Timed Mobility in Process Algebra and Petri Nets

Gabriel Ciobanu and Maciej Koutny

TECHNICAL REPORT SERIES

No. CS-TR-1292 November 2011
Abstract

We present a process algebra called TiMo in which timeouts of interactions and adaptable migrations in a distributed environment with explicit locations can be specified. Timing constraints allow to control the communication between co-located mobile processes, and a migration action with variable destination supports flexible movement from one location to another. The model of time is based on local clocks rather than a global clock. We provide a structural translation of TiMo into behaviourally equivalent high level timed Petri nets. As a result, we obtain a formal net semantics for timed interaction and migration which is both structural and allows one to deal directly with concurrency and causality.
Bibliographical details

CIOBANU, G., KOUTNY, M.
Timed Mobility in Process Algebra and Petri Nets
[By] G. Ciobanu, M. Koutny
(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1292)

Added entries

NEWCASTLE UNIVERSITY

Abstract

We present a process algebra called TiMo in which timeouts of interactions and adaptable migrations in a distributed environment with explicit locations can be specified. Timing constraints allow to control the communication between co-located mobile processes, and a migration action with variable destination supports flexible movement from one location to another. The model of time is based on local clocks rather than a global clock. We provide a structural translation of TiMo into behaviourally equivalent high level timed Petri nets. As a result, we obtain a formal net semantics for timed interaction and migration which is both structural and allows one to deal directly with concurrency and causality.

About the authors

Gabriel Ciobanu is a senior researcher at the Institute of Computer Science, Romanian Academy. He is also a Professor at "Alexandru Ioan Cuza" University of Iasi, Romania.

Maciej Koutny obtained his MSc (1982) and PhD (1984) from the Warsaw University of Technology. In 1985 he joined the then Computing Laboratory of the University of Newcastle upon Tyne to work as a Research Associate. In 1986 he became a Lecturer in Computing Science at Newcastle, and in 1994 was promoted to an established Readership at Newcastle. In 2000 he became a Professor of Computing Science.

Suggested keywords

MOBILITY
LOCAL CLOCKS
PROCESS ALGEBRA
HIGH LEVEL PETRI NETS
SYNTAX DRIVEN TRANSLATION
BEHAVIOURAL EQUIVALENCE
Timed Mobility
in
Process Algebra and Petri Nets

Gabriel Ciobanu and Maciej Koutny

1 Institute of Computer Science, Romanian Academy
   and A.I.Cuza University of Iasi
   700483 Iasi, Romania
   gabriel@iit.tuiasi.ro

2 School of Computing Science
   Newcastle University
   Newcastle upon Tyne, NE1 7RU, United Kingdom
   maciej.koutny@newcastle.ac.uk

Abstract. We present a process algebra called T\textsuperscript{M0} in which timeouts of interactions and adaptable migrations in a distributed environment with explicit locations can be specified. Timing constraints allow to control the communication between co-located mobile processes, and a migration action with variable destination supports flexible movement from one location to another. The model of time is based on local clocks rather than a global clock.

We provide a structural translation of T\textsuperscript{M0} into behaviourally equivalent high level timed Petri nets. As a result, we obtain a formal net semantics for timed interaction and migration which is both structural and allows one to deal directly with concurrency and causality.

Keywords: mobility, local clocks, process algebra, high level Petri nets, syntax driven translation, behavioural equivalence

1 Introduction

The ever increasing complexity of mobile applications requires their effective analysis and verification. Our aim here is to explore formal modelling of mobile distributed systems including also time-related aspects of process migration.

In this paper, we first introduce the T\textsuperscript{M0} (Timed Mobility) model which is a process algebra for mobile systems where – in addition to process mobility and interaction – it is possible to add timers to the basic actions. We provide the syntax and operational semantics of T\textsuperscript{M0} which is a time semantics where each location runs according to its own local clock which is invisible to processes outside this location. Processes are equipped with input and output capabilities which are active up to a predefined time deadline. If such a capability is not taken, an alternative continuation for the process is followed. Another timing constraint allows one to specify the latest time for moving a process from one
location to another. The timeout of such a migration action corresponds to
the network time limit for that action, similar to TTL in TCP/IP network
protocol.

The time model defined for TiMo generalises the one considered in the
theory of structured timed Petri nets [20] because it supports local clocks. In the second
part of the paper, we outline a structural translation of the finite process algebra
terms into behaviourally equivalent finite high level timed Petri nets with local
clocks, similar to that presented in [12]. Such a dual development yields a formal
semantics for explicit mobility and time which is structural and, at the same
time, allows one to deal directly with concurrency and causality which can be
captured in the Petri net domain. The Petri net representation should also be
useful for automatically verifying behavioural properties using suitable modal-
checking techniques and tools.

The paper generalises and extends ideas first described in [9], but here each
location has its local clock which determines the timing of actions executed at
that location. The paper is self-contained, although it would be an advantage
for the reader to be familiar with some basic concepts of process algebras [19],
high level Petri nets [8, 17] and timed Petri nets [21].

The paper is structured in the following way. We first describe the syntax
and semantics of TiMo. After that we introduce the net algebra used in the
translation from TiMo expressions to Petri nets, and then describe the transla-
tion itself. We also explain the nature of behavioural equivalence of the resulting
Petri net model and the original expression.

1.1 Running Example

To introduce the basic concepts of TiMo, we use a simple e-shops (SES) running
example illustrated in Figure 1. In this scenario, we have two customer processes
initially residing in their respective home locations homeA and homeB, and
looking for (the address of) an e-shop where the same desirable e-item can be
purchased. To find this out, each customer moves to the location info in order to
acquire the relevant address (this move takes up to 5 time units). After waiting
for 2 time units at location info without getting the desired address, the e-item
loses its importance and the customer is no longer interested in acquiring it.
The location info contains a broker who knows all about the e-shops stocking
the desired e-item. For up to 5 time units the right e-shop is that at the location
shopA, and after that for up to 7 time units at location shopB (these changes
of availability are cyclical and happen also if a location is communicated to a
customer). It is important to point out that any interaction between processes
can only happen within the same location, and so it is necessary for a customer
to move to the broker location in order to get the desirable address. The timers
can define a coordination in time of the customers, and take care of the relative
time of interaction of the processes residing at the same location.

Figure 1 portrays three possible configurations in an evolution of the running
example in which there are two customers. The active customer is initially residing
in location homeB (configuration C0), and then moving to location info to
acquire the address of an e-shop (configuration $C_1$). After receiving such an address from the broker, the customer moves to the corresponding location $shopA$ (configuration $C_2$).

![Diagram](image)

*Fig. 1. Three configurations in an evolution of the running example (time progress is not represented).*

2 \TiMo: A Process Algebra for Timed Mobility

We start by giving the syntax and semantics of \TiMo which uses timing constraints allowing, for example, to specify what is the longest time it takes a mobile process to move to another location. In \TiMo, waiting for a communication on a channel or a movement to a new location can be constrained. If an action does not happen before a predefined deadline, the waiting process switches its operation to an alternate mode. This approach leads to a method of sharing the channels over time. A timer (such as $\Delta 7$) of an output action $a^{\Delta 7}$ makes it available for communication only for the period of 7 time units. We use timers for both input and output actions. The reason for having the latter stems from the fact that in a distributed system there are both multiple clients and multiple servers, and so clients may decide to switch from one server to another depending on the waiting time.
2.1 Syntax

We assume suitable data types together with associated operations, including a set \textit{Loc} of \textit{locations} and a set \textit{Chan} of \textit{communication channels}. We also use a set \textit{Id} of process identifiers, and each \textit{id} \in \textit{Id} has arity \textit{m}\textit{id}. In what follows, we use \textit{x} to denote a finite tuple of elements \((x_1, \ldots, x_k)\) whenever it does not lead to a confusion.

The syntax of T\textsc{i}M\textsc{o} is given in Table 1, where \textit{P} represents \textit{processes} and \textit{N} represents networks. Moreover, for each \textit{id} \in \textit{Id}, there is a unique process definition of the form:

\[
id(u_1, \ldots, u_{m{id}} : X_i^{id}, \ldots, X_{m{id}}^{id}) \overset{\text{def}}{=} P_id ,
\]

where \textit{P}_id is a process expression, the \textit{u}_i\textit{s} are distinct variables playing the role of parameters, and the \textit{X}_i^{id}\textit{s} are data types. In Table 1, it is assumed that:

- \textit{a} \in \textit{Chan} is a channel, and \textit{t} \in \mathbb{N} \cup \{\infty\} represents a timeout;
- each \textit{v}_i is an expression built from data values and variables;
- each \textit{u}_i is a variable, and each \textit{X}_i is a data type;
- \textit{l} is a location or a location variable; and
- \textit{?} is a special symbol used to state that a process is temporarily 'stalled'.

\begin{table}[ht]
\centering
\begin{tabular}{ll}
\hline
\textbf{Processes} & \textbf{Networks} \\
\textit{P} ::= \textit{a}^{\Delta t}! \langle \textit{v} \rangle \text{ then } \textit{P} \text{ else } \textit{P}' & \textit{N} ::= \textit{l} \square \textit{P} \text{ } | \text{ } \textit{N} \text{ } | \text{ } \textit{N}' \\
\textit{a}^{\Delta t} ? (\textit{u} : \textit{X}) \text{ then } \textit{P} \text{ else } \textit{P}' & \text{ (input) } \\
g o^{\Delta t} \text{ l then } \textit{P} & \text{(move) } \\
\textit{P} \text{ } | \text{ } \textit{P}' & \text{ (parallel) } \\
\textit{id}(\textit{v}) & \text{ (recursion) } \\
\text{stop} & \text{ (termination) } \\
\textit{?} \textit{P} & \text{ (stalling) } \\
\hline
\end{tabular}
\caption{T\textsc{i}M\textsc{o} Syntax. The length of \textit{u} is the same as that of \textit{X}, and the length of \textit{v} in \textit{id}(\textit{v}) is \textit{m}\textit{id}.}
\end{table}

The only variable binding construct is \textit{a}^{\Delta t} ? (\textit{u} : \textit{X}) \text{ then } \textit{P} \text{ else } \textit{P}' which binds the variables \textit{u} within \textit{P} (but \textit{not} within \textit{P}'). We use \textit{fv}(\textit{P}) to denote the free variables of a process \textit{P} (and similarly for networks). For a process definition as in (1), we assume that \textit{fv}(\textit{P}_id) \subseteq \{u_1, \ldots, u_{m{id}}\}, and so the free variables of \textit{P}_id are parameter bound. Processes are defined up to the alpha-conversion, and \{\textit{v}/\textit{u}, \ldots\} \textit{P} is obtained from \textit{P} by replacing all free occurrences of a variable \textit{u} by \textit{v}, etc. possibly after alpha-converting \textit{P} in order to avoid clashes. Moreover, if \textit{v} and \textit{u} are tuples of the same length then \{\textit{v}/\textit{u}\} \textit{P} denotes \{\textit{v}_{1}/\textit{u}_{1}, \textit{v}_{2}/\textit{u}_{2}, \ldots, \textit{v}_{k}/\textit{u}_{k}\} \textit{P}. 
A process $a^{\Delta t}!\{v\} \text{ then } P \text{ else } P'$ attempts to send a tuple of values $v$ over the channel $a$ for $t$ time units. If this is successful, it continues as process $P$; otherwise it continues as the alternative process $P'$. Similarly, $a^{\Delta t}?(u:X) \text{ then } P \text{ else } P'$ is a process that for $t$ time units attempts to input a tuple of values of type $X$ and substitute them for the variables $u$. Mobility is implemented by a process $go^{\Delta t} l$ then $P$ which moves from the current location to the location $l$ within $t$ time units. Note that since $l$ can be a variable, and so its value is assigned dynamically through communication with other processes, migration actions support a flexible scheme for moving processes around a network. Processes are further constructed from the (terminated) process stop and parallel composition $P | P'$. Finally, process expressions of the form $\otimes P$ are a purely technical device which is used in the subsequent formalisation of structural operational semantics of TiMo; intuitively, $\otimes$ specifies a process $P$ which is temporarily (i.e., until a clock tick) stalled and so cannot execute any action. A located process $l[P]$ is a process running at location $l$, and a network is composed out of its components $N | N'$. 

A network $N$ is well-formed if the following hold:

- there are no free variables in $N$;
- there are no occurrences of the special symbol $\otimes$ in $N$;
- assuming that $id$ is as in the recursive equation (1), for every $id(v)$ occurring in $N$ or on the right hand side of any recursive equation, the expression $v_i$ is of type corresponding to $X_i^{id}$ (where we use the standard rules of determining the type of an expression).

One might wonder why a process can delay migration to another location. The point is that by allowing this we can model in a simple way the nondeterminism in the movement of processes which is, in general, outside the control of a system designer. Thus the timer in this case indicates the upper bound on the migration time.

### 2.2 Running Example

The following TiMo specification SES captures the essential features of the running example:

$$
\text{homeA \{ customer(homeA) \} | homeB \{ customer(homeB) \} | info \{ broker \}
$$

where the process identifiers are defined as follows:

\[
customer(home:Loc) \equiv \begin{cases} go^{\Delta 5} \text{ info then} \\
& a^{\Delta 5}?(shop:Loc) \\
& \text{ then go}^{\Delta 2} \text{ shop then stop} \\
& \text{ else go}^{\Delta 5} \text{ home then stop}
\end{cases}
\]

\[
broker \equiv a^{\Delta 5}!(shopA) \text{ then broker' else broker'}
\]

\[
broker' \equiv a^{\Delta 7}!(shopB) \text{ then broker else broker}
\]
\[(\text{Eq1})\quad N | N' \equiv N' | N\]
\[(\text{Eq2})\quad (N | N') | N'' \equiv N | (N' | N'')\]
\[(\text{Eq3})\quad l[P | P'] \equiv l[P] | l[P']\]
\[(\text{Call})\quad l[id(v)] \xrightarrow{id@l} l[\otimes (v/u)P_id]\]
\[(\text{Move})\quad l[go^{\Delta t} l' \text{ then } P] \xrightarrow{\Delta t} l'[\otimes P]\]
\[(\text{Com})\quad \begin{array}{c}
\forall i \in X_1 \ldots \forall k \in X_k
\end{array}
\]
\[
l[\alpha^{\Delta t} (v) \text{ then } P \text{ else } Q] \xrightarrow{\alpha^{\Delta t}} l[\otimes P \mid \otimes (v/u)P']
\]
\[(\text{Par})\quad \begin{array}{c}
N \xrightarrow{\nu} N'
\end{array}
\]
\[
N | N'' \xrightarrow{\nu} N' | N''
\]
\[(\text{Equiv})\quad \begin{array}{c}
N \equiv N' \quad N' \xrightarrow{\alpha} N'' \\
N'' \equiv N'''
\end{array}
\]
\[
N \xrightarrow{\gamma_i} \phi_i(N)
\]

**Table 2.** Three rules of the structural equivalence (Eq1-Eq3), and six action rules (Call Move Com Par Equiv Time) of the operational semantics. In (Par) and (Equiv) \(\psi\) is an action, and in (Time) \(l\) is a location.

### 2.3 Operational Semantics

The first component of the operational semantics of TiMo is the structural equivalence \(\equiv\) on networks. It is the smallest congruence such that the equalities (Eq1–Eq3) in Table 2 hold. Its sole purpose is to rearrange a network in order to apply the action rules which are also given in Table 2. Using (Eq1–Eq3) one can always transform a given network \(N\) into a finite parallel composition of networks of the form:

\[
l_{l_1}[P_1] \mid \ldots \mid l_{n}[P_n]
\]

(2)

such that no process \(P_i\) has the parallel composition operator at its topmost level. Each subnetwork \(l_{l_i}[P_i]\) is called a component of \(N\), the set of all components is denoted by \(\text{comp}(N)\), and the parallel composition (2) is called a component decomposition of the network \(N\). Note that these notions are well defined since component decomposition is unique up to the permutation of the components. This follows from the rule (Call) which treats recursive definitions as function
calls which take a unit of time. Another consequence of such a treatment is that it is impossible to execute an infinite sequence of action steps without executing any local clock ticks. Both these properties would not hold if, instead of an action rule (CALL), we would have a structural rule of the form \( l[\text{id}(v)] = l[\{v/u\}P_{\text{id}}] \).

Table 2 introduces two kinds of operational semantics rules: \( N \xrightarrow{\psi} N' \) and \( N \xrightarrow{\psi!} N' \). The former is an execution of an action \( \psi \) by some process, and the latter a unit time progression at location \( l \). In the rule \( (\text{TIME}) \), \( N \not\rightarrow l \) means that the rules (CALL) and (COM) as well as (MOVE) with \( \Delta t = \Delta t \) cannot be applied to \( N \) for this particular location \( l \). Moreover, \( \phi(N) \) is obtained by taking the component decomposition of \( N \) and simultaneously replacing all the components of the form:

\[
l[l[a^{\Delta t} \omega \text{ then } P \text{ else } Q]] \xrightarrow{\begin{cases} l[l[Q]] & \text{if } t = 0 \\ l[l[a^{\Delta t - t} \omega \text{ then } P \text{ else } Q]] & \text{otherwise} \end{cases}}
\]

where \( \omega \) stands for \(! (v)\) or \(? (u; X)\). After that, all the occurrences of the symbol \( \oplus \) in \( N \) are erased (this is done since processes that migrated need to be activated).

The above defines executions of individual actions. A complete computational step is captured by a derivation of the form:

\[
N \xrightarrow{\Psi} N', \quad (3)
\]

where \( \Psi = \{\psi_1, \ldots, \psi_m\} (m \geq 0) \) is a finite multiset of \( k \)-actions for some location \( l \) (i.e., actions of the form \( \text{id} \oplus l \) or \( l' @ l \) or \( a(v) @ l \)) such that:

\[
N \xrightarrow{\psi_1} N_1 \cdot \cdots \cdot N_{m-1} \xrightarrow{\psi_m} N_m \xrightarrow{\nu} N'. \quad (4)
\]

That is, a derivation is a condensed representation of a sequence of individual actions followed by a clock tick, all happening at the same location. Intuitively, we capture the cumulative effect of the concurrent execution of the multiset of actions \( \Psi \) at location \( l \). We say that \( N' \) is directly reachable from \( N \). Note that whenever there is only a time progression at a location, we have \( N \xrightarrow{\nu} N' \).

The labelled transition system \( \text{Lts}(N) \) of a well-formed network of located processes \( N \) has as its states all the well-formed networks reachable from \( N \) together with arcs labelled by the multisets of actions as given above. Its initial state is \( N \). Note that \( \text{Lts}(N) \) is well-defined thanks to the subsequent Proposition 3.

### 2.4 Running Example

Table 3 presents few derivations of the form (3) for the running example. Moreover, Table 4 shows the way the last derivation in Table 3 has been obtained using a sequence of rules applications as specified in (4).
Table 3. A sequence of derivations for the running example with the parallel subprocess homeA [customer(homeA)] omitted throughout. The first line corresponds to configuration $C_0$ in Figure 1, and the last two lines to configurations $C_1$ and $C_2$, respectively.

\[
\begin{align*}
\text{homeB} & \left[\text{customer(homeB)}\right] \mid \text{info} \left[\text{broker}\right] \\
\text{homeB} & \left[\text{customer(homeB)}\right] \mid \text{info} \left[\alpha^{\text{\text{dis}}} \langle \text{shopA} \rangle \text{ then broker' else broker'}\right] \\
\text{homeB} & \left[\text{go}^{\text{\text{dis}}} \text{ info then } \alpha^{\text{\text{dis}}} ? \langle \text{shop:Loc} \rangle \text{ then go}^{\text{\text{dis}}} \text{ shop then stop} \right] \mid \text{else go}^{\text{\text{dis}}} \text{ homeB then stop} \\
\text{info} & \left[\alpha^{\text{\text{dis}}} \langle \text{shopA} \rangle \text{ then broker' else broker'}\right] \\
\text{homeB} & \left[\text{go}^{\text{\text{dis}}} \text{ info then } \alpha^{\text{\text{dis}}} ? \langle \text{shop:Loc} \rangle \text{ then go}^{\text{\text{dis}}} \text{ shop then stop} \right] \mid \text{else go}^{\text{\text{dis}}} \text{ homeB then stop} \\
\text{info} & \left[\alpha^{\text{\text{dis}}} \langle \text{shopA} \rangle \text{ then broker' else broker'}\right] \\
\text{homeB} & \left[\text{go}^{\text{\text{dis}}} \text{ info then } \alpha^{\text{\text{dis}}} ? \langle \text{shop:Loc} \rangle \text{ then go}^{\text{\text{dis}}} \text{ shop then stop} \right] \mid \text{else go}^{\text{\text{dis}}} \text{ homeB then stop} \\
\text{info} & \left[\alpha^{\text{\text{dis}}} \langle \text{shopA} \rangle \text{ then broker' else broker'}\right] \\
\text{info} & \left[\text{a}^{\text{\text{dis}}} ? \langle \text{shop:Loc} \rangle \text{ then go}^{\text{\text{dis}}} \text{ shop then stop} \right] \mid \text{else go}^{\text{\text{dis}}} \text{ homeB then stop} \\
\text{info} & \left[\alpha^{\text{\text{dis}}} \langle \text{shopA} \rangle \text{ then broker' else broker'}\right] \\
\text{info} & \left[\text{go}^{\text{\text{dis}}} \text{ stop } \mid \text{broker'}\right] \\
\text{homeB} & \left[\text{stop} \mid \text{ info } \left[\text{a}^{\text{\text{dis}}} ? \langle \text{shopB} \rangle \text{ then broker' else broker'}\right]\right] \\
\end{align*}
\]

2.5 Properties

The first two results ensure that derivations are well defined. First, one cannot execute an unbounded sequence of action moves without time progress.

**Proposition 1.** If $N$ is a network and $N \xrightarrow{\psi_1} N_1 \cdots N_{m-1} \xrightarrow{\psi_m} N_m$, then $m \leq |\text{comp}(N)|$.

**Proof.** Each of the components of $N$ is involved in generating at most one $\psi_i$ (since the resulting subexpression is blocked by $\otimes$ until the next time tick), and that the generation of each $\psi_i$ involves at least one component of $N$. 

\[\text{Table 4. Applying operational semantics rules of Table 2 to generate a derivation of the form (3). The parallel subprocess homeA [ customer(homeA) ] is again omitted.}\]

The semantical treatment of TiMo goes beyond interleaving semantics by introducing steps of co-located actions and local time progress in the network evolution. In particular, if we start with a well-formed network, the execution (4) is made up of independent (or concurrent) individual executions. This intuition is reinforced by the following result.

**Proposition 2.** If \( N \) is a well-formed network and \( N \xrightarrow{\Psi} N' \), where \( \Psi = \{ \psi_1, \ldots, \psi_m \} \), then:

\[ N \xrightarrow{\psi_{i_1}} N_1' \cdots N_{i_m} \xrightarrow{\psi_{i_m}} N'_m \xrightarrow{\nu} N' \]

for any permutation \( i_1, \ldots, i_m \) of \( 1, \ldots, m \).

**Proof.** No component is involved in the generation of two \( \psi_i \)'s (since the resulting subexpression is blocked by \( \otimes \) until the next clock tick), and the executions in different components do not interfere with each other.

The third result is that derivations preserve well-formedness of networks.

**Proposition 3.** Networks reachable from a well-formed network are well-formed.

**Proof.** Let \( N \) be a well-formed network and \( N \xrightarrow{\Psi} N' \). Clearly, there are no occurrences of the special symbol \( \otimes \) in \( N' \) since the function \( \phi_i \) implementing the local clock tick removes all its occurrences. We then observe that \( N' \) has no free variables. The only two cases in Table 2 which need to be checked are given by rules \((\text{CALL})\) and \((\text{COM})\). Applying \((\text{CALL})\) does not introduce free variables since \( f_v(P_{id}) \subseteq \{ u_1, \ldots, u_{m_{id}} \} \) in the recursive definition (1). When applying the \((\text{COM})\) rule, the values \( v \) replace \( u \) in \( P' \) and we have \( f_v(P') \subseteq \{ u \} \). Hence \( f_v(\{v/u\}P') = \emptyset \). In the case of an application of the \( \phi_i \) function, we observe that the construct \( a^{\Delta \phi_i}(u;X) \) then \( P \) else \( Q \) binds the variables \( u \) within \( P \), but not within \( Q \). Finally, for every \( id(v) \) occurring in \( N' \), \( v_i \) is of type \( X_{i}^{id} \) which follows from the assumed well-formedness of \( N \).
3 Petri Nets with Location, Time and Mobility

We now introduce an algebra of high level timed Petri nets which will be used to translate TiMo expressions. We focus on nets modelling finite networks (i.e., those without recursive process identifiers). Such a translation still includes all the essential novel features compared to the previous works on Petri net models of process algebras with mobility. The proposed translation which maps finite process expressions into a new class of high level nets — called location, time and mobility nets (or LT$n$M-nets) — has been inspired by the box algebra [5, 6, 13] and timed box algebra [20]. In particular, we use coloured structured tokens and read arcs. The latter allow any number of transitions to simultaneously check for the presence of a resource (represented by a token) stored in a place [8].

There are two kinds of places in LT$n$M-nets:

- Control-flow places are labelled by their status symbols: the internal places by $i$; the entry places by $e$; and the exit places by $x$ and $x'$. The status of a control-flow place is used to specify its initial marking and to determine its role in the net composition operations described later on. Tokens carried by control-flow places are of the form $lt$ where $l$ is the current location of the process thread represented by the token, and $t$ is the age of the token.
- Data places are labelled by data or data variables, and are used as hold data deposited and accessed by process threads.

There are also two kinds of arcs used in LT$n$M-nets: the standard directed arcs (transferring tokens), and the undirected read arcs (checking for the presence of tokens). Arcs can be labelled by one of the following arc-annotations:

$$L:T \quad L:T' \quad L:0 \quad L':0 \quad V \quad V_i \quad (i \geq 1)$$

where $L$, $L'$, $V$ are fixed (Petri net) location variables, $T, T'$ are time variables, and the $V_i$'s are data variables; we call them arc-variables.

An unmarked LT$n$M-net is a triple $\Sigma = (S^{flow} \cup S^{data}, Tr, \iota)$, where $S^{flow}$ and $S^{data}$ are finite disjoint sets of control-flow and data places, respectively; $Tr$ is a finite set of transitions disjoint from $S^{flow}$ and $S^{data}$; $\iota$ is an annotation function defined for the places, transitions, and arcs between places and transitions. We assume that:

- for every control-flow place $s$ in $S^{flow}$, $\iota(s) \in \{e, i, x, x'\}$ gives the status of the place (below $\Sigma$ denotes the set of all the entry places of $\Sigma$);
- for every data place $s$ in $S^{data}$, $\iota(s)$ is a data or data variable (there can be at most one data place with a given label);
- for every transition $tr$ in $Tr$, $\iota(tr)$ is a pair $(\lambda(tr), \gamma(tr))$, where $\lambda(tr)$ is a label of one of the following forms:

$$\tau \quad V @ L \quad a(V_1, \ldots, V_k) \quad a! \quad a?$$

and $\gamma(tr)$ is a boolean guard of one of the following forms:
\[
(T \leq t) \quad (T = t) \quad (T' \leq t') \quad (T' = t') \quad ((T \leq t) \land (T' \leq t'))
\]

with \(t\) and \(t'\) being non-negative integers;

- for every arc \(a\), either undirected \((a = \{s, tr\})\) or directed from a place to a
  transition \((a = \{s, tr\})\) or from a transition to a place \((a = \{tr, s\})\), \(t(a)\) is
  either the empty set or a set comprising one arc-annotation.

A marking \(M\) of \(\Sigma\) is a mapping assigning to each place \(s\) a multiset of tokens. (Note
that even though all the markings in the nets resulting from translation have at most one
token on any place at all times, it is easier to treat them as multisets.) We can compare
markings component-wise, using \(\leq\), and apply the multiset sum \(\oplus\) and difference \(\ominus\),
also component-wise. In diagrams, places are represented by circles, transitions by rectangles,
directed arcs by arrows, read arcs by edges, and markings by tokens inscribed inside places. Arcs
annotated by the empty set are not drawn.

A marked \(\text{itm-net}\) is a pair \((\Sigma, M)\) such that \(\Sigma\) is an \(\text{itm-net}\) and \(M\) is an
initial marking.

If \(\text{Var}\) are the variables occurring in the annotation of a transition \(\text{tr}\) and on the
arcs adjacent to \(\text{tr}\), a binding \(\nu\) assigns to each variable in \(\text{Var}\) a value in its
domain. We only consider legal bindings, i.e., such that for an arc \(a\) between \(\text{tr}\) and
\(s\), if \(h \in \nu(a)\) then the evaluation of \(h\) under the binding \(\nu\) (denoted \(\nu(h)\))
delivers a value allowed in \(s\). The observed label of a transition executed under
the binding \(\nu\) is \(\nu(\lambda(tr))\).

Figure 2 depicts an \(\text{itm-net}\) which will later be derived from a Timo process
expression based on the running example. Note that the mapping \(\nu_1 = \{L \mapsto
hA, V \mapsto \text{inf}, T \mapsto \emptyset\}\) is a legal binding for transition \(\text{tr}\), and \(\nu_2 = \{L \mapsto
\text{inf}, V_1 \mapsto sA, T \mapsto 1, T' \mapsto \emptyset\}\) is a legal binding for \(\text{tr}'\).

### 3.1 Executing \(\text{itm-Nets}\)

Let \(M\) be a marking of an \(\text{itm-net} \Sigma\). To model the passage of time at a location
\(l\), we use the notation \(M^{\nu,l}\) to denote a marking obtained from \(M\) by replacing
each token of the form \(l:t\) by \(l:t+1\). For example, in Figure 3, \(M_1 = M^{\nu,1}_0\).

Now we explain what it means for a transition to be enabled at \(M\), and then
do the same for a group (or step) of transitions. Given a transition \(\text{tr}\) and its
binding \(\nu\), we denote by \(M_{\text{tr.in}}^\nu\) and \(M_{\text{tr.out}}\) two markings of \(\Sigma\) such that, for
every place \(s\),

\[
M_{\text{tr.in}}^\nu(s) = \bigoplus_{h \in \nu(s, tr)} \{\nu(h)\} \quad \text{and} \quad M_{\text{tr.out}}(s) = \bigoplus_{h \in \nu(tr, s)} \{\nu(h)\}.
\]

Intuitively, \(M_{\text{tr.in}}^\nu\) are the tokens consumed by the execution of \(\text{tr}\) under the
binding \(\nu\), and \(M_{\text{tr.out}}\) are the tokens which are produced. We call \((\text{tr}, \nu)\) a
transition instance, and say that \((\text{tr}, \nu)\) is \(l\)-located if \(M_{\text{tr.in}}^\nu\) contains a token of
the form \(l.t\). For example, if we consider transitions \(tr\) and \(tr'\) in Figure 2, and the two binding defined for them above, we obtain:

\[
\begin{align*}
\mathcal{M}^p_{tr, in} &= \{ s_3 \mapsto \{ hA:0 \} \} \\
\mathcal{M}^p_{tr, out} &= \{ s_4 \mapsto \{ \text{inf:0} \} \} \\
\mathcal{M}^p_{tr', in} &= \{ s_3 \mapsto \{ \text{inf:1}, s_4 \mapsto \{ \text{inf:0} \} \} \\
\mathcal{M}^p_{tr', out} &= \{ s_6 \mapsto \{ \text{inf:0} \}, s_7 \mapsto \{ \text{inf:0} \}, s_8 \mapsto \{ sA \} \} .
\end{align*}
\]

A transition instance \((tr, b)\) is enabled at a marking \(\mathcal{M}\) if \(b(\gamma(tr))\) evaluates to \(true\), \(\mathcal{M} \geq \mathcal{M}^p_{tr, in}\) and \(b(h) \in M(s)\), for all \(s\) and \(h \in i(\{ s, tr \})\).
An enabled \( l \)-located transition instance \((tr, b)\) is \( l \)-urgent at \( \mathcal{M} \) if either the label of \( tr \) is of the form \( a(V_1, \ldots, V_k) \), or the label of \( tr \) is \( V@L \) and there is no enabled transition instance \((tr', b')\) in the marking \( \mathcal{M}^\triangledown \) (intuitively, in this case \((tr, b)\) represents a migration that cannot be delayed). For example, \((tr, b_1)\) is the only (non-urgent) transition instance enabled at the initial marking of the \( \text{ltm} \)-net shown in Figure 2.

A local step in location \( l \) at a marking \( \mathcal{M} \) is the union \( \mathcal{W} \) of two sets \( U \) and \( U' \) of \( l \)-located transition instances enabled at \( \mathcal{M} \) such that:

- \( \tau \notin \iota(U) \) and \( \iota(U) \subseteq \{\tau\} \);
- \( \mathcal{M} \geq \bigoplus_{(tr, b) \in \mathcal{W}} \mathcal{M}_{tr, in}^b \);
- there is no transition instance \((tr', b')\) which is \( l \)-urgent at \( \mathcal{M} \) and satisfies \( \mathcal{M} \geq \bigoplus_{(tr, b) \in \mathcal{W}} \mathcal{M}_{tr, in}^b \);
- there is no transition instance \((tr', b')\) which is enabled at \( \mathcal{M} \) and satisfies \( \iota((tr')) = \tau \) and \( \mathcal{M} \geq \bigoplus_{(tr, b) \in \mathcal{W}} \mathcal{M}_{tr, in}^b \).

A local step \( \mathcal{W} \) can be executed leading to a new marking. The resulting evolution is denoted by \( \mathcal{M}[\Psi] \mathcal{M}' \) where:

\[
\Psi = \{b(\lambda(tr)) \mid (tr, b) \in \mathcal{W}\};
\]

\[
\mathcal{M}' = (\mathcal{M} \oplus \bigoplus_{(tr, b) \in \mathcal{W}} \mathcal{M}_{tr, in}^b)^\triangledown \oplus \bigoplus_{(tr, b) \in \mathcal{W}} \mathcal{M}_{tr, out}^b.
\]

Intuitively, a local step is formed by first selecting a set of enabled communication and migration transitions which is maximal w.r.t. \( l \)-urgent transitions, and after that adding a maximal set of \( \tau \)-labelled enabled transitions (corresponding to the expired timers in \( \text{tiMo} \) process expressions).

We then form the labelled transition system \( \text{ltS}(\Sigma, \mathcal{M}) \) in the usual way, using the evolution rule just defined.

Figure 3 shows the markings involved in three successive executions of the \( \text{ltm} \)-net shown in Figure 2, starting from its initial marking:

\[
\mathcal{M}_0 \left[ \Psi_0 \right] \mathcal{M}_1 \left[ \Psi_1 \right] \mathcal{M}_2 \left[ \Psi_2 \right] \mathcal{M}_3
\]

where \( \Psi = \emptyset \) and the only marking change is due to time progression at location \( \inf \) (recall that no transition instance located in \( \inf \) is enabled at the initial marking \( \mathcal{M}_0 \)), \( \Psi_1 = \{(tr, b_1)\} \) and \( \Psi_2 = \{(tr', b_2)\} \).

### 4 An Algebra of \( \text{ltm} \)-Nets

Among possible composition operations which could be defined for \( \text{ltm} \)-nets, three are needed to translate \( \text{tiMo} \) process expressions. The first is a ternary action operation \((\Sigma_1 \text{ then } \Sigma_2 \text{ else } \Sigma_3)\), and the other two are a binary parallel composition \((\Sigma_1 \parallel \Sigma_2)\), and sequential composition \((\Sigma_1 ; \Sigma_2)\). We now assume that

\[
\Sigma_i = (S_i^{\text{flow}} \cup S_i^{\text{data}}, \text{Tr}_i, \text{t}_i) \quad (i = 1, 2, 3)
\]

are unmarked \( \text{ltm} \)-nets with disjoint sets of places and transitions (note that one can always rename the identities of the nodes of different \( \text{ltm} \)-nets to make sure that this condition is satisfied).
Action composition The composition $\Sigma_1$ then $\Sigma_2$ else $\Sigma_3$ is defined if $\Sigma_1$ has a unique x-place $s_1$, and a unique x'-place $r_1$. It is obtained in the following way:

- $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$ are put side by side.
- For every $s_2 \in \Sigma_2$, we create a new place $s'_2$ with the status i and such that each arc $a$ between $s_i$ and $tr \in Tr_i$, for $i \in \{1, 2\}$, is replaced by an arc between $s'_2$ and $tr$ of the same kind (directed to or from, or undirected) and with the same annotation. Then the place $s_1$ of $\Sigma_1$ and the e-places of $\Sigma_2$ are deleted. The same is then done for $r_1$ and the e-places of $\Sigma_3$.
- Data places with the same label are ‘merged’ into a data place with the same label and type (we assume that data places with the same label have also the same type), and with all the arcs and annotations linking them to the transitions in $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$ being inherited by the new data place.
Parallel composition The composition $\Sigma_1 \mid \Sigma_2$ is obtained through the following procedure:

- $\Sigma_1$ and $\Sigma_2$ are put side by side.
- Data places with the same label are merged as in the previous case.

Sequential composition The composition $\Sigma_1 ; \Sigma_2$ is defined if $\Sigma_1$ has a unique $x$-place $s_1$, and no $x'$-places. It is obtained in the following way:

- $\Sigma_1$ and $\Sigma_2$ are put side by side.
- For every $s_2 \in \sigma_2$, we create a new place $s'_2$ with the status i and such that each arc $a$ between $s_i$ and $tr \in Tr_i$, for $i \in \{1, 2\}$, is replaced by an arc between $s'_2$ and $tr$ of the same kind (directed to or from, or undirected) and with the same annotation. Then the place $s_i$ of $\Sigma_i$ and the $x$-places of $\Sigma_2$ are deleted.
- Data places with the same label are merged as in the previous cases.

5 From Networks of Located Processes to Petri Nets

To translate a well-formed network of located finite Timo processes:

$$N = l_1[P_1] \mid \ldots \mid l_n[P_n]$$

we proceed in the following three phases. Below we assume that the only data values occurring in $N$ are location names (data variables are location variables). At the end of the section, we will explain how to deal with the general case.

Phase I For each $i \leq n$, we first translate $P_i$ following its syntax into $K(P_i)$, assuming that actions are translated as follows:

$$K(a^{>\omega} \text{then } P \text{ else } Q) = K(a^{>\omega}) \text{then } K(P) \text{ else } K(Q)$$
$$K(go^{>\omega} \text{then } P) = K(go^{>\omega}) ; K(P),$$

where $\omega$ is $!(v)$ or $?_i(u)$, and $K(a^{>\omega})$ and $K(go^{>\omega})$ are given in Figure 4. Moreover, the translation $K(\text{stop})$ for the terminated process consists of just one $x$-place and one $x'$-place, as shown in Figure 4.

Phase II We take the parallel composition of all the $K(P_i)$'s, and then insert the initial marking, in the following way:

- into each $x$-labelled place originating from $K(P_i)$ we insert a single token $l_i;0$;
- into each $x'$-labelled data place, where $v$ is a data rather than a data variable, we insert a single token $v$. 
**Phase III** For each pair of transitions $tr$ and $tr'$, respectively labelled by $a!$ and $a?$ and having the same number $k$ of adjacent read arcs, we create a new synchronisation transition which inherits the connectivity of both $tr$ and $tr'$. The guard of the new transition is the conjunction of the guards of $tr$ and $tr'$, and the label is $a(V_1, \ldots, V_k)@L$. After that, all transitions labelled by $a!$ or $a?$ are deleted, yielding the result of the whole translation denoted by $\text{PN}(N)$.

Intuitively, when the transition $tr$ of $K(g_0^{\Delta t} v)$ becomes enabled, the $e$-labelled place contains a single token of the form $l:0$, and the $e$-labelled place contains a single token of the form $l'$. The first token can only be removed by the execution of $tr$, whereas the second token cannot be removed and no other token
will arrive later at the \( v \)-labelled place (thanks to the well-formedness property of the network being translated). Moreover, \( l:0 \) can be changed to \( l:1 \) due to time progress at location \( l \), and later to \( l:2 \), etc., until it becomes \( l:3 \) at which point \( tr \) becomes \( l \)-urgent. As a result, \( tr \) keeps being enabled until it is executed before becoming \( l \)-urgent, or immediately after becoming \( l \)-urgent. Then the token in \( e \)-labelled place disappears, and new token \( l':0 \) is inserted into the \( e \)-labelled place (note the common \( V \) in arc-annotations of the read and output arcs adjacent to \( tr \)). The visible label generated by the executed transition \( tr \) is \( l'\oplus l \) on account of the binding \( L \mapsto l \) and \( V \mapsto l' \).

![Diagram](image)

**Fig. 5.** Communication resulting from synchronisation of two basic actions. The label of the synchronised transition \( tr \) is \( a(V_1, \ldots, V_k) \oplus L \) and its guard \( ((T \leq t) \wedge (T' \leq t')) \).

To see the idea behind the basic translations of the other two basic actions, it is best to consider a context in which they occur together creating a synchronisation transitions, as shown in Figure 5. It depicts two \( \tau \)-labelled transitions, \( tr' \) and \( tr'' \), copied from the basic translations, and a new synchronisation transition \( tr \). It is important to bear in mind the original \( a! \)-labelled and \( a? \)-labelled transitions which produced \( tr \) have been deleted after the synchronisation. Now, when a token arrives at place \( e_1 \), it has the form \( l:0 \). At this point, each \( v_2 \)-labelled place holds a single token \( d_i \) (each such token will remain there and no other token will arrive later thanks to the well-formedness property of the network). Token \( l:0 \) can then increase its ‘clock’ part up to \( l:1 \) and if place \( e_2 \) is still empty, transition \( tr' \) becomes enabled, (immediately) \( l \)-urgent, and is then executed. Similar situation happens when a token of the form \( l':0 \) arrives at place \( e_2 \) and no token is present in place \( e_1 \). The only difference is that the \( a \)-labelled places are empty. A really interesting situation, however, happens when two tokens, \( l:0 \) and \( l':0 \), appear in the places \( e_1 \) and \( e_2 \), respectively. If
\( l \neq l' \) then transition \( tr \) cannot be executed (notice the annotations on the arcs from \( e_1 \) and \( e_2 \) to \( tr \)) and only \( tr' \) and \( tr'' \) are executed later. If \( l = l' \) then \( tr \) is executed as an \( l \)-urgent transition, and inserts the value \( d_i \) in each \( u_i \)-labelled place. (Later on, any transition connected to the \( u_i \)-labelled place by a read arc can access \( d_i \).) Moreover, the tokens in \( e_1 \) and \( e_2 \) disappear, and new tokens \( t; 0 \) are inserted into places \( x_1, x_2, x'_1, x'_2 \). The visible label generated by the executed transition \( tr \) is \( a(d_1, \ldots, d_k) \otimes l \) on account of the binding \( L \leftrightarrow l \) and \( V_i \leftrightarrow d_i \).

5.1 Running Example

Figure 2 shows the result of the translation for the following slightly simplified finite version of the running example:

\[
\begin{align*}
hA [ & \text{go} \Delta_5 \inf \text{ then } a \Delta_2 ?(\text{shop}) \text{ then } \text{go} \Delta_3 \text{ shop else stop }] \mid \\
\inf [ & a \Delta_6 !(sA) \text{ then stop else stop } ]
\end{align*}
\]

Here we have only one customer residing initially in location \( hA \) who moves to location \( \inf \) in order to acquire the address of an e-shop. The broker residing in location \( \inf \) can provide such an address for up to six time units. Figure 2 depicts a translation of the above network, except that the exit places are not shown (they are all isolated and unmarked, hence irrelevant). Figure 3 shows markings involved in three successive executions of this system, given earlier on. First, we have a local time progress at location \( \inf \), after that the customer moves from location \( hA \) to location \( \inf \), and then the broker communicates the address \( sA \) of an e-shop.

5.2 Main Result

The soundness of the proposed translation is given by the following result.

**Theorem 1.** The labelled transition system of \( \mathbb{PN}(N) \) is strongly bisimilar in the sense of [18] to the labelled transition system of \( N \). More precisely, there exists a binary relation \( B \) over the vertices of the labelled transition systems \( \text{lts}(N) \) and \( \text{lts}(\mathbb{PN}(N)) \) such that \( (N, M_0) \in B \), where \( M_0 \) is the initial marking of \( \mathbb{PN}(N) \), and if \( (N', M) \in B \) then the following hold:

\(- \ N' \xrightarrow{\Psi} N'' \) implies that \( \mathcal{M}[\Psi]M' \) and \( (N'', M') \in B \) for some marking \( M' \), and

\(- \ \Sigma[\Psi]M' \) implies that \( N' \xrightarrow{\Psi} N'' \) and \( (N'', M') \in B \), for some network \( N'' \).

**Proof.** The proof shares arguments with the proof of a similar result in [12].

The main idea is to observe that the translation from process expressions to Petri nets has been defined ensuring that for every (individual or synchronised) action in the former, one can find a corresponding transition in the latter. It is then a matter of case by case analysis to conclude that two corresponding specifications simulate each other closely. A notable difference is the fact that
in the \texttt{ltm}-net model, the second branch of the communication action construct is implemented by a \(\tau\)-transition, whereas in the process algebra a rewriting is applied. Therefore, the execution of such a \(\tau\)-transition is not recorded in the labelled transition system generated by the \texttt{ltm}-net semantics. More precisely, the result is a consequence of a number of observations outlined below.

First, if we forget about the data places, transition labels, and arc annotations, treating all control-flow tokens as the standard black tokens, then \texttt{ltm}-nets look like the standard Petri boxes and we may rely on some of their properties established in [6]:

- (\texttt{Move}) is simulated by the \(\Sigma; \Sigma'\) composition;
- (\texttt{Cont}) is simulated by two \(\Sigma\) then \(\Sigma'\) else \(\Sigma\) compositions followed by transition synchronisation (involving transitions labelled by \(a!\) and \(a?\));
- (\texttt{Par}) is simulated by the parallel composition;
- the evolutions from the entry marking (one token in each entry place) respect the 1-safety of the control-flow places (i.e., at most one token is ever present on any place), since so do the basic building blocks in Figure 4;
- the introduction of the data places only adds constraints to the basic components, and those constraints are preserved in the compound nets.

We then observe that the data places are also 1-safe. Clearly, this is true in the initial marking. Moreover, the marking of a \(u_s\)-labelled place \(s\) may only be modified by a transition \(tr\) coming from a subnet derived as \(\mathbb{K}(a^{\Delta t}\,(u_1,\ldots,u_k))\), which adds one token to this place. We then observe that by the well-formedness of \(N\), there is at most one term of the kind \(a^{\Delta t}\,(u_1,\ldots,u_k)\) in \(N\). Moreover, there are no loops in the \texttt{ltm}-net generated from a finite expression, the initial marking is 1-safe, and all the initial control-flow tokens are present in the entry places which have no input transitions. