T net *with inhibitor arcs* (NI net for short) is a tuple $N = (S, Act, T)$ where S is the set of places, Act the set of labels and $T \subseteq \mathcal{M}_{fin}(S) \times \wp_{fin}(S) \times Act \times \mathcal{M}_{fin}(S)$ is the set of transitions.

A transition $t = (c, i, a, p)$ is usually written in the form $(c, i) \xrightarrow{a} p$. The set i, denoted with ${}^\circ t$, is called the *inhibitor-set* of t and represents the tokens to be "tested for absence"; markings c and p are as above. This changes the definition of enabling: a transition t is enabled at m if ${}^\bullet t \subseteq m$ and $dom(m) \cap {}^\circ t = \emptyset$. Thus, any transition t for which ${}^\circ t \cap dom({}^\bullet t) \neq \emptyset$ is called *blocked*, *unblocked* otherwise.

The execution of a transition t enabled at m producing the marking m', written $m[t\rangle m'$, is defined as above. □

Definition 5.4 *L* et $N = (S, Act, T)$ be a net. The *interleaving marking graph* of N is $IMG_N = (\mathcal{M}_{fin}(S), \rightarrow, Act)$, where $\rightarrow \subseteq \mathcal{M}_{fin}(S) \times Act \times \mathcal{M}_{fin}(S)$ is defined by $m \xrightarrow{l(t)} m'$ iff there exists a transition $t \in T$ such that $m[t\rangle m'$. With $IMG_N(m)$ we denote the interleaving marking graph reachable from the initial marking m. □

As a IMG is a labeled transition system, bisimulation equivalence can be defined on markings too; with $m \sim m'$ we denote that m and m' are bisimilar.

For nets with inhibitor arcs, a transition may be considered to be *concurrent* to another only if the former neither produces nor consumes tokens inhibiting the latter and vice versa. In other words, if a marking enables both, they can be executed in either order. Formally, a subset T' of unblocked transitions is a *set of possibly concurrent transitions* (SPCT for short) if $\forall t_1, t_2 \in T'.\, t_1 \neq t_2$ we have that $dom(t_1{}^\bullet) \cap {}^\circ t_2 = \emptyset = dom({}^\bullet t_1) \cap {}^\circ t_2$. A transition t which produces some token in its inhibitor-set (i.e., $dom(t^\bullet) \cap {}^\circ t \neq \emptyset$) is called *not self-concurrent*.

Definition 5.5 *G* iven a NI $N = (S, Act, T)$, let $G \in \mathcal{M}_{fin}(T)$ be a finite multiset of transitions. G is *enabled* at *marking* m iff

(i) $dom(G)$ is a SPCT,

(ii) if $t \in dom(G)$ is not self-concurrent, then $G(t) = 1$,

(iii) $m_1 \subseteq m$, where $m_1 = \bigoplus_{t \in dom(G)} G(t) \cdot {}^\bullet t$, and

(iv) $m(s) = 0$ for all $s \in \bigcup_{t \in dom(G)} {}^\circ t$.

If G is enabled at m, G is called a *multistep* from m to m', where $m' = (m \setminus m_1) \oplus m_2$ with $m_2 = \bigoplus_{t \in dom(G)} G(t) \cdot t^\bullet$. This will be denoted by $m[G\rangle m'$. □

Definition 5.6 *L* et $N = (S, Act, T)$ be a net. Its *multistep marking graph* is $MMG_N \stackrel{\text{def}}{=} (\mathcal{M}_{fin}(S), \rightarrow, Act)$ where $\rightarrow \subseteq \mathcal{M}_{fin}(S) \times \mathcal{M}_{fin}(Act) \times \mathcal{M}_{fin}(S)$ is defined by $m \xrightarrow{L} m'$ iff $\exists G \in \mathcal{M}_{fin}(T). m[G\rangle m'$ and $L \in \mathcal{M}_{fin}(Act)$ is such that $\forall a \in Act. L(a) = \sum_{l(t)=a} G(t)$. With $MMG_N(m)$ we denote the multistep marking graph reachable from the initial marking m. □

