Electronic Notes in Theoretical Computer Science 54 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume54.html 11 pages

Zero-safe net models for transactions in Linda [1]

Roberto Bruni and Ugo Montanari [2]

Computer Science Department, University of Pisa, 56125 Pisa, Italy

Abstract

Zero-safe nets are a variation of Petri nets, where transactions can be suitably modeled. The idea is to distinguish between stable places (whose markings define observable states) and zero-safe places (where tokens can only be temporarily allocated, defining hidden states): Transactions must start and end in observable states. We propose an extension of the coordination language Linda, called TraLinda, where a few basic primitives for expressing transactions are introduced by means of different typing of tuples. By exploiting previous results of Busi, Gorrieri and Zavattaro on the net modeling of Linda-like languages, we define a concurrent operational semantics based on zero-safe nets for TraLinda, where the typing of tuples reflects evidently on the distinction between stable and zero-safe places.

Key words: transactions, Petri nets, zero-safe nets, coordination, Linda

Introduction

Place/transition Petri nets (PT nets) are a basic model for concurrent and distributed systems where two fundamental design postulates are exploited:

(i) states are multisets (of typed resources, i.e. tokens); and

(ii) elementary actions (transition firings) can atomically fetch (and release) several state components, thus synchronizing tokens at the event level.

A main advantage of these assumptions is that the notion of concurrent firing (*step*) comes as a consequence of the multiset structure of states, i.e., computations can be straightforwardly equipped with a truly concurrent semantics (as opposed to interleaving semantics). Moreover, building on (i) and (ii), it is possible to define a taxonomy of models that encompasses some limitations of the basic paradigm:

- *nets with read arcs* (r-nets) allow for modeling "read without consume," where many readers can access concurrently the same resource (whereas in ordinary PT nets these operations are usually rendered via sequentializing self-loops);

[1] Research supported by the TMR Network GETGRATS and by the MURST Project TOSCA.

[2] Email: {bruni,ugo}@di.unipi.it

- *zero-safe nets* (ZS nets) introduces "transition synchronization" (as opposed to token synchronization discussed in point (ii) above), hence regarding suitable finite, but possibly complex and concurrent, computations as atomic events;
- *nets with inhibitor arcs* (i-nets) allow for testing the "absence of tokens," an operation that makes the framework Turing complete (as opposed to the models discussed so above, where reachability and deadlock are decidable).

Moreover: *colored* or *high-level* nets allows for using "structured data" as tokens; in *reconfigurable* nets, transition postsets may depend on the values got from the presets (whereas in all the above models connections are static), addressing network reconfigurability; in *dynamic* nets, not only a firing can modify the current marking, but can also increase the set of transitions (i.e., the control). The different flavors of nets are too many to be mentioned here (e.g., with time, probability, priorities). An interesting correspondence between some of the above models and typed subclasses of Join agents [15] has been hinted at in [1] and formally drawn in [8]. Note that certain features are to some extent orthogonal and can be mixed together with a minimum effort. For example, read and inhibitor arcs can be easily combined together to model a general concept of positive/negative *context* in *ri-nets*,[3] while in [6] we introduced read arcs inside the zero-safe framework.

Points (i–ii) are also at the basis of the coordination language Linda [16], which represents the distributed environment as a *tuple space* (a multiset-based store where structured data are allowed), with agents built on primitives for adding tuples to and retrieving tuples from the store, reading without consuming, and testing for presence of a particular tuple. These operations make Linda suitable for expressing non-atomic process coordination in a concurrent setting. When tuples are seen as non-structured data, the concurrent semantics of Linda can be suitably modeled via ri-nets [10,9]. In particular, it is interesting the distinction between the *ordered* and *unordered* semantics of Linda-like process algebras drawn in [12,11,9]: in the former semantics, the output of a tuple is seen as the atomic execution of the emission of the message followed by its rendering in the tuple space (making it available to other agents), whereas in the latter view these two phases are made autonomous (emitted messages can be rendered asynchronously). This distinction reflects on the expressivity of the language: the ordered semantics originates a Turing complete language, whilst, in the unordered case, deadlock is decidable [12].

Building on these representation results, our aim is twofold: on the one hand we want to extend the zero-safe approach to handle inhibitor arcs, and on the other hand, the nets obtained in this way can find straightforward application to the concurrent modeling of Linda-like languages equipped with a primitive mechanism for transactions (along either the ordered or unordered semantics). In fact, though ad-hoc transaction mechanisms are integrated in languages such as BizTalk Orchestration and JavaSpaces, we look for a uniform treatment of transactions for the many calculi proposed in the literature.

[3] In the literature, the terms *contextual nets* and *c-net* are abused in denoting either just positive contexts or both kinds of contexts; to avoid confusion, we stick to the unambiguous notation *ri-net*.

