Executing Transactions in Zero-Safe Nets*

Roberto Bruni and Ugo Montanari

Dipartimento di Informatica, Università di Pisa, Italia
{bruni,ugo}@di.unipi.it
http://www.di.unipi.it/~bruni
and http://www.di.unipi.it/~ugo/ugo.html

Abstract Distributed systems are often composed by many heterogeneous agents that can work concurrently and exchange information. Therefore, in their modeling via PT nets we must be aware that the basic activities of each system can vary in duration and can be constituted by smaller internal activities, i.e., transitions are conceptually refined into *transactions*. We address the issue of modeling transactions in distributed systems by using *zero-safe nets*, which extend PT nets with a simple mechanism for transition synchronization. In particular, starting from the zero-safe net that represents a certain system, we give a distributed algorithm for executing the transactions of the system as transitions of a more abstract PT net. Among the advantages of our approach, we emphasize that the zero-safe net can be much smaller than its abstract counterpart, due to the synchronization mechanism.

Keywords: PT nets, zero-safe nets, distributed transactions, net unfolding, reachability.

Introduction

A distributed system can be viewed as a collection of several components that can evolve concurrently by performing local actions, but can also exchange information according to some predefined protocols that specify how two or more components can interact.

Computational models for distributed systems are often defined using transition systems labeled over certain elementary actions. *Place/transition Petri nets* [21,23] (abbreviated as PT *nets*) can be viewed as particular structured transition systems, where the additional algebraic structure (i.e., monoidal composition of states and runs) offers a suitable basis for expressing the concurrency of such elementary actions. In fact they have been extensively used both as a foundational model for concurrent computations and as a specification language, due to their well assessed theory, an intuitive graphical presentation and several supporting tools. Notice that the PT net model assumes that each transition

* Research supported by CNR Integrated Project *Progettazione e Verifica di Sistemi Eterogenei Connessi mediante Reti*; by Esprit Working Groups *CONFER2* and *COORDINA*; and by MURST projects *Tecniche Formali per Sistemi Software* and *TOSCA: Tipi, Ordine Superiore e Concorrenza.*

synchronizes the consumption and production of its pre- and post-set. This assumption has several serious consequences, since, in general, it requires that a local activity can lock several resources, thus inhibiting the behavior of other transitions.

When designing large and complex systems via PT nets, the more convenient approach is of course to start by outlining a very abstract model and then to refine each transition (that might model a complex activity of the system) into a net that offers a better understanding of the modeled activity. For example, communication protocols for passing and retrieving information cannot leave aside, during the refinement, the fact that agent synchronization is built on finer actions (e.g., for sending message requests, acknowledgments). Moreover, such actions must be executed according to certain local/global strategies that must be completed before the interaction is closed. This means that the abstract transition is seen, at the refined level, as a possibly complex distributed computation that succeeds if and only if all the involved component accomplish their tasks, and that we call *transaction*. In particular the *commit* of the transaction synchronizes all the termination operations of local tasks. Of course, for the refinement to be correct, we must assume that the transaction is executed *atomically*, as if it were a transition. Thus the execution strategy can be only partially distributed, since certain local choices must be synchronized. However, this is similar to what happens for ordinary PT nets.

In fact, let us consider a generic interpreter for PT nets. Then, before executing any transition t, the interpreter must lock all the distributed resources that t will consume and this must be done atomically, otherwise a different transition t' could lock some of the resources needed by t. Therefore the interpreter can afford only a certain degree of distribution. There are at least two cases where atomic locking and distribution of the interpreter do not interfere: 1) nets under consideration are deterministic; 2) nets under consideration are *free choice*.[1] In the first case, at a given instant there is only one transition that can consume a certain resource and therefore the locking can be performed non atomically. In the second case all the nondeterministic choices can be performed by looking at just one place, and thus locking is not needed.

Several approaches have appeared in the literature that present different refinement techniques for top-down design of a concurrent system (e.g., the work on *Petri Box calculus* [5,4]). An extensive comparison of the different approaches and an impressive list of references can be found in [6,16]. Typically at each step a single transition (say t) of the actual net N is refined into a suitable subnet M, yielding the net $N[t \rightarrow M]$ (the net M is usually equipped with initial and final transitions that, in $N[t \rightarrow M]$, become connected to the pre- and post-set of t). This is also related to the notion of *general net morphism* proposed by Petri, that can be used to map the refined net into its abstract representative by

[1] We recall that a net is *free choice* if for any transitions t_1 and t_2 whose pre-sets are not disjoint, then the pre-sets of t_1 and t_2 consist of exactly one place, or equivalently, a net is free choice if for any place s in the pre-set of two or more transitions, then the pre-set of any such transition is given by $\{s\}$.

collapsing the structure of M into the transition t. In general some constraints must be assumed on the "daughter" net M for its behavior to be consistent with that of t, as e.g. in [26,25,24].

The approach we pursue in this paper is slightly different, in the sense that all the transitions of the abstract net are refined by runs of *the same* zero-safe net. In particular, the refinements M_1 and M_2 of two transitions t_1 and t_2 of N might use the same kind of resources, whereas in the mentioned approaches, the nets M_1 and M_2 must be disjoint (except for the interfaces of t_1 and t_2). *Zero-safe nets* (ZS nets) have been introduced in [10] to provide a basic synchronization mechanism for transitions — PT net transitions can only synchronize tokens. Besides transitions and ordinary places (here called *stable* places), ZS nets include a set of *zero* places. These are a sort of idealized resources that remain invisible to external observers, whilst *stable markings*, which just consist of tokens in stable places, define the observable states of the system. The idea is that any evolution step of a ZS net starts at some stable marking, evolves through hidden states (i.e., markings with some tokens in zero places) and eventually terminates into a stable marking. Moreover, all the stable tokens produced during a certain step are released together at the end of the step, when the commit is executed.

The synchronization of transitions can thus be performed via zero tokens. Indeed, consider a transition t_1 that produces a token in a certain zero place z, then to complete the step there must be some transition, say t_2, which is able to consume it. Since the computation is atomic for an external observer, t_1 and t_2 look as synchronized at the abstract level. Pursuing this view, a "refined" ZS net and an "abstract" PT net are supposed to model the same given system. The latter, where only stable places are considered, offers the synchronized view, which corresponds to the abstraction from the mechanism for supervising the atomic production and consumption of zero tokens. This mechanism provides a unique computational strategy, which is expressive enough to model, e.g., a multicasting system and simple process algebras (cf. [7,12]). We remark that the notions of "atomicity" and "evaluation strategy" required by the methodology for designing systems via refinement, are built-in inside the ZS net model itself.

The operational and abstract semantics of ZS nets according to the two more diffused net philosophies (called *collective token* and *individual token*) have been presented in [10,11], where it is shown that they can be formulated as universal constructions in the language of category theory. A comparison between the two approaches together with several applications has been discussed in [12] and in the Ph.D. Thesis of first author [7]. A tutorial presentation of the material can be found in [13]. We focus here on a distributed interpreter for ZS nets.

Dining Philosophers and Free Choice Nets. A simple example that fits well the situation discussed so far can be illustrated by modeling the well-known problems of "dining philosophers:" there are n philosophers (with $n \geq 2$) sat on a round table; each having a plate in front with some food on it; between each couple of plates is put a fork, for a total of n forks on the table; each philosopher cyclically thinks and eats, but to eat he needs both the fork on the left and the one on the right of his plate; after eating a few mouthfuls, the philosopher returns the forks on the table and starts thinking again.

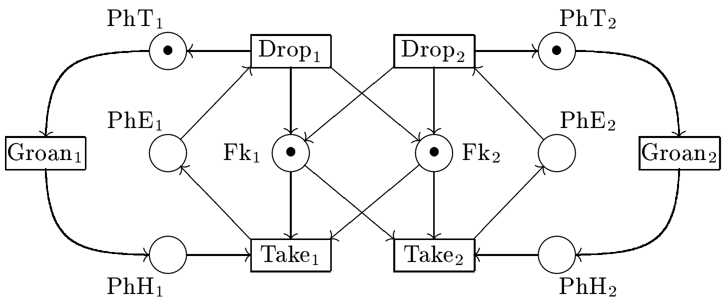

Figure 1. An abstract view for (two) dining philosophers.

The PT net for the case $n = 2$ is illustrated in Figure 1, where, as usual, places are represented by circles, transitions by rectangular boxes, tokens by bullets, and the flow relation by weighted directed arcs (unary weights are omitted). A token in one of the places PhH_i, PhE_i, and PhT_i, for $1 \leq i \leq 2$, means that the ith philosopher is hungry, is eating, and is thinking (soon after eating), respectively. A token in the place Fk_i means that the ith fork is on the table. The transitions Take_i, Drop_i, and Groan_i represent that the ith philosopher takes the forks and starts eating, finishes eating and drops the forks, feels his stomach groaning and prepares to eat, respectively. In fact, note that Take_i requires both forks and thus cannot be accomplished if the other philosopher is eating. The initial marking of the net consists of one token in each of the places PhT_1, PhT_2, Fk_1, and Fk_2 (i.e., both philosophers are thinking and both forks are on the table).

Of course, this model does not tell how the philosophers access the "resources" needed to eat, whereas the action Take_i is not trivial and requires some atomic mechanism for getting the forks. At a more refined level, for example, the strategy for executing the action Take_i could be specified as "take the ith fork (if possible), then the $((i \,\mathbf{mod}\, 2) + 1)$th fork (if possible) and eat," hence it is not difficult to imagine a deadlock where each philosopher takes one fork and cannot continue, since conflict arises. The fact is that the coordination mechanism is hidden inside transitions whose granularity is too coarse.

The situation is completely different if one wants to model the system using *free choice* nets, where all decisions are made locally (i.e., by looking at just one place). To see this, let us concentrate our attention to a subpart of the net in Figure 1, depicted in Figure 2, which will suffice to illustrate the point.

We can translate any net into a free choice net by adding special transitions that perform the local decisions required. For example, the free choice net corresponding to the net in Figure 2 is represented in Figure 3.

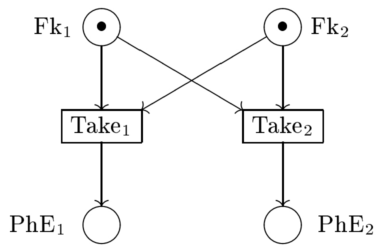

Figure 2. Centralized nondeterminism.

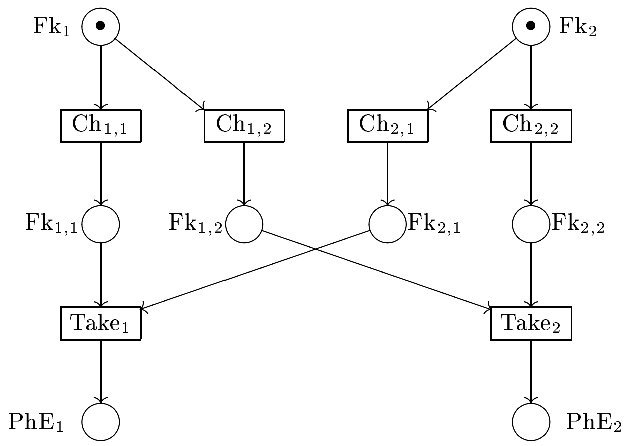

Figure 3. Local nondeterminism (free choice).

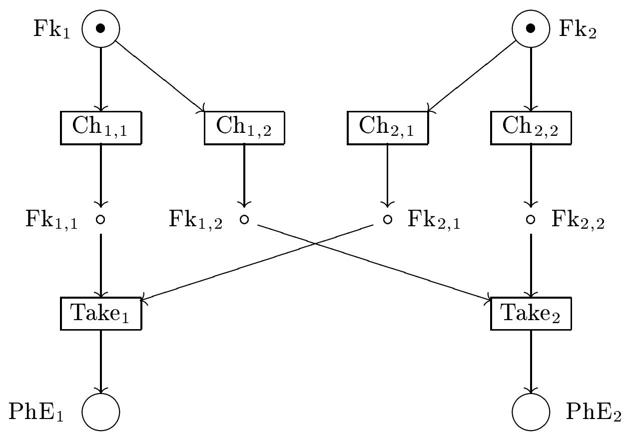

Figure 4. Atomic free choice.

The free choice net in Figure 3 models a system where two decisions can take place independently: one decision concern the assignment of the first fork either to the first or the second philosopher, the other decision concerns the assignment of the second fork. Of course it might as well happen that the first fork is assigned to the first philosopher ($\mathrm{Ch}_{1,1}$) and the second fork is assigned to the second philosopher ($\mathrm{Ch}_{2,2}$), and in such case none of the Take_i actions can occur. Thus, the translated nets admits computations not allowed in the abstract system of Figure 2, and is not a correct implementation.

