A Denotational Semantics for Timed Linda

[Extended Abstract]

F.S. de Boer
Universiteit Utrecht
Department of
Computer Science
Padualaan 14, De Uithof
3584 CH Utrecht, NL
frankb@cs.uu.nl

M. Gabbrielli
Università di Udine
Dipartimento di
Matematica e Informatica
Via delle Scienze 206
33100 Udine, Italy
gabbri@dimi.uniud.it

M.C. Meo
Università di L'Aquila
Dipartimento di Matematica
Pura ed Applicata
Via Vetoio, Loc. Coppito
67010 L'Aquila, Italy
meo@univaq.it

ABSTRACT

In [5] we introduced a Timed Linda language (T-Linda) which was obtained by a natural timed interpretation of the usual constructs of the Linda model and by including a simple primitive for specifying time-outs. Here we define a denotational model for T-Linda which is based on *timed reactive sequences*. The correctness of this model is proved w.r.t a notion of observables which include finite traces of actions and input/output pairs.

1. INTRODUCTION

Recently there has been some interest in the investigation of temporal properties of Linda-like coordination languages [12, 20, 5]. Time critical features are in fact important in the context of coordination of complex applications for open, distributed systems, where often one has the need to express timing constraints such as upper limits on the wait for an event, that is time-outs, and fixed (bounded) duration for granting a service, so called leasing. Consider for example the case of a bank teller machine. Once a card is accepted and its identification number has been checked, the machine asks the authorization of the bank to release the requested money. If the authorization does not arrive within a reasonable amount of time then the card should be given back to the customer. In order to model such a situation the language should allow us to specify that, in case a given time bound is exceeded (i.e. a time-out occurs), the wait is interrupted and an alternative action is taken. Indeed, time-outs can be expressed in JavaSpaces [30] and TSpaces [31], two coordination middlewares for distributed Java programming based on the Linda model [17, 14] and produced by Sun and IBM, respectively.

Temporal aspects of concurrent computations have been extensively studied in many different formal settings, including timed process algebras, temporal logic (and its executable versions) and the concurrent synchronous languages `ESTEREL`, `LUSTRE`, `SIGNAL` and Statecharts. In particular, timed process algebras (e.g. see [1, 2, 9, 19]) can be used to specify and verify large, non-deterministic reactive and real-time systems while deterministic concurrent synchronous languages such as `ESTEREL` [3] have specifically been designed for programming "kernels" of reactive systems.

A different approach to specify and program reactive systems has recently been defined [27, 28, 4] in the context of *concurrent constraint programming (ccp)* [25, 26, 29]. Ccp is a programming paradigm quite similar to Linda: In both cases the computation proceeds via accumulation of information in a global shared store, and information is produced by the concurrent and asynchronous activity of several processes. These can also check for the presence of information in the store, thus allowing one to express synchronization and coordination of different processes. However, while tuples are used in Linda, in ccp the information is represented in terms of constraints. Furthermore, differently from the case of Linda, in ccp once the information is produced it cannot be removed from the store, so the latter grows monotonically. In [27, 28, 4] timed extensions of ccp were defined around the hypothesis of *bounded asynchrony* [27]: Computation takes a bounded period of time (rather than being instantaneous as in `ESTEREL`) and the whole system evolves in cycles corresponding to time-units. While the language defined in [27, 28] is a deterministic one, inspired to synchronous languages and useful mainly for programming small real-time kernels, the timed ccp defined in [4] includes non-determinism and is more appropriate for specifying large systems involving several processes, possibly running on different processors, communicating via asynchronous links.

In this paper we investigate the semantics of a timed extension of Linda, called T-Linda. T-Linda was defined in [5] by introducing an explicit time-out primitive and by providing a natural timed interpretation of the usual programming constructs of Linda: A time-unit is identified with the time needed for the execution of a basic Linda action (out, in and rd) and action prefixing is interpreted as the next-time operator. The parallel operator of T-Linda is interpreted in terms of interleaving, as usual, however maximal parallelism is assumed for actions depending on time. In other words, time passes for all the parallel processes involved in a computation. This approach is different from that one of

PPDP 01 Florence, Italy

[4], where maximal parallelism was assumed for any kind of action, and it is also different from those considered in [12, 20] (see the discussion in Section 4).

We describe denotationally T-Linda by defining a fix-point semantics which is correct w.r.t. a notion of observables which considers sequences of actions performed on the tuple space. As a corollary, we obtain also correctness w.r.t the input/output notion of observables. We also conjecture that the semantics is fully abstract w.r.t. traces.

The semantics which associates to a process its observables (either traces or input/output pairs) is not compositional. The compositional semantics that we define is then based on different semantic structures, namely sequences of triples consisting of two multisets of tuples and a label. As we discuss later in the paper, these sequences are similar to the so called reactive sequences which were used in the semantics of several other languages. However reactive sequences are here provided with a different interpretation which accounts for the timing aspects: Intuitively, each triple $\langle m_i, m_i', x\rangle$ represents a computation step performed by the process P which, at time i, assuming m_i as input tuple-space (i.e. store) produces the space m_i' by performing an action of "type" x. This latter information is needed to distinguish actions representing passing of time, which may proceed according to maximal parallelism, from other actions which follow an interleaving scheduling policy.

The remaining of this paper is organized as follows: Next Section introduces the language T-Linda and its operational semantics. In Section 3 we introduce the denotational model. Section 4 concludes by discussing related work and by indicating directions for future research.

2. THE TIMED LINDA LANGUAGE

In this Section we introduce the *Timed Linda* (*T-Linda*) language and provide its operational semantics as defined in [5] by using a transition system. The material in this Section is therefore largely derived from [5].

The basic idea underlying Linda [17, 14] is that computation progresses via accumulation of information, represented in terms of tuples, in a global shared multiset called tuple space or store. Here we abstract from the specific nature of tuples and assume that these are elementary objects ranged over by $a, b, \ldots$. Information is produced and removed by the concurrent and asynchronous activity of several processes which can add a tuple a to the store by using the basic operation out(a). Dually, processes can also remove a tuple from the store by performing an in(a) operation and read a tuple from the store by means of a rd(a) operation. Differently from the case of out(a), both in(a) and rd(a) are blocking operations, that is, if a is not present in the store then the evaluation of in(a) and rd(a) is suspended, thus allowing one to express coordination among different processes. A kind of if_then_else is also present in the form of a construct rdp(a)?P_Q: If a is present in the store then the process P is evaluated, otherwise the computation proceeds with the process Q (an analogous construct inp(a)?P_Q differs from the previous one only in that the tuple a, whenever present, is removed from the store). The $\|$ operator allows one to express parallel composition of two processes A $\|$ B and it is usually described in terms of interleaving.

When querying the store for some information which is not present (yet) a process will either suspend until the required information has arrived (in case of in(a) and rd(a) processes) or will take an alternative continuation (in case of rdp(a)?P_Q and inp(a)?P_Q). However, as previously mentioned, in many practical cases often one has the need to express time-outs.

T-Linda was then defined [5] by assuming the existence of a *discrete global clock* and by assuming that the basic actions out(a), in(a) and rd(a) take one time-unit. T-Linda computations evolve in steps of one time-unit, so called clock-cycles, and action prefixing is the syntactic marker which distinguishes a time instant from the next one. So, for example, the process out(a).P has to be regarded as the process which updates the current store by adding a and then, at the *next* time instant, behaves like P. Analogously, if a is contained in the current store then the process in(a).P behaves like P at the next time instant, after having removed a from the store. If a is not present in the store at time t then the process in(a).P is suspended, i.e. at time $t+1$ it is checked again whether the store contains a. The process rd(a).P behaves like in(a).P without removing information from the store.

The parallel construct is interpreted in terms of interleaving, as usual in many (timed) process algebras and in all the main Linda dialects. Alternatively one could adopt maximal parallelism, which means that at each moment every enabled process of the system is activated. However this implies that parallel processes are executed on different processors and that, in principle, an unbounded number of processes is required, since dynamic process creation is allowed. Furthermore, differently from the case of the ccp paradigm (which does not allow information removal), competing requests for removing the same tuple should be scheduled: For example, if the store contains only one occurrence of the tuple a, then only one of the parallel requests in(a) $\|$ in(a) should be satisfied. For these reasons, differently from the case of the timed ccp language in [4], interleaving of accesses to the tuple space seems a more appropriate choice than maximal parallelism for T-Linda.

Time-outs are modeled in T-Linda by the construct

$$\mathsf{rdp(a)}_t\mathsf{?P\_Q}$$

whose meaning is analogous to that one of the un-timed version, with the difference that here one is allowed to wait t time units for the presence of the tuple a in the store and the subsequent evaluation of the process P; If this time limit is exceeded then the process Q is evaluated. Thus, in case the tuple a is not present in the tuple space the temporal behavior of $\mathsf{rdp(a)}_t\mathsf{?P\_Q}$ is in between the infinite wait of rd(a) and the no-wait of rdp(a)?P_Q. A similar construct $\mathsf{inp(a)}_t\mathsf{?P\_Q}$ which removes also the tuple a from the store could be defined analogously, we omit it since it semantic treatment would be similar to that one of $\mathsf{rdp(a)}_t\mathsf{?P\_Q}$.

Non-determinism arises in T-Linda by allowing a *choice* operator A + B, as in the case of classical timed process algebras. It is worth recalling that non-determinism arises naturally when considering large reactive systems running on different processors and communicating via asynchronous links. Furthermore, absence of non-determinism rules out the possibility of expressing interesting temporal behaviours like bounded response properties (see [22] and [32, 24]).

A notion of locality is also present and it is obtained by introducing the process A \ a which behaves like A, with a considered *local* to A.

Summarizing, we obtain the following syntax.

DEFINITION 2.1. *[Timed Linda Language] [5] Assuming a given set of tuples* T*, with typical elements* $a, b \ldots$*, the syntax of the* Timed Linda processes *is given by the following grammar:*

$$\mathsf{P} \quad ::= \quad \mathsf{stop} \mid \mathsf{out(a).P} \mid \mathsf{in(a).P} \mid \mathsf{rd(a).P} \mid \mathsf{rdp(a)_t?P\_P} \mid \mathsf{P} \parallel \mathsf{P} \mid \mathsf{P} + \mathsf{P} \mid \mathsf{P} \setminus \mathsf{a} \mid \mathsf{X} \mid \mathsf{recX.P}$$

where t *is a natural number greater or equal to 0.*

In the previous definition, as usual, we assume that only guarded forms of recursion are used, that is, a recursive call is always prefixed by an $\mathsf{in}, \mathsf{out}, \mathsf{rd}$ or rdp action.

2.1 Operational semantics

The operational model of *T-Linda* can be formally described by a labeled transition system $T = (Conf, Label, \longrightarrow)$ where we assert that each transition step takes exactly one time-unit. Configurations *Conf* are pairs consisting of a process and a multi-set of tuples (the tuple space, also called store). In the following $\uplus$ denotes the multisets union, while Label $= \{\tau, \sigma\}$ is the set of labels. We use labels to distinguish "real" computational steps performed by processes which have the control (label σ) from the transitions which model only the passing of time (label τ). So σ-actions are those performed by processes which modify the store (out, in), which perform a check on the store ($\mathsf{rd}, \mathsf{rdp_t}$) or that correspond to exceeding a time-out ($\mathsf{rdp_0}$). On the other hand τ-actions are those performed by time-out processes ($\mathsf{rdp_t}$) in case they have not the control.

The transition relation $\longrightarrow \subseteq Conf \times Labels \times Conf$ is the least relation satisfying the (axioms and the) rules R1-R14 in Table 1 and characterizes the (temporal) evolution of the system. So, $\langle \mathsf{P}, m\rangle \xrightarrow{x} \langle \mathsf{Q}, m'\rangle$ with $x \in \{\sigma, \tau\}$ means that if at time t we have the process P and the store m, then at time $t+1$ we have the process Q and the store m'. Let us now briefly discuss the rules in Table 1.

The process stop represents successful termination, so it cannot make any transition. Axiom **R1** shows that we are considering a non-blocking output operation: The process $\mathsf{out(a).P}$ adds a to the tuple space m and then it behaves as P at the next time instant. Note that the evaluation of an output action takes one time-unit and the updated space $m \uplus \{a\}$ will be visible only starting from the next time instant.

Both $\mathsf{in(a)}$ and $\mathsf{rd(a)}$ are blocking operations, as shown by the axioms **R2** and **R3**: In case the store does not contain a the computation is suspended, otherwise the computation proceeds (removing a only in case of the in operation). As it results from the transition rules, also the evaluation of the $\mathsf{in(a)}$ and $\mathsf{rd(a)}$ actions takes one time-unit.

The axioms **R4-R8** show that the time-out process $\mathsf{rdp(a)_t?P\_Q}$ behaves either as P or as Q depending on the presence of a in the tuple space in the next t time units: if $t > 0$ and a is present in the tuple space then P is evaluated (rule **R4**). If $t > 0$ and a is not present then the check for a is repeated at the next time instant and the value of the counter t is decreased (axiom **R5**); Note that in this case we use the label σ, since a check on the store has been performed. As shown by axiom **R6**, the counter can be decreased also by performing a τ-action: Intuitively this rule is used to model the situation in which, even though the evaluation of the time-out started already, another (parallel) process has the control. In this case, differently from the approach in [12], time continues to elapse (via τ-actions) also for the time-out process (see also the rules for the parallel operator). Axioms **R7** and **R8** show that if the time-out is exceeded, i.e. the counter t has reached the value of 0, then the process $\mathsf{rdp(a)_t?P\_Q}$ behaves as Q. The presence of the τ action in axiom **R8** ensure that the time-out process terminates at the correct time instant also when a σ action cannot be performed.