Transaction as transition synchronization. While PT nets rely on place synchronization, they lack synchronization of transitions, which would allow to regard some (finite, but possibly complex and concurrent) computations as atomic events. Transition synchronization is e.g. essential in modeling distributed decision algorithms, where no particular location can implement a choice point. Our observation is probably not surprising, since the necessity of defining entities in isolation that can cooperate by sending and receiving information, but are otherwise seen like black boxes from the environment, was already evident in Dijkstra communicating processes and in process algebras. Transition synchronization is also a key aspect of open ended systems: When designing the processes in isolation, the programmer must consider all the interactions that can take place, but these may well depend on the received data and may require some atomicity assumptions on their execution (think e.g., of communication protocols, with transmission requests and acks, or of e-commerce applications, where customer's payment and goods/service delivery must be either both guaranteed or both canceled when the transaction ends).

We denote atomic computations in a concurrent scenario by the term *transactions* (abusing databases terminology). When adding transactions, two main issues must be handled: *semantics*, i.e. a theoretical characterization of transactions, making it possible to study their properties and the way in which they can be combined together (e.g., in parallel, sequentially); and *algorithms*, i.e. the development of distributed interpreters able to implement transactions, consistently with the semantic level (e.g., using backtrack). Hence, it is convenient to select a formal language where these issues can be easily dealt with also at the syntax level. The solution proposed in [5] tries to make these intuitions concrete. In fact, the simplest way to synchronize the execution of transitions in PT nets is via token exchanging over (a subset of) places. ZS nets exploit this idea by distinguishing between *stable* places (the ordinary repositories for resources, defining observable states) and *zero safe* places that cannot contain tokens in any observable state. The firing of a transition will possibly put tokens in zero-safe places, beginning a transaction; these tokens (called *zero tokens*) are used to coordinate the transaction. All zero tokens must be removed to commit the transaction. Moreover, all the stable tokens produced during the transaction are effectively released only when the transaction ends. Thus, all the stable resources fetched during the transaction must be present in the initial stable marking. This describes, to some extent, the low-level model.

At the abstract level, transactions must be seen as ordinary transitions. This viewpoint yields a PT net A_B, which is the abstract counterpart of the ZS net B: the places of N are the stable places of B, and each transition of A_B corresponds to an elementary transaction of B (i.e., a transaction that cannot be decomposed in two smaller disjoint transactions). Note that A_B can become infinite also when B is finite, and that transactions retain all the causal and concurrent information about the synchronized evolution of B. Moreover, a distributed interpreter for ZS nets has been proposed in [4], which is based on the ordinary net unfolding. Hence, ZS nets can be used, e.g. to give a modular presentation of distributed decision making, as any net can be modeled as the abstract counterpart of a free choice ZS net [4].

Transactions in Linda. Linda is the most representative language for coordination, where asynchronous communication is obtained by inserting, reading and withdrawing elements to and from a shared multiset of tuples. In a way which is analogous to the use of positive/negative contexts in nets, Linda also allows for testing the presence/absence of messages in the tuple space. Its communication mechanism is called *generative* because, once generated and until removed, each message become equally accessible to all processes without being bound to a particular one. However, no primitive for expressing transactions is in the core language (but see e.g., the extension proposed in [13] for studying *serializability* in JavaSpaces).

Thanks to the analogy with Petri nets (tuples as tokens), the idea is to provide a way for programming transactions by distinguishing a class of *low-level* messages whose only role is to coordinate the exchange of services or information between agents. We call *zero-safe* these messages, because it only makes sense for them to exist inside a transaction, i.e., their lifetime lasts as long as necessary for their producers and consumers to agreeing on some decision. The remaining messages are called *stable* to make clear that the information they contain contributes substantially to the actual configuration of the system.

As a simple example, let us consider the modeling of multiset rewriting: An agent P needs all the messages $a_1,...,a_n$, regardless of the order in which they are consumed, to start a task Q. Of course, we look for a simple way of specifying that no process P' can in the meantime fetch the resources $a_1,...,a_n$, deadlocking P. The idea is to write P as the parallel composition of $in(a_1)\_out(p_Z),...,in(a_n)\_out(p_Z)$ and $in(p_Z)\_...\_in(p_Z)\_Q$, where: all a_i are stable, p_Z is zero-safe, in is the input operation, out is the output operation and $\_$ denotes the atomic prefixing, with Q prefixed n times by $in(p_Z)$. Then, the transaction starts as soon as any a_i is consumed and the atomic prefixing force the output of a zero-safe message p_Z; moreover, the transaction can only commit when n messages p_Z have been produced, which the atomic prefix of Q can consume. Note that Q cannot be activated unless all a_i have been retrieved. We call TraLinda the language obtained by extending the kernel of Linda with the primitives for transactions, which are partially described above.

Structure of the paper.
In Section 1, we recall the basics of Petri nets with read arcs, inhibitor arcs and zero-safe places, together with the syntax and the concurrent semantics of Linda. Section 2 introduces inhibitor arcs in the zero-safe approach, while Section 3 defines TraLinda and its concurrent semantics based on the nets discussed in Section 2.

1 Background

1.1 Petri nets

A *net* is a triple $N = (S_N, T_N, \mathsf{F}_N)$, where S_N is the set of *places* $a, a', \ldots$, T_N is the set of *transitions* $t, t', \ldots$ (with $S_N \cap T_N = \varnothing$), and $\mathsf{F}_N \subseteq (S_N \times T_N) \cup (T_N \times S_N)$ is called the *flow relation*.

$$\frac{u \in S^{\oplus}}{u \xrightarrow{\lfloor u \rfloor}_R u} \qquad \frac{t{:}\, u \xrightarrow{w} v \in T}{u \oplus w \xRightarrow{w}_R v \oplus w} \qquad \frac{u_1 \oplus w_1 \xRightarrow{w_1}_R v_1 \oplus w_1,\ u_2 \oplus w_2 \xRightarrow{w_2}_R v_2 \oplus w_2}{u_1 \oplus u_2 \oplus \lfloor w_1 \oplus w_2 \rfloor \xRightarrow{\lfloor w_1 \oplus w_2 \rfloor}_R v_1 \oplus v_2 \oplus \lfloor w_1 \oplus w_2 \rfloor}$$

Figure 1. The inference rules for $\_ \xRightarrow{\_}_R \_$.

The elements of F_N are called *arcs*, and we write $x\, \mathsf{F}_N\, y$ for $(x,y) \in \mathsf{F}_N$. We shall denote $S_N \cup T_N$ by N and omit subscripts when no confusion can arise. As usual, the *pre-* and *postset* of $x \in N$, are ${}^\bullet x = \{y \in N \mid y\, \mathsf{F}\, x\}$ and $x^\bullet = \{y \in N \mid x\, \mathsf{F}\, y\}$. We assume ${}^\bullet t \neq \varnothing$ for all $t \in T$.

A *marking* $u{:}\, S \to \mathbb{N}$ is a finite multiset of places. It can be written either as $u = \{n_1 a_1, \ldots, n_k a_k\}$ where each n_i dictates the number of *tokens* in a_i (if $n_i = 0$ then $n_i a_i$ is omitted), or as the formal sum $u = \bigoplus_{a_i \in S} n_i a_i$ denoting an element of the free commutative monoid $S^{\oplus}$ on S (the monoidal operation is defined by $(\bigoplus_i n_i a_i) \oplus (\bigoplus_i m_i a_i) = (\bigoplus_i (n_i + m_i) a_i)$ with 0 as the unit). We shall overload the symbol $\subseteq$ to denote multiset inclusion.

A *marked place/transition Petri net* (PT net) is a tuple $N = (S, T, \mathsf{F}, \mathsf{W}, u_{\text{in}})$ such that (S, T, F) is a net, the function $\mathsf{W}{:}\, \mathsf{F} \to \mathbb{N}$ assigns a positive *weight* to each arc in F, and the finite multiset $u_{\text{in}}{:}\, S \to \mathbb{N}$ is the *initial marking* of N.

We find convenient to view F as a function $\mathsf{F}{:}\, (S \times T) \cup (T \times S) \to \{0, 1\}$, with $x\, \mathsf{F}\, y \iff \mathsf{F}(x,y) \neq 0$. Then, for PT nets we can represent both F and W as the *multiset relation* $\mathsf{F}{:}\, (S \times T) \cup (T \times S) \to \mathbb{N}$. For any transition $t \in T$, let $\mathrm{pre}(t)$ and $\mathrm{post}(t)$ be the multisets over S such that $\mathrm{pre}(t)(a) = \mathsf{F}(a,t)$ and $\mathrm{post}(t)(a) = \mathsf{F}(t,a)$, for all $a \in S$. We write $t{:}\, u \to v$ for a transition t with $\mathrm{pre}(t) = u$ and $\mathrm{post}(t) = v$.