Zero-safe nets allow for overcoming this problem by executing only certain atomic transactions, where tokens produced in low-level resources are also consumed. In our example, the invisible resources consist of places $\mathrm{Fk}_{i,j}$ for $1 \leq i, j \leq 2$, that can be interpreted as zero-places. In this way the computation performing $\mathrm{Ch}_{1,1}$ and $\mathrm{Ch}_{2,2}$ is forbidden because it stops in an invisible state, i.e., a state that contains zero tokens. Figure 4 represents the low-level model as a ZS net. (Smaller circles stands for zero-places.)

This can be seen as a general procedure for moving from free choice nets to non deadlocking nets.

Concurrent Strategies, Operationally. Though the methodological framework of ZS nets is to some extent satisfactory from the theoretical point of view, from the operational point of view still many aspects must be faced. Indeed, we must specify the computational machinery for performing only correct transactions, recovering deadlocks and treating infinite low-level computations. Moreover, in many cases for such machinery performance could be more relevant than completeness, as it happens in most implementations of Prolog where depth-first goal resolution is preferred to the safer but more resource expensive breadth-first strategy.

In this paper we illustrate our proposal for equipping zero-safe nets with such an operational tool. In particular, we will answer positively to the question: Given a ZS net B and a stable marking u is it possible to compute in a distributed and efficient fashion the set of markings that can be reached from u via the execution of an atomic transaction? The solution relies on a modification of the interpreter for unfolding PT nets [20,27,19], which is extended with a commit rule that enforces the synchronous terminations of transactions. Since there can exist an infinite number of transactions enabled by a marking of a finite ZS net, a termination problem is stated and proposed for further study.

Structure of the paper. In Section 1.1 we recall some basic notion about PT nets, whereas in Section 1.2 we shift the attention to ZS nets. In Section 2 we rephrase the ordinary net unfolding in terms of two simple inference rules and, as an original contribution, extend it to deal with transaction steps on ZS nets. In particular, we show that a fair distributed algorithm based on the modified unfolding rules can be used to enumerate the reachable markings. We conclude by formulating in Section 3 some interesting open problems for which, at the moment, we have only some partial solutions.

1 Preliminaries

In this section we fix the notation and summarize the main concepts needed in the rest of the paper.

1.1 Place/Transition Petri Nets

Place/transition Petri nets (PT nets) can be regarded as graphs with distributed states described by finite distributions of typed resources (called *tokens*), where resources are typed over the set of *places* S. States are usually called *markings* and represented as multisets $u: S \to \mathbb{N}$, where $u(a)$ indicates the number of tokens that place a carries in u. A marking can be written either as $u = \{n_1 a_1, ..., n_k a_k\}$ with $n_i = u(a_i)$, or as the formal sum $u = n_1 a_1 \oplus \cdots \oplus n_k a_k$ denoting an element of the free commutative monoid $S^\oplus$ on the set of places S (the order of summands is immaterial and the monoidal composition is defined by taking $(\bigoplus_i n_i a_i) \oplus (\bigoplus_i m_i a_i) = (\bigoplus_i (n_i + m_i) a_i)$ and 0 as the neutral element). Of course, the monoid $(\mu(S), \cup, \emptyset)$ of finite multisets on S (with multiset union as monoidal operation and the empty multiset as unit) is isomorphic to $S^\oplus$. We denote multiset difference by the operator $\_ \ominus \_$.

Definition 1.1 (PT net). *A marked* PT net *is a 4-tuple $N = (S, T, F, u_{\mathrm{in}})$ where S is the set of places, T is the set of transitions ($S \cap T = \emptyset$), the multirelation $F: (S \times T) \cup (T \times S) \to \mathbb{N}$ is the flow relation and u_{in} is the initial marking of N, making precise the initial state of the system.*

When no confusion can arise we will omit to specify that our nets are "marked." For any transition $t \in T$, the *pre-set* and the *post-set* of t, denoted by $\mathrm{pre}(t)$ and $\mathrm{post}(t)$ respectively, are the multisets over S such that $\mathrm{pre}(t)(a) = F(a, t)$ and $\mathrm{post}(t)(a) = F(t, a)$, for all $a \in S$. The resources in $\mathrm{pre}(t)$ are those required by the action t to be performed (*fired*), and corresponding tokens are consumed by any firing of t. The resources in $\mathrm{post}(t)$ are fresh tokens produced by the action t after a firing. Therefore a PT net $N = (S, T, F, u_{\mathrm{in}})$ can be equivalently defined as the (marked) graph

$$(S^\oplus, T, \mathrm{pre}, \mathrm{post}, u_{\mathrm{in}})$$

where $\mathrm{pre}(\_), \mathrm{post}(\_): T \to S^\oplus$ define the source and target of transitions, respectively. As usual we will write $t: u \to v$ for a transition t with $\mathrm{pre}(t) = u$ and $\mathrm{post}(t) = v$.

The dynamics of a net can be easily expressed by the step relation $\_ \Rightarrow_N \_$ that is defined by the inference rules in Table 1: identities represent idle resources, generators represent the firing of a transition within the minimal marking that can enable it, and parallel composition provides concurrent execution of generators and idle steps.

The extension of this approach to computations $u_0 \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_n$ is not as straightforward as one would expect. Indeed, concurrent semantics must

$$\frac{u \in S^{\oplus}}{u \Rightarrow_N u} \qquad \textbf{(identities)}$$

$$\frac{t\colon u \to v \in T}{u \Rightarrow_N v} \qquad \textbf{(generators)}$$

$$\frac{u \Rightarrow_N v,\ u' \Rightarrow_N v'}{u \oplus u' \Rightarrow_N v \oplus v'} \qquad \textbf{(parallel composition)}$$

Table 1: The inference rules for $\_ \Rightarrow_N \_$.

consider as equivalent all the computations where the same *concurrent* events are executed in different orders, and we cannot leave out of consideration the distinction between *collective* and *individual token philosophies* (noticed e.g., in [17], but see also [8,9] where it is explained how the two approaches can influence the *categorical*, *logical* and *behavioral* semantics of nets).

The simplest approach relies on the collective token philosophy (CTph), where semantics should not distinguish among different instances of the idealized resources. Of course, this is true only if any such instance is *operationally* equivalent to all the others. As a major drawback, this approach disregards that operationally equivalent resources may have different origins and histories, and may, therefore, carry different *causality* information. An alternative approach takes the name of individual token philosophy (ITph). According to the ITph, causal dependencies are a central aspect in the dynamic evolution of a net. As a consequence, only the computations that refer to isomorphic *Goltz-Reisig processes* [18] can be identified, and causality information is fully maintained, whereas for CTph the equivalence classes correspond to the *commutative processes* of Best and Devillers [3].

Whatever the preferred approach is, a more informative semantics is required, which takes into account suitable *proof terms* for computations. Such proof terms can be axiomatized to faithfully recover process semantics. If one is simply interested in "reachability" matters then proof terms can be ignored and consequently also the distinction between the CTph and ITph is not needed. In this case the obvious rules

$$\frac{u \Rightarrow_N v}{u \Rightarrow_N^* v} \qquad \textbf{(one-step computations)}$$

$$\frac{u \Rightarrow_N^* v,\ v \Rightarrow_N w}{u \Rightarrow_N^* w} \qquad \textbf{(sequential composition)}$$

can be introduced (transitive closure). In this sense, processes can be seen as (concurrent) *computation strategies*.

1.2 Zero-Safe Nets

Zero-safe nets augment PT nets with special places called *zero places*. Their role is to coordinate, via the token flow, the atomic execution of several transitions. From an abstract viewpoint, all the coordinated transitions will appear as being synchronized. However no new interaction mechanism is needed, and the coordination of the transitions participating in a step is handled by the ordinary behavioral rules.

Definition 1.2 (Zero-safe net). *A* zero-safe net *(also* ZS *net for short) is a tuple* $B = (S^{\oplus}, T, \mathrm{pre}, \mathrm{post}, u_{\mathrm{in}}, Z)$ *where* $N_B = (S^{\oplus}, T, \mathrm{pre}, \mathrm{post}, u_{\mathrm{in}})$ *is the* underlying PT *net and the set* $Z \subseteq S$ *is the set of* zero places. *The places in* $S \setminus Z$ *are called* stable places. *A* stable marking *is a multiset of stable places, and the initial marking* u_{in} *must be stable.*

Stable markings describe "observable" states of the system, whereas the presence of one or more zero tokens in a given marking denotes it as internal and hence unobservable. We call *stable tokens* and *zero tokens* the tokens that respectively belong to stable places and to zero places.

Remark 1.1. Since $S^{\oplus}$ is a free commutative monoid, it is isomorphic to the cartesian product $(S \setminus Z)^{\oplus} \times Z^{\oplus}$ and we can write $t\colon (u, x) \to (v, y)$ for a transition t with $\mathrm{pre}(t) = u \oplus x$ and $\mathrm{post}(t) = v \oplus y$, where u and v are stable multisets and x and y are multisets over Z.

As for PT nets, we can define the behavior of ZS nets by means of a step relation $\_ \Rrightarrow_B \_$, defined by the inference rules in Table 2. An auxiliary relation $\_ \rightrightarrows_B \_$ is introduced for modeling *transaction segments*.

$\dfrac{u \oplus x \Rightarrow_{N_B} v \oplus y, \quad u, v \in (S \setminus Z)^{\oplus}, \quad x, y \in Z^{\oplus}}{(u, x) \rightrightarrows_B (v, y)}$	**(underlying)**
$\dfrac{(u, x) \rightrightarrows_B (v, y), \;\; (u', y) \rightrightarrows_B (v', y')}{(u \oplus u', x) \rightrightarrows_B (v \oplus v', y')}$	**(horizontal composition)**
$\dfrac{(u, 0) \rightrightarrows_B (v, 0)}{u \Rrightarrow_B v}$	**(commit)**

Table 2: The inference rules for $\_ \Rrightarrow_B \_$.

Notice that we can take advantage of the step relation $\_ \Rightarrow_{N_B} \_$ of the underlying net for concurrently executing several transitions (rule **underlying**). The rule **horizontal composition** acts as parallel composition for stable resources and as sequential composition for zero places. We call it "horizontal" because we prefer to view it as a synchronization mechanism rather than as the ordinary sequential composition of computations, which flows vertically from top to down. The rule **commit** select the transaction segments that correspond to acceptable steps: they must start from a stable marking and end up in a stable marking. Note that, as a particular instance of the horizontal composition of two transaction segments $(u, 0) \rightrightarrows_B (v, 0)$ and $(u', 0) \rightrightarrows_B (v', 0)$, we can derive their parallel composition $(u \oplus u', 0) \rightrightarrows_B (v \oplus v', 0)$.

Abstract Net. At an abstract level, the system modeled via the ZS net B can be equivalently described via a PT net $\mathcal{N}(B)$ such that $S_{\mathcal{N}(B)} = S_B \setminus Z_B$ and $(\_ \Rightarrow_{\mathcal{N}(B)} \_) = (\_ \Rrightarrow_B \_)$. Among the several PT nets that verify these conditions we can choose the optimal one, which is freely generated [10,11]. Informally the transitions of such net must represent the proofs of transaction steps $u \Rrightarrow_B v$ taken up to equivalence (permutation of concurrent events) and that cannot be decomposed into shorter proofs. When these two conditions are verified, the concurrent kernel of the behavior has been identified, and all the steps can be generated by it.

It is worth remarking that the construction of (proved) transaction steps can be defined, in the language of category theory, as an adjunction, i.e., it is a free construction and thus preserves several net compositions (defined as colimits in the category of ZS nets). Moreover the construction of the abstract net defines a coreflection, whose universal properties confirm that it is the optimal such construction and that it is unique (up to isomorphic nets).

Remark 1.2. The constructions can be pursued according to either the *CTph* or the *ITph*. The two approaches yield the same step relation but different abstract nets. Whatever the chosen approach is, the categorical constructions given above are both possible.

2 An Operational Definition for Transactions

The question that we want to address in this paper is essentially how one can implement ZS nets. Since we have already defined their operational semantics, the answer should not be very complicated. But the problem is that the operational semantics relies on some sort of meta-definition, where one computes on the underlying net, builds transaction segments, and then can discard undesired behaviors and accept the good ones, acting as a filter. This means that there are important questions which can be asked for any actual interpreter — Is backtracking necessary? Is the implementation correct? And complete? Does any more efficient implementation exist?

We try to answer these questions (see the Conclusions) by defining a machinery for computing on ZS nets and studying its properties. The idea is to have a level where suitable computations of the underlying net are simulated and examined, in such a way that as soon as a connected transaction is computed, then the corresponding abstract behavior is generated (e.g., it can be used to compute on the system).

The idea is to adapt the classical net unfolding, which is used to extract the event structure that precisely represents the abstract behavior of the net, to pursue concurrently all the nondeterministic runs of the ZS net under inspection, in such a way that "commit" states are recognized and generated.

Whether one is interested in distinguishing between different concurrent proofs or is just interested in the step relation $\_ \Rrightarrow_B \_$ is an important issue. For simplicity, we will consider the problem of computing the set of markings that can be reached in one transaction step, i.e., the set $\mathscr{R}(B,u) = \{v \mid u \Rrightarrow_B v\}$. However, the solution we propose can be easily adapted to compute the abstract net of B.

2.1 PT Net Unfolding

The unfolding of a net gives a constructive way to generate all the possible computations, offering a satisfactory mathematical description of the interplay between nondeterminism and causality (and concurrency). In fact the unfolding construction allows for bridging the gap between PT nets and *prime algebraic domains*, yielding a truly concurrent semantics. Some obvious references to this approach are [20,27,19], but we suggest also the interesting overview [15]. It is worth remarking that our presentation is slightly different from usual ones, since it is presented as the minimal net generated by suitable inference rules, rather than by making explicit the chain of finite nets that approximate it.

The construction provides a distributed interpreter for PT nets. We remark that the unfolding applies only to marked nets, i.e., it requires an initial marking.

Starting from a net N, the unfolding produces a nondeterministic occurrence net $\mathcal{U}(N)$ (an acyclic net, where transition pre- and post-markings are sets instead of multisets and where each place has at most one entering arc, i.e., backward conflicts are not allowed), together with a mapping from $\mathcal{U}(N)$ to N that tells which places and transitions of the unfolding are instances of the same element of N. Hence the places of $\mathcal{U}(N)$ represent the tokens and the transitions (called *events*) the occurrences of transitions in N in all possible runs of the net. For this kind of nets the notion of *causally dependent*, of *conflicting* and of *concurrent* elements can be straightforwardly defined and are represented by the binary relations $\_ \preceq \_$, $\_\#\_$ and $\mathbf{co}(\_,\_)$, respectively. Formally, the relation $\_ \preceq \_$ is the transitive and reflexive closure of the *immediate precedence* relation $\_ \prec_0 \_$ defined as

$$\prec_0 \stackrel{\text{def}}{=} \{(a,t) \mid a \in \text{pre}(t)\} \cup \{(t,a) \mid a \in \text{post}(t)\},$$

while the binary conflict relation is defined as the minimal symmetric relation that contains $\_\#_0\_$ defined below, and that is hereditary with respect to $\_ \preceq \_$ (i.e., such that if $o_1, o_2 \in S \cup T$ and $o_1 \# o_2$ and $o_1 \preceq o$ then also $o \# o_2$).

$$t_1 \#_0 t_2 \stackrel{\text{def}}{\Leftrightarrow} t_1 \neq t_2 \;\wedge\; \text{pre}(t_1) \cap \text{pre}(t_2) \neq \emptyset$$

Since the conflict relation is required to be irreflexive, then $\_ \preceq \_$ and $\_\#\_$ have empty intersection. The concurrency relation is defined by letting $\mathbf{co}(o_1, o_2)$ if it is not the case that ($o_1 \prec o_2$ or $o_2 \prec o_1$ or $o_1 \# o_2$). In particular, the relation $\mathbf{co}$ is usually extended to generic sets of elements by writing $\mathbf{co}(X)$ if and only if for all $o_1, o_2 \in X$ we have $\mathbf{co}(o_1, o_2)$.

More concretely, the places of $\mathcal{U}(N)$ have the form $\langle a, n, H \rangle$, where $a \in S_N$, n is a positive natural number that is used to distinguish different tokens with the same history, and H is the history of the place under inspection and therefore either consists of just one event (the one that produced the token) or is empty (if the token is in the initial marking). Analogously, a generic transition of $\mathcal{U}(N)$ has the form $\langle t, H \rangle$ with $t \in T_N$, since each transition is completely identified by its history H, which in this case consists of the set of consumed tokens. The set H cannot be empty since transitions with empty pre-set are not allowed. The net $\mathcal{U}(N)$ is defined as the minimal net generated by the rules in Table 3.

$$\frac{u_{\mathrm{in}}(a) = n,\ 1 \leq k \leq n}{\langle a, k, \emptyset \rangle \in S_{\mathcal{U}(N)}}$$

$$\frac{t \colon u \to \bigoplus_{j \in J} n_j b_j \in T_N,\ \Theta = \{\langle a_i, k_i, H_i \rangle \mid i \in I\} \subseteq S_{\mathcal{U}(N)},\ \mathbf{co}(\Theta),\ u = \bigoplus_{i \in I} a_i}{e = \langle t, \Theta \rangle \in T_{\mathcal{U}(N)},\ \Upsilon = \{\langle b_j, m, \{e\} \rangle \mid j \in J,\ 1 \leq m \leq n_j\} \subseteq S_{\mathcal{U}(N)},\ \mathrm{pre}(e) = \Theta,\ \mathrm{post}(e) = \Upsilon}$$

Table 3: The unfolding $\mathcal{U}(N)$.

We now give a computational interpretation of such rules. The first rule defines the initial marking of $\mathcal{U}(N)$. The second rule is the core of the unfolding: it looks for a set Θ of concurrent tokens that enables a transition t of N, atomically locks them, fires the event e (that is an occurrence of t), and produces some fresh tokens according to $\mathrm{post}(t)$. Notice that the condition $\mathbf{co}(\Theta)$ depends exclusively on the histories H_i for $i \in I$ and therefore cannot be altered by successive firings. In fact, as in *memoizing* for logic programming, or more generally in *dynamic programming*, the history is completely encoded in the tokens, so that it is not necessary to compute it at every firing. Also note that histories retain concurrent information rather than just sequential, therefore each token/event is generated exactly once (though it can be referred many times successively). Moreover, several occurrences of the second rule can be applied concurrently and therefore the unfolding can be implemented as a distributed algorithm.

2.2 ZS Net Unfolding

The unfolding of the underlying net N_B does not yield a faithful representation of the behavior of B. In fact, we must forbid the consumption of stable resources that were not inserted in the initial marking.

Example 2.1. Let us consider the ZS net B in Figure 5. It has three stable places a, b and c, a zero place z and two transitions t_1 and t_2, with

– $\mathrm{pre}(t_1) = a$ and $\mathrm{post}(t_1) = b \oplus z$;
– $\mathrm{pre}(t_2) = b \oplus c \oplus z$ and $\mathrm{post}(t_2) = 0$.

Then, if $u_{\mathrm{in}} = a \oplus c$, the net B cannot perform any transaction, because although t_1 is enabled and thus can fire, transition t_2 must be used to end the transaction consuming the fresh token in z, but this would require also the consumption of the token in b generated by the firing of t_1, which is not available before the commit. Hence $\mathscr{R}(B, a \oplus c) = \{a \oplus c\}$. Instead, if we add an initial token in b, then such token can be used to close the transaction, by firing t_2 after t_1, and therefore $\mathscr{R}(B, a \oplus b \oplus c) = \{a \oplus b \oplus c, b\}$.

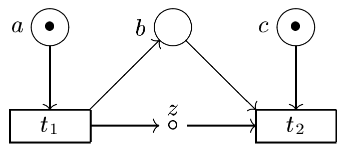

Figure 5. The ZS net of Example 2.1.

Thus, to some extent, we are unfolding on zero places only. Moreover, we must be able to apply the commit when the transaction step has consumed all the zero tokens produced so far.

The net $\mathcal{U}(B)$ is defined as the minimal net generated by the rules in Table 4. Together with the unfolding net we compute the set of reachable markings $\mathscr{R}(B, u)$ for a generic stable marking u.

$$\frac{u(a) = n,\ 1 \leq k \leq n}{\langle a, k, \emptyset\rangle \in S_{\mathcal{U}(B)}}$$

$$\frac{t\colon (u, x) \to (v, \bigoplus_{j \in J} n_j z_j) \in T_B,\ \Theta = \{\langle s_i, k_i, H_i\rangle \mid i \in I\} \subseteq S_{\mathcal{U}(B)},\ \mathbf{co}(\Theta),\ u \oplus x = \bigoplus_{i \in I} s_i}{e = \langle t, \Theta\rangle \in T_{\mathcal{U}(B)},\ \Upsilon = \{\langle z_j, m, \{e\}\rangle \mid j \in J,\ 1 \leq m \leq n_j\} \subseteq S_{\mathcal{U}(B)},\ \mathrm{pre}(e) = \Theta,\ \mathrm{post}(e) = \Upsilon}$$

$$\frac{}{u \in \mathscr{R}(B, u)} \qquad \frac{\Gamma \subseteq T_{\mathcal{U}(B)},\ \mathbf{co}(\Gamma),\ \mathbf{ZProd}(\Gamma) = \mathbf{ZCons}(\Gamma)}{u \ominus \mathbf{SCons}(\Gamma) \oplus \mathbf{SProd}(\Gamma) \in \mathscr{R}(B, u)}$$

Table 4: The unfolding $\mathcal{U}(B)$.

The first two rules define the unfolding, which remains similar to the classical algorithm, except for the fact that stable tokens in the post-set of the fired transition are not released to the system. In fact, while the set Θ must contains enough tokens to provide both the stable and the zero resources needed by t (as expressed by the condition $u \oplus x = \bigoplus_{i\in I} a_i$), the tokens that are produced by the occurrence of t applied to Θ (i.e., tokens in the set $\Upsilon = \mathrm{post}(e)$) just match the zero place component $\bigoplus_{j\in J} n_j z_j$ of $\mathrm{post}(t)$ and not the stable place component v. The marking v will not be released until a commit related to e will occur.

The third rule is obvious. The fourth rule defines the commit of a transaction step. To shorten the notation, we introduce the following functions that given an event e return the set of zero tokens respectively consumed and produced by the ancestors of e (and by e itself), i.e., we let

$$\mathbf{ZCons}(e) \stackrel{\mathrm{def}}{=} \bigcup_{\langle t,\Theta\rangle \preceq e} \{\langle z,k,H\rangle \in \Theta \mid z \in Z_B\}$$

$$\mathbf{ZProd}(e) \stackrel{\mathrm{def}}{=} \bigcup_{e' \preceq e} \mathrm{post}(e')$$

where $\mathbf{ZCons}(e)$ is the set of zero tokens that have been consumed by some $e' \preceq e$; similarly $\mathbf{ZProd}(e)$ represents the set of zero tokens that have been produced by some $e' \preceq e$ (note that for any $\langle z,k,H\rangle \in \mathbf{ZCons}(e)$ we have $\langle z,k,H\rangle \preceq e$, while $\mathbf{ZProd}(e)$ can also contain tokens that are concurrent with e or produced by e). We remark that $\mathbf{ZCons}(e) \subseteq \mathbf{ZProd}(e)$, because the marking u is stable and therefore does not contain zero tokens with empty histories. For stable places the situation is different, since we are just interested in knowing how many tokens have been consumed and will be produced for each places by the antecedents of e, thus

$$\mathbf{SCons}(e) \stackrel{\mathrm{def}}{=} \bigoplus_{\langle t:(u,x)\to(v,y),\Theta\rangle \preceq e} u$$

$$\mathbf{SProd}(e) \stackrel{\mathrm{def}}{=} \bigoplus_{\langle t:(u,x)\to(v,y),\Theta\rangle \preceq e} v.$$

The four functions that we have defined are extended to sets of events in the obvious way.

The fourth rule takes a set Γ of concurrent events and checks that the zero tokens produced by each event are consumed by an antecedent of some other event in Γ. The latter condition can be conveniently expressed as the equality $\mathbf{ZProd}(\Gamma) = \mathbf{ZCons}(\Gamma)$. In fact, if a certain token o is in $\mathbf{ZProd}(\Gamma)$, then the condition states that there exists at least an event $e \in \Gamma$ and a unique[2] $e' \preceq e$ such that $o \in \mathrm{pre}(e)$. If these premises are satisfied, then the rule extends $\mathscr{R}(B,u)$ with the multiset obtained by subtracting from u the stable resources consumed by all the antecedents of events in Γ, but adding those that would have been produced during the step. This rule defines a commit since it synchronizes local commits, as the following result shows.

[2] Otherwise a conflict would arise.

Proposition 2.1. *If $\Gamma \subseteq T_{\mathcal{U}(B)}$ such that* $\mathbf{co}(\Gamma)$ *and* $\mathbf{ZProd}(\Gamma) = \mathbf{ZCons}(\Gamma)$*, then for any $e = \langle t, \Theta \rangle \in \Gamma$ we have that t does not produce any zero token.*

Proof. We proceed by contradiction. Let us assume that $t\colon (u, x) \to (v, y)$ for some stable markings u, v and zero markings x and y, with $y \neq 0$. Since $e \in \Gamma \subseteq T_{\mathcal{U}(B)}$, it must have been generated by the second rule in Table 4. Hence $\Upsilon = \mathrm{post}(e) \neq \emptyset$. Let o be a token in Υ. Since $\Upsilon \subseteq \mathbf{ZProd}(\Gamma)$ and $\mathbf{ZProd}(\Gamma) = \mathbf{ZCons}(\Gamma)$ by hypothesis, it follows that there exist an event $e' \neq e$ in Γ and an event $e'' \preceq e'$ such that $o \in \mathrm{pre}(e'')$. But then we have $e \preceq o \preceq e'' \preceq e'$ with $e, e' \in \Gamma$, that contradicts the hypothesis $\mathbf{co}(\Gamma)$, concluding the proof. $\square$

The resulting algorithm is as much distributed as the classical one when applied to the abstract net of B. In fact all the useful relations are defined by just looking at the history of the elements in the premises, which, under the atomicity assumption reduce to the stable preset of the abstract step. To improve efficiency, the sets $\mathbf{ZProd}(e)$, $\mathbf{ZCons}(e)$, $\mathbf{SProd}(e)$ and $\mathbf{SCons}(e)$ could be also encoded in e more directly, although they can be easily calculated from the history component.

The main result can be formulated as the following theorem.

Theorem 2.1. $\mathscr{R}(B, u) = \{v \mid u \Rrightarrow_B v\}$.

Proof. (sketch) The inclusion $\mathscr{R}(B, u) \subseteq \{v \mid u \Rrightarrow_B v\}$ can be proved by rule induction. In fact, if $w \in \mathscr{R}(B, u)$, then either $u = w$ and we can apply the rules **identities**, **underlying** and **commit** to conclude that $u \in \{v \mid u \Rrightarrow_B v\}$, or $w = u \ominus \mathbf{SCons}(\Gamma) \oplus \mathbf{SProd}(\Gamma)$ for some set $\Gamma \subseteq T_{\mathcal{U}(B)}$, such that $\mathbf{co}(\Gamma)$ and $\mathbf{ZProd}(\Gamma) = \mathbf{ZCons}(\Gamma)$, and therefore there exists a concurrent computation for which Γ is a slice, that starts from the marking $\mathbf{SCons}(\Gamma)$ included in u and that consumes all the zero tokens that produces. Moreover, such computation is completely described by the events in

$$C = \{e' \mid \exists e \in \Gamma,\ e' \preceq e\}.$$

By summing up all the stable tokens produced by the transitions associated to the events in C we obtain exactly $\mathbf{SProd}(\Gamma)$. Therefore we have

$$(\mathbf{SCons}(\Gamma), 0) \rightrightarrows_B (\mathbf{SProd}(\Gamma), 0).$$

By the rules **identities** and **underlying** we have also

$$(u \ominus \mathbf{SCons}(\Gamma), 0) \rightrightarrows_B (u \ominus \mathbf{SCons}(\Gamma), 0).$$

Horizontally composing the two we get

$$(u, 0) \rightrightarrows_B (u \ominus \mathbf{SCons}(\Gamma) \oplus \mathbf{SProd}(\Gamma), 0).$$

Recalling that $w = u \ominus \mathbf{SCons}(\Gamma) \oplus \mathbf{SProd}(\Gamma)$ and applying the rule **commit** we obtain $u \Rrightarrow w$, that proves the inclusion.

Conversely, to prove that $\{v \mid u \Rrightarrow_B v\} \subseteq \mathscr{R}(B, u)$, we proceed by induction on the proof of $u \Rrightarrow_B w$, by mimicking the firings in the unfolding of the net. $\square$

3 Related Issues

3.1 Computing the Abstract Net

Since the unfolding encodes the proof of the transaction step (via the history components), it is possible to use the same scheme for computing the abstract net (whatever philosophy is preferred). However, for doing this efficiently, we must be able to recognize isomorphic processes. For example, note that given a certain computed process, any renaming of the stable tokens in the initial marking (i.e., any permutation of tokens in the same place) yields a different but isomorphic process that is also calculated during the unfolding. To solve this problem, we can either try to avoid having several isomorphic processes in the unfolding by some clever construction, or just check if a freshly computed transaction is isomorphic to some transaction already computed when executing the commit.

In general, the problem of checking the equivalence of two (finite, deterministic) processes can be exponential on the process size, the problem being given by multiple copies of the initial resources and by multiple instances of a resource produced by the same transition, making it hard to establish the correspondence between analogous elements of the two processes. However, for 1-safe computations the problem is clearly linear (the correspondence, if it exists, is the obvious one). The problem is also linear for processes of *semi-weighted nets*[3] when the initial marking is a set.

3.2 Some Considerations on Computability

Since a ZS net B can contain cycles that produce an unbounded number of zero tokens, the unfolding can become infinite.

Example 3.1. Let B be the ZS net (see Figure 6) with stable places a and b, $u_{\text{in}} = a$, and three transitions t_1, t_2 and t_3, with

- $\text{pre}(t_1) = a$ and $\text{post}(t_1) = z$;
- $\text{pre}(t_2) = z$ and $\text{post}(t_2) = 2z$;
- $\text{pre}(t_3) = z$ and $\text{post}(t_3) = b$.

Then $\mathscr{R}(B, a) = \{a\} \cup \{nb \mid n > 0\}$. In fact any transaction can be started by a firing of t_1, then t_2 can fire any number of times and t_3 can fire as many times as t_2 plus one. Moreover, note that since the net is not semi-weighted (because of t_2), many isomorphic processes will be computed during the unfolding.

So an important question concerns the decidability of $\mathscr{R}(B, u)$. In a private communication to the authors [14], Nadia Busi proved that such set is indeed decidable. Roughly speaking the idea is to simulate the behaviors of B by a PT

[3] A semi-weighted net is a net where the post-set of transitions are sets instead of multisets, whereas pre-sets can still be multisets.

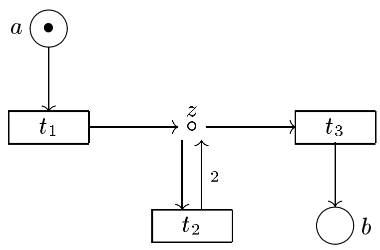

Figure 6. The ZS net of Example 3.1.

net with exactly one inhibitor arc,[4] for which the reachability problem has been solved in [22].

The set $\mathscr{R}(B, u)$ is recursively enumerated by the inference system, and if it is infinite we cannot do any improvement. But when $\mathscr{R}(B, u)$ is finite, it would be desirable to find some condition for halting the execution of the algorithm, that otherwise could continue computing transaction segments that cannot be completed.

3.3 Halting Problem for the Distributed Algorithm

Finding some general condition for halting the unfolding of ZS is an open problem that we leave for future investigations. Of course, in many cases, some very simple criterion can be applied. The problematic cases are given by cycles that do not involve stable places. If the net under inspection contain no such cycle, then the algorithm will eventually terminate. Now suppose that the net contains an increasing cycle of that kind for the zero place z, and observe that there is no decreasing cycle and that all other transitions that consumes some zero tokens from z, except the one involved in the cycle, consumes some stable token, then we can remember that the stable resources are bounded and therefore we have a bound for the tokens that can be consumed from z. Since we only consider transactions that lead to stable markings, such a bound can be put also on the zero tokens that can be created in z. It is worth noticing that these criteria are sufficient to deal with the examples of ZS nets we encountered in our previous work.

Conclusions

We have proposed the framework of zero-safe nets as a basis for modeling and implementing distributed transactions. In fact, ZS nets can provide both the

[4] Nets with inhibitor arcs, also called with negative arcs, have been introduced in [1,2] for modeling systems where the presence of certain resources can inhibit the firing of some transitions. We recall that the reachability problem is undecidable for the class of nets with two or more inhibitor arcs.

refined view of the systems where actions have finer grain and an abstract view where transactions are seen just as transitions of an ordinary PT net. Working at the level of ZS nets is very important since it allows to keep smaller the size of the system description (for example the abstract net can have an infinite number of transitions also when the refined net is finite).

We have extended previous semantic investigations with the definition of a distributed interpreter for computing on ZS nets. We want to remark that the resulting implementation does not violate the locality assumptions, since it is completely analogous to the widely accepted implementation for PT nets.

We can now answer to the questions formulated at the beginning of Section 2. Thus backtracking is not necessary and the correctness of the interpreter is given by Theorem 2.1. Completeness is ensured by our inference system. We are confident to find a nice formalization of halting criteria for an expressive class of ZS nets.

References

1. T. Agerwala. A complete model for representing the coordination of asynchronous processes. Hopkins Computer Research Report 32, John Hopkins University, 1974.
2. T. Agerwala and M. Flynn. Comments on capabilities, limitations and "correctness" of Petri nets. *Computer Architecture News*, 4(2):81–86, 1973.
3. E. Best and R. Devillers. Sequential and concurrent behaviour in Petri net theory. *Theoret. Comput. Sci.*, 55:87–136, 1987.
4. E. Best, R. Devillers, and J. Esparza. General refinement and recursion for the Petri Box calculus. In *Proceedings STACS'93*, volume 665 of *Lect. Notes in Comput. Sci.*, pages 130–140. Springer-Verlag, 1993.
5. E. Best, R. Devillers, and J. Hall. The Box calculus: A new causal algebra with multi-label communication. In G. Rozenberg, editor, *Advances in Petri Nets'92*, volume 609 of *Lect. Notes in Comput. Sci.*, pages 21–69. Springer-Verlag, 1992.
6. W. Brauer, R. Gold, and W. Vogler. A survey of behaviour and equivalence preserving refinements of Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets'90*, volume 483 of *Lect. Notes in Comput. Sci.*, pages 1–46. Springer-Verlag, 1991.
7. R. Bruni. *Tile Logic for Synchronized Rewriting of Concurrent Systems.* PhD thesis TD-1/99, Computer Science Department, University of Pisa, 1999.
8. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. A comparison of petri net semantics under the collective token philosophy. In J. Hsiang, A. Ohori, editors, *Proceedings ASIAN'98, 4th Asian Computing Science Conference*, volume 1538 of *Lect. Notes in Comput. Sci.*, pages 225–244. Springer Verlag, 1998.
9. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial semantics for Petri nets under the individual token philosophy. In M. Hofmann, G. Rosolini, D. Pavlovic, editors, *Proceedings CTCS'99, 8th conference on Category Theory and Computer Science*, volume 29 of *Elect. Notes in Th. Comput. Sci.*, Elsevier Science, 1999.
10. R. Bruni and U. Montanari. Zero-safe nets, or transition synchronization made simple. In C. Palamidessi and J. Parrow, editors, *Proceedings EXPRESS'97, 4th workshop on Expressiveness in Concurrency*, volume 7 of *Elect. Notes in Th. Comput. Sci.*, Elsevier Science, 1997.

11. R. Bruni and U. Montanari. Zero-safe nets: The individual token approach. In F. Parisi-Presicce, editor, *Proceedings WADT'97, 12th workshop on Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lect. Notes in Comput. Sci.*, pages 122–140. Springer-Verlag, 1998.
12. R. Bruni and U. Montanari. Zero-safe nets: Comparing the collective and individual token approaches. *Inform. and Comput.*, 156:46–89. Academic Press, 2000.
13. R. Bruni and U. Montanari. Zero-safe nets: Composing nets via transition synchronization. In H. Weber, H. Ehrig, and W. Reisig, editors, *Proceedings Int. Colloquium on Petri Net Technologies for Modelling Communication Based Systems*, pages 43–80. Fraunhofer Gesellschaft ISST, 1999.
14. N. Busi. On zero safe nets, April 1999. Private communication.
15. R.J. van Glabbeek. Petri nets, configuration structures and higher dimensional automata. In J.C.M. Baeten and S. Mauw, editors, *Proceedings CONCUR'99, 10th International Conference on Concurrency Theory*, volume 1664 of *Lect. Notes in Comput. Sci.*, pages 21–27. Springer-Verlag, 1999.
16. R.J. van Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. Hildesheimer Informatik Bericht 6/98, Institut fuer Informatik, Universitaet Hildesheim, 1998.
17. R.J. van Glabbeek and G.D. Plotkin. Configuration structures. In D. Kozen, editor, *Proceedings LICS'95, 10th Annual IEEE Symposium on Logic In Computer Science*, pages 199–209. IEEE Computer Society Press, 1995.
18. U. Goltz and W. Reisig. The non-sequential behaviour of Petri nets. *Inform. and Comput.*, 57:125–147. Academic Press, 1983.
19. J. Meseguer, U. Montanari, and V. Sassone. Process versus unfolding semantics for place/transition Petri nets. *Theoret. Comput. Sci.*, 153(1-2):171–210, 1996.
20. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoret. Comput. Sci.*, 13:85–108, 1981.
21. C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.
22. K. Reinhardt. Reachability in Petri nets with inhibitor arcs. Technical Report WSI-96-30, Wilhelm Schickard Institut für Informatik, Universität Tübingen, 1996.
23. W. Reisig. *Petri Nets: An Introduction.* EACTS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.
24. I. Suzuki and T. Murata. A method for stepwise refinement and abstraction of Petri nets. *J. Comput. and System Sci.*, 27:51–76, 1983.
25. R. Valette. Analysis of Petri nets by stepwise refinement. *J. Comput. and System Sci.*, 18:35–46, 1979.
26. W. Vogler. Behaviour preserving refinements of Petri nets. In G. Tinhofer and G. Schmidt, editors, *Proceedings 12th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 246 of *Lect. Notes in Comput. Sci.*, pages 82–93. Springer-Verlag, 1987.
27. G. Winskel. Event structures. In W. Brauer, editor, *Proceedings Advanced Course on Petri Nets*, volume 255 of *Lect. Notes in Comput. Sci.*, pages 325–392. Springer-Verlag, 1987.