5.2 DiX as a Net with Inhibitor Arcs

The P/T net semantics proposed in the literature are of two types. The first one, called *location*-oriented, exploits the syntactical structure of the process terms (notably, the parallel operator) to define their associated sets of places. This idea dates back to [8] and is used also in [9, 27]. The second one, called *label*-oriented, ignores the syntactical structure of the parallel operator and only information on the label of the transition is kept. For instance, [13, 15] study this approach for CCS without restriction and relabeling, while [11] follows this line to represent the reduction semantics of π-calculus [24]. Our approach mainly follows the label-oriented one, where places have a syntactical name as in [15, 11]. The approach is extended here to nets with inhibitor arcs. The main features of our proposal are:

(i) For each $a \in \mathcal{A}$, there is a place named $\backslash a$ which inhibits the execution of all the transitions with label a or $\bar{a}$.

(ii) For each conflict name $n \in Con$, there is a place named n which inhibits the transitions of all those processes having $\bar{n}$ among their conflict names.

Moreover, as we do not consider the syntactical structure of parallel composition, we need to have always fresh names (for both actions and conflicts) to rename the restriction binders. To be more explicit, if we want to give a net semantics to $(a.nil|\bar{a}.nil)\backslash a|a.nil$, we would like to say that the restriction place $\backslash a$ must inhibit only the transitions for a and $\bar{a}$ inside its scope (letting the synchronization to occur freely) and not the transition of the rightmost subprocess. This can be easily obtained by requiring the bound occurrence of a is transformed into a new name a', leaving the name a for the free occurrence of a; the resulting equivalent term is $(a'.nil|\bar{a'}.nil)\backslash a'|a.nil$. This renaming is needed also for the bound conflicts. The function dec which associates a marking to every agent changes all the bound names to new, fresh ones. Hence, a restriction binder really acts only on the bound transitions inside its scope.

The set S_{DiX} of places is defined as follows:

$$\begin{array}{rcl} S_{DiX} & = & \{nil\} \\ & \cup & \{I\alpha.(p, A, C) \mid I \subseteq Con, \alpha \in Act, p \in \mathcal{P}, A \subseteq \mathcal{A}, C \subseteq \mathcal{C}\} \\ & \cup & \{\backslash a \mid a \in \mathcal{A}\} \\ & \cup & \{n \mid n \in Con\} \end{array}$$

The place $I\alpha.(p, A, C)$ denotes a sequential process – with I as conflict set – that can perform an α and after that behaves as p where, however, the bound action names and conflict names must be renamed with the names in A and C, respectively.

In order to define the decomposition function *dec* which associates a finite multiset on S_{DiX} to each DiX agent, we need some auxiliary definitions.

When we say that a set A is denumerable, we implicitly assume that an effective enumeration is associated to $A = \{A_0, A_1, \ldots, A_i, \ldots\}$. With $Even(A)$ and $Odd(A)$, we denote the sets $\{A_0, A_2, \ldots, A_{2i}, \ldots\}$ and $\{A_1, A_3, \ldots, A_{2i+1}, \ldots\}$, respectively; furthermore, $first(A)$ denotes A_0. The operator $I(\_)$ on sets of places applies elementwise according to the following:

$$\begin{array}{l} I(nil) = nil \\ I(\backslash a) = \backslash a \\ I(J\alpha.(p, A, C)) = (I \cup J)\alpha.(p, A, C) \\ I(n) = n \end{array}$$

Finally, with the substitution $p[b'/b]$ of a name $b' \notin Sort_a(p)$ for b in p we mean that all the free occurrences of action b are replaced by the new name b'; note that as b' is new, no free occurrence of b is bound after the substitution. Analogously for $p[m'/m]$, where $m' \notin Sort_c(p)$.

The function $dec : \mathcal{P} \times \wp(\mathcal{A}) \times \wp(\mathcal{C}) \longrightarrow \mathcal{M}_{fin}(S_{DiX})$, which defines the decomposition of DiX processes into markings, is reported in Table 3.

The NI net for DiX is the triple $N_{DiX} = (S_{DiX}, Act, T_{DiX})$, where the set T_{DiX} of net transitions is the least set generated by the three axiom schemata reported in Table 4. Given an agent p, the NI net associated to p is N_{DiX} with initial marking $dec(p, \mathcal{A} \setminus Sort_a(p), \mathcal{C} \setminus Sort_c(p))$.

Table 3. Decomposition function

$$dec(nil, A, C) = \{nil\}$$

$$dec(\alpha.p, A, C) = \{\emptyset\alpha.(p, A, C)\}$$

$$dec(p \,|\, q, A, C) = dec(p, \mathrm{Even}(A), \mathrm{Even}(C)) \oplus dec(q, \mathrm{Odd}(A), \mathrm{Odd}(C))$$

$$dec(Ip, A, C) = Idec(p, A, C)$$

$$dec(p\backslash a, A, C) = dec(p[a'/a], A', C) \oplus \{\backslash a'\} \quad a' = first(A) \text{ and } A' = A \setminus \{a'\}$$

$$dec(p\backslash n, A, C) = dec(p[n'/n], A, C') \quad n' = first(C) \text{ and } C' = C \setminus \{n'\}$$

$$dec(\mathrm{rec}\, x.p, A, C) = dec(p[\mathrm{rec}\, x.p/x], A, C)$$

Table 4. Axiom schemata for transitions

1. if $I \cap \bar{I} = \emptyset$ and $\lambda \in \{a, \bar{a}\}$ then

$$(I\lambda.(p, A, C), \{\backslash a\} \oplus \bar{I}) \xrightarrow{\lambda} dec(p, A, C) \oplus I$$

2. if $I \cap \bar{I} = \emptyset$ then

$$(I\tau.(p, A, C), \bar{I}) \xrightarrow{\tau} dec(p, A, C) \oplus I$$

3. if $(I \cup J) \cap (\bar{I} \cup \bar{J}) = \emptyset$ then

$$(Ia.(p, A, C) \oplus J\bar{a}.(p', A', C'), \bar{I} \oplus \bar{J}) \xrightarrow{\tau} dec(p, A, C) \oplus dec(p', A', C') \oplus I \oplus J$$

Let us now illustrate our semantics through two simple examples. We have followed the usual drawing conventions: a place is a circle, decorated by its name. A transition is a short horizontal bar, decorated by its label. An arc of the preset (postset) of a transition is depicted as an arrow from (to) the place to (from) the transition (This arc may be labeled by a natural number, denoting the multiplicity of the tokens; this number is 1 by default). An arc of the inhibitor-set of a transition is represented as a special arrow – with a final, white circle – from the inhibiting place to the transition.

Example 5.7 In Fig. 1 the net for $p = (a|\bar{a})$ is drawn. A net for $q = (a|\bar{a})\backslash a$ can easily be obtained by replacing a with a new name a'. Indeed, a feature of our net semantics is that the only substantial difference between these two nets is in the initial marking: for p there is one token in place $\emptyset a$ and one token in $\emptyset\bar{a}$, whilst the marking for q puts one token in $\backslash a'$, besides $\emptyset a'$ and $\emptyset\bar{a}'$. □

Example 5.8 In Fig. 2 the net for the CCS agent $a + (b|c)$, after its translation in DiX, is drawn. This should help in understanding the main differences in the treatment of distributed choice in our approach w.r.t. the location-oriented ones [9, 12, 27]. The linearity of the representation is

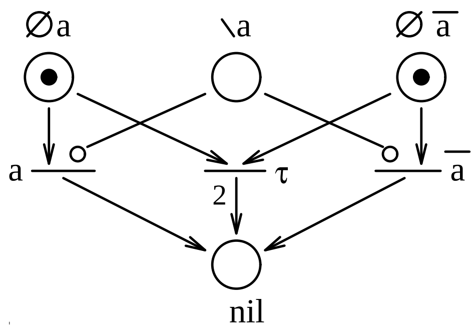

Figure 1. The net for $a|\bar{a}$. The net for $(a|\bar{a})\backslash a$ is isomorphic, but with a token also in the place corresponding to $\backslash a$.

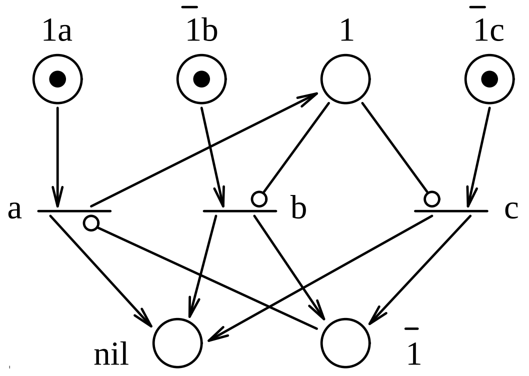

Figure 2. The net for $a + (b|c)$, where the places $\backslash a$, $\backslash b$, $\backslash c$ and the relative inhibitor arcs are omitted.

clear: if p is represented by a net with n places and q is represented by a net with m places, then we need $n + m + 2$ places to reproduce the net for $p + q$. □

Finally, we list some properties of our net semantics which can be easily proved by induction on the structure of transitions.

Proposition 5.9 The following hold for the net N_{DiX}:

(i) For every transition $t \in T_{DiX}$, ${}^\bullet t$ and ${}^\circ t$ are sets, whereas in general $t^\bullet$ may not be a set.

(ii) If $s \in S_{DiX}$ is of the form $\backslash a$ or n, then for every transition $t \in T$ we have that $s \notin {}^\bullet t$.

(iii) If $s \in S_{DiX}$ is of the form $\backslash a$, then for every transition $t \in T$ we have that $s \notin t^\bullet$.

(iv) If $t \in T_{DiX}$, then t is unblocked (i.e., $dom({}^\bullet t) \cap {}^\circ t = \emptyset$).

(v) If $t \in T_{DiX}$, then t is self-concurrent (i.e., $dom(t^\bullet) \cap {}^\circ t = \emptyset$).

5.3 Retrievability

In this section we want to hint the proof of "retrievability" of the interleaving semantics in the net semantics. Formally, we would like to show that for any agent p, the transition system $LTS_{DiX}(p)$ is bisimilar to the interleaving marking graph $IMG_{N_{DiX}}(m)$ where $m = dec(p, \mathcal{A} \setminus Sort_a(p), \mathcal{C} \setminus Sort_c(p))$. However, note that LTS_{DiX} is labeled on pairs (α, I), whilst $IMG_{N_{DiX}}$ is labeled on actions only. Hence, the comparison should somehow take into account also the set of conflicts in the net.

As we would like to simplify as much as possible this correspondence proof, we will compare equivalent variants of both the SOS semantics and the net semantics. In particular, we take the SOS semantics defined on $\mathcal{P}_{kill}$, i.e., with the explicit rule (kill). Moreover, rule (conf) is replaced by the following rule

$$\text{(conf2)} \qquad \frac{p \xrightarrow{\alpha, I} p'}{Jp \xrightarrow{\alpha, I \cup J} \mathsf{kill}(Jp', J)} \qquad (I \cup J) \cap \bar{J} = \emptyset$$

which is equivalent to (conf), since the equation $\mathsf{kill}(p', J) = \mathsf{kill}(Jp', J)$, derivable from those reported in Table 2, is sound w.r.t. bisimulation. Nonetheless, remember that the initial state is always a DiX agent $p \in \mathcal{P}$ and that we are interested only to those $\mathcal{P}_{kill}$ agents which are reachable from a DiX agent.

The net semantics we will consider is also extended to agents in $\mathcal{P}_{kill}$. The marking associated to $\mathsf{kill}(p, I)$ is

$$dec(\mathsf{kill}(p, I), A, C) = dec(p, A, C) \oplus I$$

where the declared set of conflict names I is transformed in an explicit set of conflict tokens in the net. This extension is conservative, as no net transition produces a term of the form $\mathsf{kill}(p, I)$. Furthermore, rules in Table 4 are modified by "prefixing" the conflict set of the consumed tokens to the produced ones. For instance, the first axiom scheme is replaced by the following:

$$\text{if } I \cap \bar{I} = \emptyset \text{ and } \lambda \in \{a, \bar{a}\} \text{ then } (I\lambda.(p, A, C), \{\backslash a\} \oplus \bar{I}) \xrightarrow{\lambda} Idec(p, A, C) \oplus I$$

where the set I occurs also as a prefix of $dec(p, A, C)$. We first show that this new net semantics is bisimulation equivalent to the original one. The sufficient condition is provided by the following proposition, which justifies the additional "prefix" I in the new rules.

Proposition 5.10 Given $p \in \mathcal{P}$, let $m \oplus I$ be a marking reachable from the initial marking $dec(p, \mathcal{A} \setminus Sort_a(p), \mathcal{C} \setminus Sort_c(p))$ with the transitions in T_{DiX}. Then, $m \oplus I \;\sim\; Im \oplus I$.

Intuitively, this holds because if a transition has produced the tokens in I, then what Im can do is exactly what m can do: the former checks the absence of $\bar{I}$, and this is ensured by the fact that in no reachable marking a conflict place n and its opposite $\bar{n}$ can both contain some tokens.

We now introduce a new equivalence notion on markings, we call *conflict similarity*, which implies bisimulation and also has some nice structural properties. These will be helpful in establishing the correspondence between the new SOS operational semantics and the new net semantics which, in fact, is set modulo conflict similarity of markings.

Definition 5.11 L et m, m' be two markings. We say that m is conflict-similar to m', $m \simeq m'$, if $\forall n \in S_{DiX} .\, m(n) > 0 \Leftrightarrow m'(n) > 0$ and $m(s) = m'(s)$ for all $s \neq n, s \in S_{DiX}$. □

Proposition 5.12 The following hold:

(i) If $m \simeq m'$, then $m \sim m'$.
(ii) If $m \simeq m'$ then $Im \simeq Im'$.
(iii) If $m \simeq m'$ then $m \oplus m'' \simeq m' \oplus m''$.
(iv) $m \oplus I \simeq m \oplus I \oplus I$.

Finally, the main theorem which states the correspondence between net transitions and SOS ones and shows that the set I of conflicts in the label of a SOS transition is exactly the set of conflict places which possess at least one token in the target marking and do not possess tokens in the source marking, and whose name is not local, i.e. has not been renamed by a new one.

Theorem 5.13 Let $p \in \mathcal{P}_{kill}$ be reachable from $q \in \mathcal{P}$; let also A be such that $A \cap Sort_a(p) = \emptyset$ and C such that $C \cap Sort_c(p) = \emptyset$.

If $p \xrightarrow{\alpha, I} p'$, there exists then a marking m such that $dec(p, A, C) \xrightarrow{\alpha} m$, $m \simeq dec(p', A, C)$ and $I = \{n \in Con \mid n \in m \text{ and } n \notin (dec(p, A, C) \oplus C \oplus \overline{C})\}$.

If $dec(p,A,C) \xrightarrow{\alpha} m$ there exists then p' such that $p \xrightarrow{\alpha,I} p'$, $dec(p',A,C) \simeq m$ and $I = \{n \in Con \mid n \in m$ and $n \notin (dec(p,A,C) \oplus C \oplus \overline{C})\}$.

The proof is by induction on the reachability relation, and then by induction on the proof of each single transition.

The *concurrent semantics* for a DiX agent p is given by its multistep marking graph $MMG_{N_{DiX}}(m)$, where $m = dec(p, \mathcal{A} \setminus Sort_a(p), \mathcal{C} \setminus Sort_c(p))$.

6 Concluding Remarks

The process algebra DiX is able to express very complex forms of binary conflicts. Moreover, it is not difficult to adapt the given definitions to express also n-ary conflicts (even if the net semantics does not scale to n-ary conflicts). The main changes are the following ones:

(i) no opposite conflict names are needed,

(ii) a conflict relation $\mathcal{R}$ is to be considered as external to the syntax,

(iii) a different side condition to rules (conf) and (sync) in order to prevent the possible execution of those transitions labeled by a set of conflicts in $\mathcal{R}$,

(iv) kill(p,I) is Ip.

A simple example of ternary conflict is represented by a system composed of three events – labeled a, b and c, respectively – which can be executed in pairs, but never all together. This is represented by $((1a.nil|2b.nil)|3c.nil)\backslash\{1,2,3\}$ with the external conflict relation $\mathcal{R} = \{(1,2,3)\}$.

The main achievement of this paper is a new net semantics for CCS, based on a class of nets which has never been used to model concurrent languages (to the best of our knowledge). Multistep semantics for P/T nets with inhibitor arcs have been investigated in [6] where, however, a different notion of P/T net (and of concurrent enabling) is considered. Also some nice work on C/E nets [28] with inhibitor arcs is reported in [21] and in [25]; we share with the latter the same basic intuition on concurrency: two transitions such that one produces a token in a place of the inhibitor set of the other cannot be performed in one multistep, as they cannot be executed in either order. Future research will be devoted to prove that the net semantics we have defined shows all the intended concurrency. To this aim, we need first to define properly what is a causal semantics for P/T nets with inhibitor arcs, and then prove that it coincides with the other causality based semantics proposed for CCS (see, e.g., [4]).

An obvious question is whether we can include relabeling in our approach. Even if we can extend DiX with the relabeling operator and prove that CCS is a subcalculus of DiX with relabeling, the net semantics we can offer for this operation is not very attractive. (Similar problems arise also for the operator of TCSP parallel composition, even if the solution we can offer is less awkward.) Hence, we have preferred to skip this operator.

Nonetheless, we think that our approach is general enough to show its usefulness in other cases. For instance, for π-calculus [24] (because relabeling as well as other structural operators are not part of that calculus). Therefore, similarly to DiX which is the distributed conflict analogous to CCS, we can design a new first-order calculus with distributed conflicts which is an extension of the π-calculus. We claim that the DiX-net semantics described here scales – with minor adjustments – to this new calculus, hence providing also full π-calculus with a distributed semantics in term of P/T nets with inhibitor arcs [5].

In our approach, the need of using P/T nets extended with inhibitor arcs is due to the fact that the basic model is not powerful enough to express external nondeterminism and restriction, if we do not make use of the syntactical structure of the terms, as in [8]. Because of the equivalence of P/T nets with the basic CHAM [2] and the Gamma language [1], we think that also these models can be similarly extended to cope with nondeterminism.

As a final comment, we would like to draw some analogy between our approach to net semantics and the so-called “ask & tell” coordination mechanism. A shared data space of conflicts and restrictions is tested each time an action is executed in order to check the absence of the conflicts of the active subagent and of the restriction of the action that the subagent wants to perform. Moreover, the execution of the action can produce some new elements in the shared data space. We could say that it is a form of concurrent constraint, where the constraints increase monotonically (tokens in places of conflicts and restrictions are never consumed). Future research will be devoted to study the relationship between these two different models of computation.

Acknowledgements
We would like to thank Cosimo Laneve and the anonymous referee for their useful comments.

References

[1] J.P. Banâtre, D. Le Métayer, "Programming by Multiset Transformation", *Comm. of ACM* 36, 98-111, 1993.
[2] G. Berry, G. Boudol, "The Chemical Abstract Machine", *Theoretical Computer Science* 96, 217-248, 1992.
[3] G. Boudol, "Flow Event Structures and Flow Nets", LNCS 469, Springer, 62-95, 1990.
[4] G. Boudol, I. Castellani, "Three Equivalent Semantics for CCS", LNCS 469, Springer, 96-141, 1990.
[5] N. Busi, R. Gorrieri, "A Distributed Semantics for π-calculus based on P/T Nets with Inhibitor Arcs", forthcoming.
[6] G. Chiola, S. Donatelli, G. Franceschinis "Priorities, Inhibitor Arcs and Concurrency in P/T Nets", in Proc. *12^{th} Int. Conf. on Appl. and Theory of Petri Nets'91*, 182-205, 1991.
[7] Ph. Darondeau, P. Degano, "Causal Trees", in Proc. *ICALP'89*, LNCS 372, Springer, 234-248, 1989.
[8] P. Degano, R. De Nicola, U. Montanari, "Partial Ordering Derivations for CCS", in Proc. *FTC'85*, LNCS 199, Springer, 520-533, 1985.
[9] P. Degano, R. De Nicola, U. Montanari, "A Distributed Operational Semantics for CCS based on C/E Systems", *Acta Informatica* 26, 59-91, 1988.
[10] P. Degano, R. De Nicola, U. Montanari, "Partial Ordering Semantics for CCS", *Theoretical Computer Science* 75, 223-262, 1990.
[11] J. Engelfriet, "A Multiset Semantics for the π-calculus with Replication", in Proc. *CONCUR'93*, LNCS 715, Springer, 7-21, 1993.
[12] R. van Glabbeek, F. Vaandrager, "Petri Net Models for Algebraic Theories of Concurrency", in Proc. *PARLE'87* , LNCS 259, Springer, 224-242, 1987.
[13] U. Goltz, "On Representing CCS Programs by Finite Petri Nets", in Proc. *MFCS'88*, LNCS 324, Springer, 339-350, 1988.
[14] U. Goltz, "CCS and Petri Nets", LNCS 469, Springer, 334-357, 1990.
[15] R. Gorrieri, U. Montanari, "SCONE: A Simple Calculus of Nets", in Proc. *CONCUR'90*, LNCS 458, Springer, 2-30, 1990.
[16] R. Gorrieri, U. Montanari, "Distributed Implementation of CCS", in *Advances in Petri Nets'93*, LNCS 674, Springer, 244-266, 1993.
[17] R. Gorrieri, U. Montanari, "On the Implementation of Concurrent Calculi in Net Calculi: Two Case Studies", *Theoretical Computer Science*, to appear.
[18] R. Gorrieri, S. Marchetti, U. Montanari, "A^2CCS: Atomic Actions for CCS", *Theoretical Computer Science* 72(2/3), 202-223, 1990.
[19] J.F. Groote, F. Vaandrager, "Structured Operational Semantics and Bisimulation as a Congruence", *Information and Computation* 100(2), 202-260, 1992.
[20] M. Hack, "Petri Net Languages", Technical Report 159, MIT, 1976.
[21] R. Janicki, M. Koutny, "Invariant Semantics of Nets with Inhibitor Arcs", in Proc. *CONCUR'91*, LNCS 527, Springer, 317-331, 1991.
[22] R. Milner, *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980.
[23] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.
[24] R. Milner, J. Parrow, D. Walker, "A Calculus of Mobile Processes", *Information and Computation* 100, 1-77, 1992.
[25] U. Montanari, F. Rossi "Contextual Nets", *Acta Informatica*, to appear.
[26] M. Nielsen, G.D. Plotkin, G. Winskel, "Petri Nets, Event Structures and Domains: Part I", *Theoretical Computer Science* 13(1), 85-108, 1981.
[27] E. R. Olderog, *Nets, Terms and Formulas*, Cambridge Tracts in Theoretical Computer Science 23, CUP, 1991.
[28] W. Reisig, "Petri Nets: An Introduction", EATCS Monographs in Computer Science, Springer, 1985.