A marked *net with read arcs* (*r-net*) is a tuple $R = (S, T, \mathsf{F}, \mathsf{R}, u_{\text{in}})$, where $N_R = (S, T, \mathsf{F}, u_{\text{in}})$ is the underlying PT net and $\mathsf{R}{:}\, S \times T \to \{0, 1\}$ is the *context relation*. Though there are no technical difficulties in dealing with context multirelations (see e.g. [6]), for the current presentation is simpler to consider just context relations, having in mind the *maximum sharing hypothesis* of [7]. We denote by $\mathrm{ctx}(t)$ the (multi)set of places defined by $\mathrm{ctx}(t)(a) = \mathsf{R}(a,t)$ for all $a \in S$ and by $\lfloor u \rfloor$ the set $\{a \mid u(a) > 0\}$ of non-empty places of u. The minimum amount of resources that t requires to be enabled is $\mathrm{pre}(t) \oplus \mathrm{ctx}(t)$: The tokens in $\mathrm{pre}(t)$ are fetched, while those in $\mathrm{ctx}(t)$ are just read, and other transitions can access them, concurrently with t. For $t \in T$ with $\mathrm{pre}(t) = u$, $\mathrm{post}(t) = v$ and $\mathrm{ctx}(t) = w$, we write $t{:}\, u \xrightarrow{w} v$.

For X a multiset of transitions, and u a marking, we say that X is enabled at u if $\lfloor \bigoplus_{t \in X} \mathrm{ctx}(t) \rfloor \oplus \bigoplus_{t \in T} X(t) \cdot \mathrm{pre}(t) \subseteq u$. We say that u evolves to the marking v via X, written $u\, [X\rangle\, v$, if X is enabled at u and $v = u \oplus \bigoplus_{t \in T} X(t) \cdot \mathrm{post}(t) \ominus \bigoplus_{t \in T} X(t) \cdot \mathrm{pre}(t)$, with $\ominus$ denoting multiset difference. Note that if u has enough tokens to satisfy also the 'context' of X, then v is obtained from u just by removing $\bigoplus_{t \in T} X(t) \cdot \mathrm{pre}(t)$ and then adding $\bigoplus_{t \in T} X(t) \cdot \mathrm{post}(t)$.

The step relation can be equivalently defined by the inference rules in Figure 1, that carry also information about the context used in the step. The meaning of $u \xRightarrow{w}_R v$ is that a step can lead from u to v reading w (note that $\lfloor u \rfloor \supseteq w \subseteq \lfloor v \rfloor$). Idle tokens contribute to contexts. Transitions yield basic steps. When building larger steps, the maximum common context of the substeps is shared.

$$\frac{u \oplus x \overset{w}{\Longrightarrow}_{R_D} v \oplus y,}{(u,x) \overset{w}{\rightrightarrows}_D (v,y)} \qquad \frac{(u_1 \oplus w, x) \overset{w}{\rightrightarrows}_D (v_1 \oplus w, y),\ (u_2 \oplus w, y) \overset{w}{\rightrightarrows}_D (v_2 \oplus w, y')}{(u_1 \oplus u_2 \oplus w, x) \overset{w}{\rightrightarrows}_D (v_1 \oplus v_2 \oplus w, y')} \qquad \frac{(u,0) \overset{w}{\rightrightarrows}_D (v,0)}{u \overset{w}{\Rrightarrow}_D v}$$

Figure 2. The inference rules for $\_ \overset{-}{\Rrightarrow}_D \_$.

1.2 Zero-safe nets

According to the ordinary terminology, in a ‘0-safe’ net all places cannot contain any token in all reachable markings. We use the terminology *zero-safe net* —note the word ‘zero’ instead of the digit ‘0’— to mean that the net contains special places, called *zero places*, whose role is that of coordinating the atomic execution of transitions. However, no new interaction mechanism is needed, and the coordination of the transitions participating in a step is handled by the usual token-pushing rules of nets, assuming late delivery of stable tokens (postponed to the end of the transaction). Zero-safe places cannot contain any token in any *observable* state.

A *zero-safe net* (ZS net) is a tuple $B = (S, T, \mathsf{F}, u_{\text{in}}, Z)$ where $N_B = (S, T, \mathsf{F}, u_{\text{in}})$ is the *underlying* PT net of B and $Z \subseteq S$ is the set of *zero places*. The places in $L_B = S \setminus Z$ 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, whereas the presence of one or more zero tokens in a given marking makes it be *unobservable*. We call *stable tokens* and *zero tokens* the tokens that respectively belong to stable places and to zero places. Since $S^\oplus$ is a free commutative monoid, $S^\oplus \simeq L^\oplus \times Z^\oplus$ and we can write $t\colon (u,x) \to (v,y)$ for a transition t with $\text{pre}(t) = u \oplus x$ and $\text{post}(t) = v \oplus y$, where u and v are stable multisets and $x, y \in Z^\oplus$.

Zero-safe nets have been introduced in [2,3] and then extended in [6], by allowing the combined use of zero places and read arcs. A ZS *r-net* is a tuple $D = (S, T, \mathsf{F}, \mathsf{R}, Z, u_{\text{in}})$ such that $R_D = (S, T, \mathsf{F}, \mathsf{R}, u_{\text{in}})$ is a r-net and $(S, T, \mathsf{F}, Z, u_{\text{in}})$ is a ZS net. Though zero places can be used as context in [6], for simplicity we shall assume that $\text{ctx}(t) \subseteq S \setminus Z$ for all $t \in T$. In defining the dynamics of ZS r-nets, we can follow two main alternatives. The crucial point is whether to forbid or not that a stable token is read (possibly many times) and then also fetched during the same transaction. In the following we consider the semantics that forbids these kinds of consumptions, which is illustrated in Figure 2 (where $u, v, w \in L^\oplus$ and $x, y, y' \in Z^\oplus$). The second rule is crucial: it sequentializes on zero tokens, while composing in parallel on stable tokens (sharing the whole stable context w of the two substeps).

At the abstract level, the system modeled by D can be equivalently described via an r-net A_D such that $S_{A_D} = S_D \setminus Z_D$ and $(\_ \Rightarrow_{A_D} \_) = (\_ \Rrightarrow_D \_)$. Among the several r-nets that satisfy these conditions we can choose the optimal one, whose transitions represent the proofs of transaction steps $u \overset{w}{\Rrightarrow}_D v$ 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. (The abstract net can be defined according to either the collective or the individual token philosophy noticed in [17]: the two approaches yield the same step relation but different abstract nets.)

$$P ::= 0 \mid M \mid \eta.P \mid \mu?P : P \mid P+P \mid P|P \qquad M ::= \langle a \rangle$$

$$\eta ::= out(a) \mid rd(a) \mid in(a) \mid !in(a) \qquad \mu ::= rdp(a) \mid inp(a)$$

Figure 3. Linda-like process calculus.

1.3 Concurrent semantics for Linda

Thanks to their visual representation and their straightforward encoding of concurrency, nets have often been used as a computational model for concurrent and distributed languages such as Linda. The communication primitives of Linda are: (1) the output of a message $out(a)$; (2) the reading of a message (without consuming it) $rd(a)$; and (3) the fetching of a message $in(a)$, after which the message is no longer available in the tuple space. Two additional predicates $rdp(a)$ and $inp(a)$ allow, respectively, for (4) checking for the presence of a message without consuming it; and (5) atomically testing and consume the message if present. The two predicates can be used as the boolean component of conditional constructs.

In [10,9], first a process calculus is introduced, which embeds all the above primitives and then ri-nets are used to give a truly concurrent semantics to the calculus. In particular, the $rd(a)$ operation is modeled via read arcs (allowing for multiple concurrent readings of a), while predicates also require inhibitor arcs (to acknowledge the absence of messages, acting consequently). We recall that an *ri-net* is an r-net $(S, T, \mathsf{F}, \mathsf{R}, u_{\text{in}})$ together with a relation $\mathsf{I} \subseteq T \times S$, which expresses negative enabling conditions: if $\mathsf{I}(t,a)$, then t is never enabled at u with $u(a) > 0$.

It is worth noticing that the semantics of the $out(a)$ operation can give rise to two different interpretations: *ordered*, where the emission and the rendering of a message form a single and atomic action; or *unordered*, where emission and rendering are two autonomous actions. The difference is crucial, because in the second case, deadlock is decidable (it is possible to find an ordinary PT nets which is deadlock equivalent to the ri-net of the original encoding [10]). The syntax of the language is recalled in Figure 3, where the prefix $!in(a)$ means guarded replication and $\langle a \rangle$ represents a message in the tuple space. Due to space limitation we refer to [10] for the net encoding. Roughly speaking, the net modeling of Linda is defined by associating distinct places with messages $\langle a \rangle$ and sequential processes P, and suitable transitions with computational moves of sequential processes. A marking of the resulting net is a multiset of active processes and messages in the tuple space.

For example, for any process of the form $P = rdp(a)?P_1 : P_2$, the modeling net involves two transitions having the place P as preset, a read (resp. inhibitor) arc to $\langle a \rangle$, and $dec(P_1)$ (resp. $dec(P_2)$) as postset, with $dec(\_)$ the obvious decomposition function of processes into the multiset of their sequential components. Then, according to the ordered semantics, outputs are modeled via transitions with preset $out(a).P$ and postset $\langle a \rangle \oplus dec(P)$, whereas according to the unordered semantics, for each message a an additional place $\langle\langle a \rangle\rangle$ is considered that models emitted messages not yet available to other processes, and consequently, two transitions are considered: one from $out(a).P$ to $\langle\langle a \rangle\rangle \oplus dec(P)$ and the other from $\langle\langle a \rangle\rangle$ to $\langle a \rangle$.

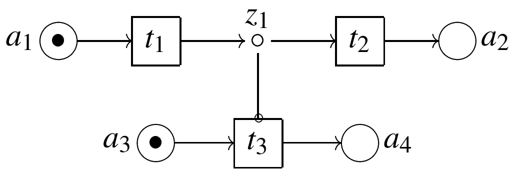

Figure 4. Is the abstract behavior of this net correct?

$$\frac{u \in S^{\oplus}}{\varnothing \vdash\!\circ\, u \stackrel{\lfloor u \rfloor}{\Longrightarrow}_C u} \qquad \frac{t : u \stackrel{w}{\longrightarrow} v \in T,}{t^{\mathsf{I}} \vdash\!\circ\, u \oplus w \stackrel{w}{\Longrightarrow}_C v \oplus w}$$

$$\frac{X_1 \vdash\!\circ\, u_1 \oplus w_1 \stackrel{w_1}{\Longrightarrow}_C v_1 \oplus w_1,\; X_2 \vdash\!\circ\, u_2 \oplus w_2 \stackrel{w_2}{\Longrightarrow}_C v_2 \oplus w_1}{\lfloor X_1 \oplus X_2 \rfloor \vdash\!\circ\, u_1 \oplus u_2 \oplus \lfloor w_1 \oplus w_2 \rfloor \stackrel{\lfloor w_1 \oplus w_2 \rfloor}{\Longrightarrow}_C v_1 \oplus v_2 \oplus \lfloor w_1 \oplus w_2 \rfloor}$$

Figure 5. The inference rules for $\_ \vdash\!\circ\, \_ \stackrel{\_}{\Longrightarrow}_C \_$.

2 Zero-safe nets with inhibitor arcs

Our first goal is to extend the zero-safe approach to nets with inhibitor arcs. This is not completely straightforward, as the following example demonstrates. Let us consider the net in Figure 4. At the abstract level, there are two apparently independent transactions: one from a_1 to a_2 (atomic execution of t_1 followed by t_2) and one from a_3 to a_4, but of course at the low level this activities can hardly be seen as independent because of the token flow in z_1, which disappears at the abstract level. To solve this ambiguity, we restrict the usage of inhibitor arcs to stable places only.

Definition 2.1 A *zero-safe ri-net* (ZS ri-net) is a tuple $E = (S, T, \mathsf{F}, \mathsf{R}, \mathsf{I}, u_{\mathrm{in}}, Z)$, where $D_E = (S, T, \mathsf{F}, \mathsf{R}, u_{\mathrm{in}}, Z)$ is a ZS r-net, and $\mathsf{I} \subseteq T \times (S \setminus Z)$ is the *inhibitor relation*. The underlying ri-net of E is $C_E(S, T, \mathsf{F}, \mathsf{R}, \mathsf{I}, u_{\mathrm{in}})$

We let $t^{\mathsf{I}} = \{a \mid \mathsf{I}(t, a)\}$, and assume that relations F, R, I are pairwise disjoint. The operational semantics of ordinary ri-nets C is recalled in Figure 5, where in the third rule we assume $(X_1 \oplus X_2) \cap (u_1 \oplus u_2 \oplus w_1 \oplus w_2 \oplus v_1 \oplus v_2) = \varnothing$. In fact, when composing two steps in parallel, we must check that each step does not involve tokens that inhibit the execution of the other step. A generic step $X \vdash\!\circ\, u \stackrel{w}{\Longrightarrow}_C v$ means that v can be reached from u reading w, and also that this step can be applied inside any other marking $u' \supseteq u$ provided that $u'(a) = 0$ for all $a \in X$. The operational semantics of ZS ri-nets is then the obvious extension illustrated in Figure 6 (where in the third rule we assume again $(X_1 \oplus X_2) \cap (u_1 \oplus u_2 \oplus w \oplus v_1 \oplus v_2) = \varnothing$), because we never concatenate on inhibitor places (they must be stable, and therefore the third rule just behaves like the parallel composition for them).

Analogously to [5], we can define the abstract ri-net A_E of E according to the two main philosophies of concurrency (collective or individual token), by characterizing the minimal concurrent steps $X \vdash\!\circ\, u \stackrel{v}{\Rrightarrow}_E v$ and viewing them as atomic activities with preset u, postset v, reading w and inhibited by X.

$$\frac{X \vdash\!\!\circ\, u \oplus x \xRightarrow{w}_{C_E} v \oplus y,}{X \vdash\!\!\circ\, (u,x) \overset{w}{\rightrightarrows}_E (v,y)} \qquad\qquad \frac{X \vdash\!\!\circ\, (u,0) \overset{w}{\rightrightarrows}_E (v,0)}{X \vdash\!\!\circ\, u \overset{w}{\Rrightarrow}_E v}$$

$$\frac{X_1 \vdash\!\!\circ\, (u_1 \oplus w, x) \overset{w}{\rightrightarrows}_E (v_1 \oplus w, y),\ X_2 \vdash\!\!\circ\, (u_2 \oplus w, y) \overset{w}{\rightrightarrows}_E (v_2 \oplus w, y')}{\lfloor X_1 \oplus X_2 \rfloor \vdash\!\!\circ\, (u_1 \oplus u_2 \oplus w, x) \overset{w}{\rightrightarrows}_E (v_1 \oplus v_2 \oplus w, y')}$$

Figure 6. The inference rules for $\_ \vdash\!\!\circ\, \_ \overset{-}{\Rrightarrow}_E \_$.

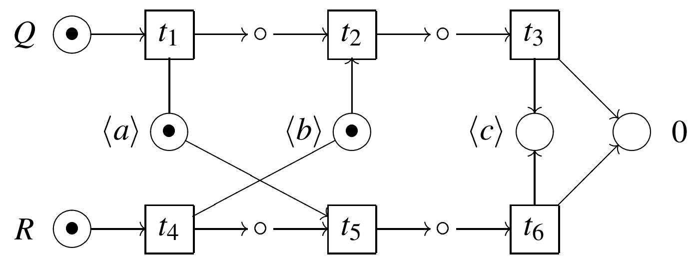

Figure 7. The ZS ri-net for $P = rd(a)\_in(b)\_out(c).0|rd(b)\_in(a)\_out(c).0|\langle a \rangle|\langle b \rangle$.

3 Transactions in Linda

Our second goal is to define a satisfactory treatment of transactions in Linda. The idea is to distinguishing between *low-level* (i.e., zero-safe) and *high-level* (i.e., stable) messages. Hence we assume two disjoint types, H and L, and a typing relation $a : \tau$ are given, such that the predicates μ of conditional expressions can only test for presence of observable messages $a : \mathtt{H}$. Moreover, we introduce the atomic prefixing η_P, where η can only be executed if P can commit.

The suitable semantic framework where to interpret the resulting language, that we call TraLinda, is then provided by ZS ri-nets. In fact we can straightforwardly adopt the translation in [10] to get a finite ri-net $C(P)$ for each agent P, and then take the subset of places associated to temporary messages as the set of zero-safe places. To handle atomic prefixing, we must also introduce a zero-safe place P_Z for each sequential agent P, which replaces P in the postsets of transitions associated with (atomic) prefixing. Since inhibitor arcs are only inserted because of conditional statements and the boolean predicates can only involve observable messages, we have that the resulting net $E(P)$ is a ZS ri-net, and as such it comes equipped with an abstract view of the system, which is the ri-net $A_{E(P)}$.

For example, in $P = out(a).0|in(a).0$, the two sequential subprocesses $Q = out(a).0$ and $R = in(a).0$ can communicate asynchronously if $a : \mathtt{H}$, but must communicate synchronously if $a : \mathtt{L}$. In fact, in $E(P)$ we have the initial marking $Q \oplus R$, with a transition $t_1 : Q \to \langle a \rangle \oplus 0$ for the first process, and a transition $t_2 : R \oplus \langle a \rangle \to 0$ for the second process, where $\langle a \rangle$ is zero-safe iff $a : \mathtt{L}$. Thus, if $a : \mathtt{L}$, a firing of t_1 opens a transaction that only the firing of t_2 can commit, resulting in the abstract transition from $Q \oplus R$ to $2 \cdot 0$ (two inactive processes/tokens), where the two operations are executed atomically (i.e., they are synchronized at the abstract level).

As another example, let $a,b,c : \mathtt{H}$ and take the process

$$P = rd(a)\_in(b)\_out(c).0|rd(b)\_in(a)\_out(c).0|\langle a\rangle|\langle b\rangle.$$

Then, only one of the two atomic threads can be executed, because the same stable token (e.g., $\langle a\rangle$ or $\langle b\rangle$) cannot be first read and then consumed during the same transaction. Thus, if $Q = rd(a)\_in(b)\_out(c).0$ is executed first, then the message $\langle b\rangle$ is fetched and cannot be read by $R = rd(b)\_in(a)\_out(c).0$, while if R executes, then $\langle a\rangle$ is consumed (see Figure 7). In fact, at the abstract level we have a transition from $Q \oplus \langle b\rangle$ to $\langle c\rangle \oplus 0$ and context $\langle a\rangle$ associated with the Q thread, and a transition from $R \oplus \langle a\rangle$ to $\langle c\rangle \oplus 0$ and context $\langle b\rangle$ corresponding to the R thread. The situation is very different if $P' = rd(a).in(b).out(c).0|rd(b).in(a).out(c).0|\langle a\rangle|\langle b\rangle$ is considered, where first the two read operations can be executed concurrently, then the two messages $\langle a\rangle$ and $\langle b\rangle$ can be retrieved, and finally two copies of the message $out(c)$ can be emitted.