Rules **R9-R10** model the parallel composition operator in terms of *interleaving*, since only one basic σ-action is allowed for each transition (i.e. for each unit of time). This means that the access to the shared resource consisting of the global tuple space is granted to one process a time. However, time passes for all the processes appearing in the $\parallel$ context at the external level, as shown by rule **R9**, since τ-actions are allowed together with σ-actions. On the other hand, a parallel component is allowed to proceed in isolation if (and only if) the other parallel component cannot perform a τ-action (rule **R10**). To summarize, we adopt maximal parallelism for time elapsing (i.e. τ-actions) and an interleaving model for basic computation steps (i.e. σ-actions).

We have adopted this approach, different from that one in [12], because it seems more adequate to the nature of time-out operators not to interrupt the elapsing of time once the evaluation of a time-out has started (see Section 4 for a detailed comparison with [12]).

Clearly one could start the elapsing of time when the time out process is scheduled, rather than when it appears in the top-level current parallel context. This modification could easily be obtained by adding a syntactic construct to differentiate active time outs from inactive ones and by changing accordingly the transition system. One could also easily modify the semantics (both operational and denotational) to consider a more liberal assumption which allows multiple read actions in parallel. On the contrary, considering full maximal parallelism, i.e. allowing each enabled process to proceed, would cause several substantial differences.

Rules $\mathbf{R11} - \mathbf{R13}$ define the behavior of the choice operator. Here, differently from the usual case, when performing τ-actions we do not select a branch in the choice (rules **R12** and **R13**) because these actions, as previously mentioned, denote only the passing of time. Analogously to the case of the parallel operator, a process P in a choice $\mathsf{P} + \mathsf{Q}$ can locally advance its time (i.e. perform τ-actions) only if no other activated process in Q requires passing of time.

As specified by rule **R14**, the process $\mathsf{P} \setminus \mathsf{a}$ behaves like P, with a considered *local* to P, i.e. the information on a provided by the external tuple space is hidden from P and, conversely, the information on a produced locally by P is hidden from external world. To describe locality in rule **R13** the syntax has been extended by a process $\mathsf{P}^d \setminus \mathsf{a}$ where d is a local tuple space of P containing information on a which is hidden in the external store. When the computation starts the local store is empty, i.e. $\mathsf{P} \setminus \mathsf{a} = \mathsf{P}^{\emptyset} \setminus \mathsf{a}$. In this rule we use also the following notation: Given a multiset m, we denote by $m \downarrow a$ the multiset obtained from m by deleting all the occurrences of the tuple a and we denote by $m \uparrow a$ the multiset consisting only of the occurrences in m of the tuple a.

Rule **R15** treats the case of a recursive definition in the usual way (recall that guarded recursion is assumed, i.e. a recursive call is always prefixed by an $\mathsf{in}, \mathsf{out}, \mathsf{rd}$ or rdp action). Even though we use a rule with a negative premise,

R1 $\langle \mathsf{out(a).P}, m\rangle \xrightarrow{\sigma} \langle \mathsf{P}, m \uplus \{a\}\rangle$

R2 $\langle \mathsf{in(a).P}, m \uplus \{a\}\rangle \xrightarrow{\sigma} \langle \mathsf{P}, m\rangle$

R3 $\langle \mathsf{rd(a).P}, m \uplus \{a\}\rangle \xrightarrow{\sigma} \langle \mathsf{P}, m \uplus \{a\}\rangle$

R4 $\langle \mathsf{rdp(a)_t?P\_Q}, m \uplus \{a\}\rangle \xrightarrow{\sigma} \langle \mathsf{P}, m \uplus \{a\}\rangle \quad t > 0$

R5 $\langle \mathsf{rdp(a)_t?P\_Q}, m\rangle \xrightarrow{\sigma} \langle \mathsf{rdp(a)_{t-1}?P\_Q}, m\rangle \quad t > 0 \ \text{ and } \ a \notin m$

R6 $\langle \mathsf{rdp(a)_t?P\_Q}, m\rangle \xrightarrow{\tau} \langle \mathsf{rdp(a)_{t-1}?P\_Q}, m\rangle \quad t > 0$

R7 $\langle \mathsf{rdp(a)_0?P\_Q}, m\rangle \xrightarrow{\sigma} \langle \mathsf{Q}, m\rangle$

R8 $\langle \mathsf{rdp(a)_0?P\_Q}, m\rangle \xrightarrow{\tau} \langle \mathsf{Q}, m\rangle$

R9 $$\frac{\langle \mathsf{P}, m\rangle \xrightarrow{x} \langle \mathsf{P}', m'\rangle \quad \langle \mathsf{Q}, m\rangle \xrightarrow{\tau} \langle \mathsf{Q}', m\rangle \quad x \in \{\sigma, \tau\}}{\langle \mathsf{P} \parallel \mathsf{Q}, m\rangle \xrightarrow{x} \langle \mathsf{P}' \parallel \mathsf{Q}', m'\rangle}$$

R10 $$\frac{\langle \mathsf{P}, m\rangle \xrightarrow{x} \langle \mathsf{P}', m'\rangle \quad \langle \mathsf{Q}, m\rangle \not\xrightarrow{\tau} \quad x \in \{\sigma, \tau\}}{\langle \mathsf{P} \parallel \mathsf{Q}, m\rangle \xrightarrow{x} \langle \mathsf{P}' \parallel \mathsf{Q}, m'\rangle}$$

R11 $$\frac{\langle \mathsf{P}, m\rangle \xrightarrow{\sigma} \langle \mathsf{P}', m'\rangle}{\langle \mathsf{P} + \mathsf{Q}, m\rangle \xrightarrow{\sigma} \langle \mathsf{P}', m'\rangle}$$

R12 $$\frac{\langle \mathsf{P}, m\rangle \xrightarrow{\tau} \langle \mathsf{P}', m\rangle \quad \langle \mathsf{Q}, m\rangle \xrightarrow{\tau} \langle \mathsf{Q}', m\rangle}{\langle \mathsf{P} + \mathsf{Q}, m\rangle \xrightarrow{\tau} \langle \mathsf{P}' + \mathsf{Q}', m\rangle}$$

R13 $$\frac{\langle \mathsf{P}, m\rangle \xrightarrow{\tau} \langle \mathsf{P}', m\rangle \quad \langle \mathsf{Q}, m\rangle \not\xrightarrow{\tau}}{\langle \mathsf{P} + \mathsf{Q}, m\rangle \xrightarrow{\tau} \langle \mathsf{P}' + \mathsf{Q}, m\rangle}$$

R14 $$\frac{\langle \mathsf{P}, (m \downarrow a) \uplus d\rangle \xrightarrow{x} \langle \mathsf{Q}, m'\rangle \quad x \in \{\sigma, \tau\}}{\langle \mathsf{P^d} \setminus \mathsf{a}, m\rangle \xrightarrow{x} \langle \mathsf{Q^{m' \uparrow a}} \setminus \mathsf{a}, (m' \downarrow a) \uplus (m \uparrow a)\rangle}$$

R15 $$\frac{\langle \mathsf{P[recX.P/X]}, m\rangle \xrightarrow{x} \langle \mathsf{P}', m'\rangle \quad x \in \{\sigma, \tau\}}{\langle \mathsf{recX.P}, m\rangle \xrightarrow{x} \langle \mathsf{P}', m'\rangle}$$

Table 1: The transition system for *T-Linda* (symmetric rules of rules R9,R10,R11 and R13 are omitted).

the relation $\longrightarrow$ described by the rules in Table 1 is well defined since the transition system it is strictly stratifiable (see [18]).

Using such a transition system we can then define the following notion of observables which considers the behavior of terminating computations, including the deadlocked ones, in terms of traces. Other notions of observables (e.g. input/ouput pairs) can be obtained as simple abstractions of the traces we consider here. Note that we consider only sequences of transition steps which do not involve τ-steps, as these do not correspond to actions on the store (also timeouts can perform σ-actions, see rules **R4** and **R5**).

DEFINITION 2.2. *[Observables] Let* P *be a T-Linda process. We define*

$$\mathcal{O}(\mathsf{P}) = \{m_1 m_2 \dots m_n \mid \langle \mathsf{P}, m_1\rangle \xrightarrow{\sigma} \langle \mathsf{P}_2, m_2\rangle \xrightarrow{\sigma} \langle \mathsf{P}_n, m_n\rangle \xrightarrow{\sigma} \not\rightarrow\}.$$

Other notions of observables (e.g. input/ouput pairs or "resting points") can be obtained as simple abstractions of the previous notion. For example, input/ouput pairs are obviously defined as follows.

DEFINITION 2.3. *[Observables] Let* P *be a T-Linda process. We define*

$$\mathcal{O}_{io}(\mathsf{P}) = \{\langle m_1, m_n\rangle \mid \textit{ there exists } m_1 m_2 \dots m_n \in \mathcal{O}(\mathsf{P})\}.$$

3. THE DENOTATIONAL MODEL

It is easy to see that the operational semantics which associates to a process A its observables $\mathcal{O}_{io}(\mathsf{P})$ is not compositional. For example, consider the processes

$$\mathsf{P} = \mathsf{rdp(a)}_1\mathsf{?out(b)}\lrcorner\mathsf{loop}$$

and

$$\mathsf{Q} = \mathsf{rdp(a)}_2\mathsf{?out(b)}\lrcorner\mathsf{loop}$$

where loop is any non terminating process. Then $\mathcal{O}(\mathsf{P}) = \mathcal{O}(\mathsf{Q})$ holds, however we have that

$$\{c\}\{a,c\}\{a,c\}\{a,b,c\} \in \mathcal{O}(\mathsf{Q} \parallel \mathsf{out(a)}) \setminus \mathcal{O}(\mathsf{P} \parallel \mathsf{out(a)}).$$

As a consequence, also the semantics which associates to a process P the observables $\mathcal{O}_{io}(\mathsf{P})$ is not compositional.

In this Section we then define a compositional characterization of the operational semantics obtained by using *timed reactive sequences* to represent *Linda* computations. These sequences are similar to those used in the semantics of dataflow languages [21], imperative languages [6, 10], (timed) ccp [8, 4] and Linda [11]. However, differently from the previous cases, here we consider triples rather than pairs, as σ-actions have to be distinguished from τ-actions. This difference, which accounts for the temporal features of T-Linda, leads to a different technical development. Our denotational model associates to a process a set of (timed) reactive sequences of the form

$$\langle m_1, m_1', x_1\rangle \cdots \langle m_n, m_n', x_n\rangle\langle m, m, \sigma\rangle$$

where for any i, $1 \le i \le n$, m, m_i, m_i' are multisets and $x_i \in \{\sigma, \tau\}$. Any triple $\langle m_i, m_i', x_i\rangle$ represents a computation step performed by a generic process at time i: Intuitively, the process transforms the tuple space from m_i to m_i' by performing a transition step labeled by x_i or, in other words, m_i is the assumption on the tuple space, x_i is the label of the performed step and m_i' is the contribution of the process itself. The last triple indicates that no further information can be produced by the process, thus pointing out that a "resting point" has been reached.

Actually this intuitive interpretation of sequences is not completely adequate. Indeed, the basic idea underlying the denotational model then is that, differently from the case of the operational semantics, inactive processes can always make a τ-step, where an inactive process is either a suspended one (due to the absence of the required tuple in the store) or a non scheduled component of a parallel construct. These additional τ-steps, which represent passing of time and are needed to obtain a compositional model in a simple way, are then added to denotations as triples of the form $\langle m, m, \tau\rangle$. For example, the denotation of the process out(a) (we omit the continuation for simplicity) contains all the triples of the form $\langle m, m \uplus \{a\}, \sigma\rangle$ for any possible initial store m, as these represent the action of adding the tuple a to the current store. However, such a denotation contains also sequences where $\langle m, m \uplus \{a\}, \sigma\rangle$ is preceded by a finite sequence of triples of the form $\langle m, m, \tau\rangle$, where such a sequence represents the passing of time while the process is inactive (because some other parallel process is scheduled).

Before defining formally the denotational semantics we need to define the operators $\tilde{out}$, $\tilde{in}$, $\tilde{rd}$, $\tilde{rdp}$, $\tilde{\parallel}$, $\tilde{+}$ and $\tilde{\setminus}$ which act on sets of sequences and are the semantic counterparts of the syntactic constructs appearing in the language. Here and in the following the set of all reactive sequences is denoted by $\mathcal{S}$, with typical elements $s, s_1 \dots$, while sets of reactive sequences are denoted by $S, S_1 \dots$ and $\wp(\mathcal{S})$ denotes the set of subsets of $\mathcal{S}$. Furthermore, $\cdot$ indicates the operator which concatenates sequence and given a sequence $s = \langle m_1, m_1', x_1\rangle\langle m_2, m_2', x_2\rangle \cdots \langle m_n, m_n, \sigma\rangle$ we define $first(s) = m_1$ and $result(s) = m_n$. We also assume that functions (with range $\wp(\mathcal{S})$) are ordered by the pointwise extension of the ordering $\subseteq$.

DEFINITION 3.1. *Let S, S_1 and S_2 be sets of reactive sequences. Then we define the operators $\tilde{out}$, $\tilde{in}$, $\tilde{rd}$, $\tilde{rdp}$, $\tilde{\parallel}$, $\tilde{+}$ and $\tilde{\setminus}$ as follows:*

The Out-Operator $\tilde{out} : \mathsf{T} \times \wp(\mathcal{S}) \to \wp(\mathcal{S})$ *is the least function (w.r.t. the ordering induced by $\subseteq$) which satisfies the following equation*

$$\begin{aligned}\tilde{out}(a,S) = \quad & \{s \in \mathcal{S} \mid s = \langle m, m \uplus \{a\}, \sigma\rangle \cdot s' \textit{ and } s' \in S\ \} \\ & \cup \\ & \{s \in \mathcal{S} \mid s = \langle m, m, \tau\rangle \cdot s' \textit{ and } s' \in \tilde{out}(a,S)\ \}.\end{aligned}$$

The In-Operator $\tilde{in} : \mathsf{T} \times \wp(\mathcal{S}) \to \wp(\mathcal{S})$ *is the least function which satisfies the following equation*

$$\begin{aligned}\tilde{in}(a,S) = \quad & \{s \in \mathcal{S} \mid \textit{either } s = \langle m \uplus \{a\}, m, \sigma\rangle \cdot s' \textit{ and} \\ & \qquad s' \in S \textit{ or } s = \langle m, m, \sigma\rangle \textit{ and } a \notin m\ \} \\ & \cup \\ & \{s \in \mathcal{S} \mid s = \langle m, m, \tau\rangle \cdot s' \textit{ and } s' \in \tilde{in}(a,S)\ \}.\end{aligned}$$

The Rd-Operator $\tilde{rd} : \mathsf{T} \times \wp(\mathcal{S}) \to \wp(\mathcal{S})$ *is the least function which satisfies the following equation*

$$\begin{aligned}\tilde{rd}(a,S) = \quad & \{s \in \mathcal{S} \mid \textit{either } s = \langle m \uplus \{a\}, m \uplus \{a\}, \sigma\rangle \cdot s' \\ & \qquad \textit{and } s' \in S \textit{ or} \\ & \qquad s = \langle m, m, \sigma\rangle \textit{ and } a \notin m\ \} \\ & \cup \\ & \{s \in \mathcal{S} \mid s = \langle m, m, \tau\rangle \cdot s' \textit{ and } s' \in \tilde{rd}(a,S)\ \}.\end{aligned}$$

The $\mathbf{Rdp_t}$-Operator $\tilde{rdp}_t : \mathsf{T} \times \wp(\mathcal{S}) \times \wp(\mathcal{S}) \to \wp(\mathcal{S})$, *with*

$t > 0$, *is defined as* $r\tilde{d}p_t(a, S_1, S_2) =$

$$\begin{array}{ll}\{s \in \mathcal{S} \mid & \textit{either } s = \langle m \uplus \{a\}, m \uplus \{a\}, \sigma\rangle \cdot s' \textit{ and } s' \in S_1 \\ & \textit{or } s = \langle m, m, \sigma\rangle \cdot s',\ a \notin m \textit{ and} \\ & s' \in r\tilde{d}p_{t-1}(a, S_1, S_2)\ \} \\ \cup & \\ \{s \in \mathcal{S} \mid & s = \langle m, m, \tau\rangle \cdot s' \textit{ and } s' \in r\tilde{d}p_{t-1}(a, S_1, S_2)\ \}.\end{array}$$

The Rdp$_0$-Operator $r\tilde{d}p_0 : \mathsf{T} \times \wp(\mathcal{S}) \times \wp(\mathcal{S}) \to \wp(\mathcal{S})$ *is the least function which satisfies the following equation:* $r\tilde{d}p_0(a, S_1, S_2) =$

$$\begin{array}{ll}\{s \in \mathcal{S} \mid & \textit{either } s = \langle m, m, \sigma\rangle \cdot s' \textit{ and } s' \in S_2 \\ & \textit{or } s = \langle m, m, \tau\rangle \cdot s' \textit{ and } s' \in r\tilde{d}p_0(a, S_1, S_2)\ \}.\end{array}$$

Parallel Composition. *Let* $\tilde{\|} \in \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ *be the (commutative and associative) partial operator defined by induction on the length of the sequences as follows:*

$\langle m, m, \sigma\rangle \tilde{\|} \langle m, m, \sigma\rangle = \langle m, m, \sigma\rangle$,
$\langle m, m', x\rangle \cdot s \tilde{\|} \langle m, m, \tau\rangle \cdot s' = \langle m, m, \tau\rangle \cdot s' \tilde{\|} \langle m, m', x\rangle \cdot s = \langle m, m', x\rangle \cdot (s \tilde{\|} s')$,

where $x \in \{\sigma, \tau\}$.
We define $S_1 \tilde{\|} S_2$ *as the point-wise extension of the above operator to sets.*

Choice. *Let* S *be a set of reactive sequences and let* m, m' *be multisets. We define*

$$\Sigma(S) = \{s \mid \quad \textit{there exists } m, m', s' \textit{ such that } s = \langle m, m', \sigma\rangle \cdot s' \textit{ and } s \in S\}$$

so, $\Sigma(S)$ *denotes the subset of* S *consisting of sequences starting with a* σ*-action. We also define*

$$\Gamma(S, m) = \{s \mid \langle m, m, \tau\rangle \cdot s \in S\};$$

$\Gamma(S, m)$ *denotes then the set of sequences obtained by removing the first element from those sequences in* S *which start with a* τ*-action and input store* m.

The semantic choice operator $\tilde{+} : \wp(\mathcal{S}) \times \wp(\mathcal{S}) \to \wp(\mathcal{S})$ *is then the least function which satisfies the following equation:*

$$\begin{array}{lll}S_1 \tilde{+} S_2 = & \{s \in \mathcal{S} \mid & s \in \Sigma(S_1) \cup \Sigma(S_2)\} \\ & \cup & \\ & \{s \in \mathcal{S} \mid & s = \langle m, m, \tau\rangle \cdot s', \\ & & s' \in \Gamma(S_1, m) \tilde{+} \Gamma(S_2, m)\} \\ & \cup & \\ & \{\langle m, m, \sigma\rangle \in \mathcal{S} \mid \langle m, m, \sigma\rangle \in S_1 \cap S_2\ \}.\end{array}$$

Hiding Operator. *We first need the notions of* a*-connected and* a*-invariant sequence which are similar to those introduced in [7]. First let us recall that, as previously defined, given a multiset* m, *we denote by* $m \downarrow a$ *the multiset obtained from* m *by deleting all the occurrences of the tuple* a *and we denote by* $m \uparrow a$ *the multiset consisting only of the occurrences in* m *of the tuple* a.

Given a sequence $s = \langle m_1, m_1', x_1\rangle \cdots \langle m_n, m_n, \sigma\rangle$, *we denote by* $s \downarrow a$ *the sequence obtained from* s *by deleting each occurrence of the tuple* a *in the multisets* $m_1, m_1', \ldots, m_n$. *Namely*

$$s \downarrow a = \langle m_1 \downarrow a, m_1' \downarrow a, x_1\rangle \cdots \langle m_n \downarrow a, m_n \downarrow a, \sigma\rangle.$$

Then we say that s *is* a*-connected if*

- $m_1 \downarrow a = m_1$ *(that is, the input multiset of* s' *does not contain any occurrence of the tuple* a*) and*
- $m_i \uparrow a = m_{i-1}' \uparrow a$ *for each* $i \in [2, n]$ *(that is, each input multiset* m_i *does not contain any information on* a *which has not been produced previously in the sequence by some* m_j'*; note that a generic* m' *can introduce new* a*'s).*

Moreover, the sequence s *is* a*-invariant if*

- *for all computation steps* $\langle m, m', x\rangle$ *of* s,

$$m' \uparrow a = m \uparrow a$$

holds.

The semantic hiding operator then can be defined as follows:

$$\tilde{\setminus}(a, S) = \{s \in \mathcal{S} \mid \quad \textit{there exists } s' \in S \textit{ s.t. } s \downarrow a = s' \downarrow a, \ s' \textit{ is } a\textit{-connected and } s \textit{ is } a\textit{-invariant}\}.$$

It is immediate to see that the previous semantic operators are well defined, that is, the least functions which satisfies the equations actually exist and can be obtained by a standard fix-point construction.

The definitions of the $o\tilde{u}t, i\tilde{n}, r\tilde{d}$ and $r\tilde{d}p_t$ operators reflect the operational behaviour of their syntactic counterpart with the mentioned addition of τ-steps. In the definition of $i\tilde{n}$, $r\tilde{d}$ and $r\tilde{d}p_t$ we also include (as postfixes) triples of the form $\langle m, m, \sigma\rangle$ which denote suspension. More generally, as previously mentioned, these indicate that no further information can be generated, i.e. that a final result of computation has been reached (notice that each process is terminated by Stop, whose denotation contains sequences ending with a $\langle m, m, \sigma\rangle$ triple).

In the semantic parallel operator (acting on sequences) we require that at each point of time at most one σ-action is present and the two arguments of the operator agree with respect to the contribution of the environment (the first component of the triple). We also require that the two arguments have the same length (in all other cases the parallel composition is assumed being undefined): this is necessary to reflect the passage of time since the $i-th$ element of any sequence corresponds to the given process' action on the $i-th$ time step. Even though we merge point-wise sequences of the same length, this models an interleaving approach for σ-actions, because of the previously mentioned addition of τ-steps to denotations.

Concerning the semantic choice operator, we include in the result all the sequences which start with a σ-action and belong to one of the two arguments: Clearly this models the case in which one of the two component is chosen and a σ-action is performed (see rule **R11**). In case both the arguments S_1 and S_2 contain a sequence starting with $\langle m, m, \tau\rangle$, say $\langle m, m, \tau\rangle \cdot s_1$ and $\langle m, m, \tau\rangle \cdot s_2$, we also include in $S_1 \tilde{+} S_2$ a sequence starting with $\langle m, m, \tau\rangle$ and continuing with a sequence resulting from the (semantic) choice between s_1 and s_2: This covers both cases considered by rules **R12** and **R13**, because of the addition of τ-steps to denotations.

In the hiding operator we say that a sequence is a-connected if no information on a is present in the input store which has not been already accumulated by the computation of the process itself. A sequence is a-invariant if its computation steps do not provide more information on a. These definitions are essentially the same as those used to model the notion of locality in [7].

We can then define the denotational semantics $\mathcal{D}$ as follows. Here Processes denote the set of T-Linda processes.

E1	$\mathcal{D}[\![\mathsf{stop}]\!] = \{\langle m_1, m_1, \tau\rangle\langle m_2, m_2, \tau\rangle \cdots \langle m_n, m_n, \sigma\rangle \in \mathcal{S} \mid n \geq 1\}$
E2	$\mathcal{D}[\![\mathsf{out(a).P}]\!] = \tilde{out}(a, \mathcal{D}[\![\mathsf{P}]\!])$.
E3	$\mathcal{D}[\![\mathsf{in(a).P}]\!] = \tilde{in}(a, \mathcal{D}[\![\mathsf{P}]\!])$.
E4	$\mathcal{D}[\![\mathsf{rd(a).P}]\!] = \tilde{rd}(a, \mathcal{D}[\![\mathsf{P}]\!])$.
E5	$\mathcal{D}[\![\mathsf{rdp(a)_t?P\_Q}]\!] = \tilde{rdp}_t(a, \mathcal{D}[\![\mathsf{P}]\!], \mathcal{D}[\![\mathsf{Q}]\!])$.
E6	$\mathcal{D}[\![\mathsf{A} \parallel \mathsf{B}]\!] = \mathcal{D}[\![\mathsf{A}]\!] \tilde{\parallel} \mathcal{D}[\![\mathsf{B}]\!]$
E7	$\mathcal{D}[\![\mathsf{P}_1 + \mathsf{P}_2]\!] = \mathcal{D}[\![\mathsf{P}_1]\!] \tilde{+} \mathcal{D}[\![\mathsf{P}_2]\!]$.
E8	$\mathcal{D}[\![\mathsf{P} \setminus \mathsf{a}]\!] = \tilde{\setminus}(a, \mathcal{D}[\![\mathsf{P}]\!])$.
E9	$\mathcal{D}[\![\mathsf{recX.P}]\!] = \mathcal{D}[\![\mathsf{P[recX.P/X]}]\!]$

Table 2: Denotational semantics of *T-Linda*

DEFINITION 3.2. *The denotational semantics $\mathcal{D} : Processes \to \wp(\mathcal{S})$ is the least function, w.r.t. the ordering induced by $\subseteq$, which satisfies the equations in Table 2.*

Also $\mathcal{D}$ is well defined and can be obtained by a fix-point construction. To see this, let us define an interpretation as a mapping $I : Processes \to \wp(\mathcal{S})$. Then let us denote by $\mathcal{I}$ the cpo of all the interpretations (with the ordering induced by $\subseteq$). We can then associate to the equations in Table 2 a monotonic (and continuous) mapping $\mathcal{F} : \mathcal{I} \to \mathcal{I}$ defined by the equations of Table 2, provided that we replace the symbol $\mathcal{D}$ for $\mathcal{F}(I)$ and that we replace equation **E9** for the following one: $\mathcal{F}(I)(\mathsf{recX.P}) = I(\mathsf{P[recX.P/X]})$.

Then, one can easily prove that a function satisfies the equations in Table 2 iff it is a fix-point of the function $\mathcal{F}$. Because this function is continuous (on a cpo) well known results ensure us that its least fix-point exists and it equals $\mathcal{F}^\omega$, where the powers are defined as follows: $\mathcal{F}^0 = I_0$ (this is the least interpretation which maps any process to the empty set); $\mathcal{F}^n = \mathcal{F}(\mathcal{F}^n - 1)$ and $\mathcal{F}^\omega = lub\{\mathcal{F} \uparrow n | n \geq 0\}$ (where lub is the least upper bound on the cpo $\mathcal{I}$).

As for the correctness of the denotational semantics, we notice that some reactive sequences do not correspond to real computations: Clearly, when considering a real computation no further contribution from the environment is possible. This means that, at each step, the assumption on the current store must be equal to the store produced by the previous step. In other words, for any two consecutive steps $\langle m_i, m_i', x_1\rangle\langle m_{i+1}, m_{i+1}', x_2\rangle$ we must have $m_i' = m_{i+1}$. Furthermore, triples containing τ-actions do not correspond to observable computational steps, as these involve σ-actions only. So we are led to the following.

DEFINITION 3.3. *Let $s = \langle m_1, m_1', x_1\rangle\langle m_2, m_2', x_2\rangle \cdots \langle m_{n-1}, m_{n-1}', x_{n-1}\rangle\langle m_n, m_n, \sigma\rangle$ be a reactive sequence. We say that s is connected if $m_i = m_{i-1}'$ for each i, $2 \leq i \leq n$, and $x_j \neq \tau$ for each j, $1 \leq j \leq n-1$.*

According to the previous definition, a sequence is connected if all the information assumed on the tuple space is produced by the process itself, apart from the initial input, and only σ-actions are involved. A connected sequence represents a *T-Linda* computation where the first (assumed) store is the input, while the last (produced) store is the result. This is the content of the following result.

THEOREM 3.4. *For any process P we have $\mathcal{O}(\mathsf{P}) =$*

$$\{m_1 m_2 \ldots m_n \mid \begin{array}{l}\textit{there exists a connected sequence}\\ s \in \mathcal{D}[\![\mathsf{P}]\!] \textit{ such that } s =\\ \langle m_1, m_1', x_1\rangle\langle m_2, m_2', x_2\rangle \cdots \langle m_n, m_n, \sigma\rangle\}.\end{array}$$

The proof of this result uses a modified transition system, where inactive (either suspended or not scheduled) processes can perform τ-actions. When considering our notions of observables, we can prove that such a modified transition system is equivalent to the previous one and agrees with the denotational model.

The following corollary, which shows the correctness w.r.t. the input/ouput, is immediate.

COROLLARY 3.5. *For any process P we have*

$$\mathcal{O}_{io}(\mathsf{P}) = \{\langle m, m'\rangle \mid \begin{array}{l}\textit{there exists a connected sequence}\\ s \in \mathcal{D}[\![\mathsf{P}]\!], \textit{ such that}\\ first(s) = m \textit{ and } result(s) = m'\}.\end{array}$$

From the definition of $\mathcal{D}[\![P]\!]$ (in particular from the definition of the semantic parallel operator) it follows that if a sequence s cannot be composed via $\tilde{\parallel}$ to obtain a connected sequence then such a sequence is useless and it can be eliminated. We then introduce the following abstraction on the semantics $\mathcal{D}[\![P]\!]$.

DEFINITION 3.6. *We say that a sequence s is semi-connected if there exists s' such that $s \tilde{\parallel} s'$ is defined and is a connected sequence. We then define*

$$\alpha(\mathcal{D}[\![P]\!]) = \{s \mid s \in \mathcal{D}[\![P]\!] \textit{ and } s \textit{ is semi-connected }\}.$$

It is straightforward to check that also $\alpha(\mathcal{D}[\![P]\!])$ is correct and compositional, as it satisfies the equations in Table 2.

So we could just define directly $\alpha(\mathcal{D}[\![P]\!])$ from scratch, by assuming that $\mathcal{S}$ contains only semi-connected sequences (we used a different definition for the sake of simplicity).

We conjecture that the denotational semantics $\alpha(\mathcal{D}[\![P]\!])$ is fully abstract w.r.t. the observables $\mathcal{O}$, that is that $\mathcal{O}(\mathsf{P}) = \mathcal{O}(\mathsf{Q})$ iff $\alpha(\mathcal{D}[\![P]\!]) = \alpha(\mathcal{D}[\![Q]\!])$ holds. We can prove this full abstraction result for a modified language which allows the simultaneous addition, deletion and check for presence of several tuples. When considering a language that allows to add/remove only a tuple a time, as we do, the proof of the full abstraction result becomes more involved and we are currently working out the details.

4. RELATED AND FUTURE WORK

Even though the original proposal of the Linda coordination languages dates back to the eighties [17, 14], the definition of specific process calculi to reason formally about these languages is quite recent [13, 16]. In particular, concerning temporal aspects, the only other formal treatments we are aware of are in [12, 20]. In [12] the authors investigate the semantics of several coordination primitives (including time-outs) which appeared in some recent extensions of the Linda model, namely JavaSpaces [30] and TSpaces [31]. Time is also considered in [20] where several variants of a language inspired to the ESTEREL model are introduced and compared.

The main differences between our approach and those in [12, 20] is that we investigate a timed Linda language mainly from the denotational semantics perspective while in [12, 20] the authors focus on the operational semantics. Furthermore, our interpretation of time and of the time-out construct is different from those appearing in the mentioned papers. We assume that once the elapsing of time for a time-out has started it cannot be interrupted, while this is not the case for the time-out considered in [12]. More precisely, similarly to what happens in some timed process algebras (e.g. [9]), actions in [12] are partitioned into instantaneous and time consuming ones and a time consuming action, such as the time-out, can be interleaved with an arbitrary number of out and in actions, which are considered instantaneous. On the other hand, we do not consider the out and in actions instantaneous, since these are the basic computational steps on the store, and if these action are interleaved with a time-out then elapsing of time is not interrupted. Also the approach followed in [20] partitions the basic actions into instantaneous and time consuming ones. Following the ESTEREL model, computation in [20] proceeds in "bursts of activity", that is, it uses a two phases schema: In the first phase elementary actions are executed instantaneously. Then, when no process can be further reduced, time progresses by one unit. This approach is similar to that one of the tcc language defined in [27] and it is substantially different from ours.

Related to the present paper is also [4], where we investigated a timed ccp language. Clearly, there are relevant differences with [4], since the ccp paradigm does not allow removal of information from the global store and this simplifies considerably the semantic issues. Furthermore, maximal parallelism was assumed in [4] while here we consider an interleaving model. This is a relevant difference, which causes also a different definition of the time-out construct: In fact, under the maximal parallelism hypothesis one could define inductively the time-out construct in terms of the usual Linda operators as follows: $\mathsf{rdp(a)}_1\mathsf{?P\_Q} = \mathsf{rdp(a)?P\_Q}$ and $\mathsf{rdp(a)}_t\mathsf{?P\_Q} = \mathsf{rdp(a)?P\_(rdp(a)}_{t-1}\mathsf{?P\_Q)}$ for $t > 1$. On the other hand, using an interleaving model and assuming, as we do, that once the evaluation of the time-out has started the elapsing of time can not be interrupted, the previous definition does not work, as one can easily check by considering a process consisting of two parallel time-outs[1].

We already mentioned some other papers which use reactive sequences for defining the semantics of several languages [21, 10, 8, 4, 11]. The main difference with them is in the specific model of computation that we use, where interleaving and maximal parallelism coexist, which leads us to use σ and τ annotations and to a different technical development.

In the definition of T-Linda we made the simplifying assumption that basic actions take one time unit. We can easily relax this assumption by allowing actions of different (discrete) duration without major changes in the semantic treatment. Also, our denotational semantics could easily be adapted to the other timed Linda languages mentioned before.

Apart from a further investigation of the full abstraction issue, ongoing and future work includes an investigation of the relations between Linda and ccp, especially considering temporal issues, and an extension of our approach in order to take into account also preemption mechanisms. The *preemption* of A is the action of aborting an active process A while starting another process B. There are many practical cases where a preemption is necessary: For example, if A is the process controlling the normal activity of some physical device and some abnormal situation arises, then A must be preempted and the exception handler B should be started. Preemption primitives are present explicitly, for example, in ESTEREL like languages [3]. Under the assumption of maximal parallelism a weak form of preemption can be naturally obtained in terms of the time-out construct (see [4]), while the presence of interleaving makes the definition of a preemption construct more complicated. Finally, a last topic which deserves further investigation is the study of temporal aspects in the context of languages for mobile computing based on the Linda paradigm, such as Klaim [15], since the implementations of these languages in some cases already include a notion of time-out.

5. REFERENCES

[1] L. Aceto and D. Murphy. Timing and causality in process algebra. *Acta Informatica*, 33(4): 317-350, 1996.

[2] J. Baeten and J. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2): 142-188, 1991.

[3] G. Berry and G. Gonthier. The ESTEREL programming language: Design, semantics and implementation. *Science of Computer Programming*, 19(2):87-152, 1992.

[4] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Timed Concurrent Constraint Language. *Information and Computation*, 161, 2000.

[5] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Timed Linda Language. *Proc. of 4-th International Conference*

[1] The point here is that one would need a rule of the form $\langle \mathsf{rdp(a)?P\_Q}, m\rangle \xrightarrow{\tau} \langle \mathsf{Q}, m\rangle$ with $a \notin m$, however this contradicts our interpretation of τ-actions, since in this case a check on the store (to test whether $a \notin m$) is performed.

on Coordination Languages and Models, vol. 1906 of LNCS. Springer-Verlag, 2000.

[6] F.S. de Boer, J.N. Kok, C. Palamidessi, and J.J.M.M. Rutten. The failure of failures in a paradigm for asynchronous communication. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings of CONCUR'91*, volume 527 of *Lecture Notes in Computer Science*, pages 111–126. Springer-Verlag, 1991.

[7] F.S. de Boer, J.N. Kok, C. Palamidessi, and J.J.M.M. Rutten. On Blocks: locality and asynchronous communication. In *Proc. of REX workshop on Semantics - Foundations and Applications*, volume 666 of LNCS, pages 73–90. Springer-Verlag, 1992.

[8] F.S. de Boer and C. Palamidessi. A Fully Abstract Model for Concurrent Constraint Programming. In S. Abramsky and T.S.E. Maibaum, editors, *Proc. of TAPSOFT/CAAP*, volume 493 of LNCS, pages 296–319. Springer-Verlag, 1991.

[9] P. Bremond-Gregoire and I. Lee. A Process Algebra of Communicating Shared Resources with Dense Time and Priorities. *Theoretical Computer Science* 189, 1997.

[10] S. Brookes. A fully abstract semantics of a shared variable parallel language. In *Proc. Eighth IEEE Symposium on Logic In Computer Science*. IEEE Computer Society Press, 1993.

[11] A. Brogi and J.-M. Jacquet. Modeling Coordination via asynchronous communication. In *Proc. of 1st International Conference on Coordination Languages and Models*, LNCS. Springer-Verlag, 1997.

[12] N. Busi, R. Gorrieri, and G. Zavattaro. Process Calculi for Coordination: from Linda to JavaSpaces. In *Proc. of 8-th International Conference on Algebraic Metodology and Software Technology*, LNCS. Springer-Verlag, 2000.

[13] N. Busi, R. Gorrieri and G. Zavattaro. A Process Algebraic View of Linda Coordination Primitives. *Theoretical Computer Science*, 192(2):167-199, 1998.

[14] N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4): 444-458, 1989.

[15] R. De Nicola, G. Ferrari and R. Pugliese. KLAIM: A Kernel Lamguage for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5): 315-330, 1998.

[16] R. De Nicola and R. Pugliese. Linda based Applicative and Imperative Process Algebras. To appear in *Theoretical Computer Science*, 2000.

[17] D. Gelernter. Generative communication in Linda. *ACM Transcations on Programming Languages and Systems*, 70(1): 80-112, 1985.

[18] J.F. Groote. Transition system specifications with negative premises. *Theoretical Computer Scince*, 118: 263-299, 1993.

[19] M. Hennessy and T. Regan. A temporal process algebra. *Information and Computation*, 117: 221-239, 1995.

[20] J.-M. Jacquet, K. De Bosschere and A. Brogi. On Timed Coordination Languages. In *Proc. of 4-th International Conference on Coordination Languages and Models*, vol. 1906 of LNCS. Springer-Verlag, 2000.

[21] B. Jonsson. A model and a proof system for asynchronous processes. In *Proc. of the 4th ACM Symp. on Principles of Distributed Computing*, pages 49–58. ACM Press, 1985.

[22] R. Koymans. Specifying real-time properties with metric temporal logix. *Real-Time systems*, 2(4):255-299, 1990.

[23] F. Moller and C. Tofts. A Temporal Calculus of Communicating Systems. In *Proc. CONCUR 90*, volume 459 of LNCS, pages 401-414. Springer-Verlag, 1990.

[24] C. Palamidessi and F.D. Valencia. A Concurrent Constraint Calculus for timed systems. *Draft*, 2000.

[25] V.A. Saraswat and M. Rinard. Concurrent constraint programming. In *Proc. of Seventeenth ACM Symposium on Principles of Programming Languages*, pages 232–245. ACM Press, 1990.

[26] V.A. Saraswat, M. Rinard, and P. Panangaden. Semantic Foundation of Concurrent Constraint Programming. In *Proc. Eighteenth ACM Symposium on Principles of Programming Languages*, pages 333-353. ACM Press, 1991.

[27] V.A. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of Timed Concurrent Constraint Programming. In S. Abramsky editor, *Proc. of the Ninth IEEE Symposium on Logic in Computer Science*, pages 71–80. IEEE Computer Press, July 1994.

[28] V.A. Saraswat, R. Jagadeesan, and V. Gupta Timed Default Concurrent Constraint Programming. *Journal of Symbolic Computation*, 22(5-6):475–520, 1996.

[29] G. Smolka. The Definition of Kernel Oz. In A. Podelski editor, *Constraints: Basics and Trends*, LNCS 910, pages 251–292. Springer-Verlag, 1995.

[30] Sun Microsystem, Inc. JavaSpaces Specifications, 1998.

[31] P. Wyckoff, S.W. McLaughry, T.J. Lehman, and D.A. Ford. TSpaces. *IBM Systems Journal*, 37(3), 1998.

[32] F.D. Valencia. Reactive Constraint Programming. Brics Progress Report, June 2000.