Conclusion

We have shown how to handle inhibitor arcs inside the zero-safe approach, with application to the modeling of transactions in the original extension TraLinda of the language Linda. The results presented here are part of a broader research, which aims at integrating the zero-safe approach with distributed languages such as the Join calculus or the JavaSpaces middleware [18], to obtain a general purpose environment where distributed transactions can be faithfully designed, programmed and executed. In particular, though recent results suggest that several techniques inspired by Linda-like coordination languages are also adequate to deal with JavaSpaces, due to space limitation, we leave to the full version of this paper the comparison with the coordination primitives considered e.g. in [14,13], together with a precise relationship between the zero-safe approach and the ordered/unordered semantics of Linda.

Acknowledgement
We thank the anonymous referees for their kind comments and suggestions.

References

[1] A. Asperti and N. Busi. Mobile Petri nets. Tech. Rep. UBLCS 96-10, Computer Science Dept, Univ. of Bologna, 1996. `ftp://ftp.cs.unibo.it/pub/techreports/96-10.ps.gz`

[2] R. Bruni and U. Montanari. Zero-safe nets, or transition synchronization made simple. *Proc. EXPRESS'97*, *Elect. Notes in Th. Comput. Sci.* 7. Elsevier, 1997. `http://www.elsevier.nl/locate/entcs/volume7.html`

[3] R. Bruni and U. Montanari. Zero-safe nets: The individual token approach. *Proc. WADT'97*, *Lect. Notes in Comput. Sci.* 1376, pp. 122–140. Springer, 1998. `http://www.di.unipi.it/~bruni/publications/wadt97.ps.gz`

[4] R. Bruni and U. Montanari. Executing transactions in zero-safe nets. *Proc. ICATPN 2000*, *Lect. Notes in Comput. Sci.* 1825, pp. 83–102. Springer, 2000. http://link.springer.de/link/service/series/0558/tocs/t1825.htm

[5] R. Bruni and U. Montanari. Zero-safe nets: Comparing the collective and individual token approaches. *Inform. and Comput.*, 156:46–89, 2000. http://www.idealibrary.com/links/toc/inco/156/1/0

[6] R. Bruni and U. Montanari. Transactions and zero-safe nets. To appear in *Advances in Petri Nets: Unifying Petri Nets*. Springer, 2001. http://www.di.unipi.it/~bruni/publications/uapnzs.ps.gz

[7] R. Bruni and V. Sassone. Algebraic models for contextual nets. *Proc. ICALP 2000*, *Lect. Notes in Comput. Sci.* 1853, pp. 175–186. Springer, 2000. http://www.di.unipi.it/~bruni/publications/icalp2000.ps.gz

[8] M. Buscemi and V. Sassone. High-level Petri nets as type theories in the Join calculus. *Proc. FoSSaCS'01*, *Lect. Notes in Comput. Sci.* 2030, pp. 104–120. Springer, 2001. http://link.springer.de/link/service/series/0558/tocs/t2030.htm

[9] N. Busi. *Petri Nets with Inhibitor and Read Arcs: Semantics, Analysis and Application to Process Calculi*. PhD thesis, Mathematics Department, University of Siena, 1998. http://www.cs.unibo.it/~busi/www/thesis.ps.gz

[10] N. Busi, R. Gorrieri, and G. Zavattaro. A truly concurrent view of Linda interprocess communication. Tech. Rep. UBLCS 97-02, Computer Science Dept., Univ. of Bologna, 1996. ftp://ftp.cs.unibo.it/pub/techreports/97-02.ps.gz

[11] N. Busi, R. Gorrieri, and G. Zavattaro. Comparing three semantics for Linda-like languages. *Theoret. Comput. Sci.*, 240(1):49–90, 2000.

[12] N. Busi, R. Gorrieri, and G. Zavattaro. On the expressiveness of Linda coordination primitives. *Inform. and Comput.*, 156(1–2):90–121, 2000. http://www.idealibrary.com/links/toc/inco/156/1/0

[13] N. Busi and G. Zavattaro. Process calculi for coordination: from Linda to JavaSpaces. *Proc. AMAST'00*, *Lect. Notes in Comput. Sci.* 1816, pp. 198–212. Springer, 2000. http://link.springer.de/link/service/series/0558/tocs/t1816.htm

[14] N. Busi, R. Gorrieri, and G. Zavattaro. On the serializability of transactions in JavaSpaces. *Proc. ConCoord*, *Elect. Notes in Th. Comput. Sci.* 54. Elsevier, 2001.

[15] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. *Proc. CONCUR'96*, *Lect. Notes in Comput. Sci.* 1119, pp. 406–421. Springer, 1996.

[16] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[17] R.J. van Glabbeek and G.D. Plotkin. Configuration structures. *Proc. LICS'95*, pp. 199–209. IEEE Computer Society Press, 1995.

[18] Sun Microsystem, Inc. JavaSpaces$^{\mathrm{TM}}$ specifications, v.1.1, 2000. http://www.sun.com/jini/specs/js1_1.pdf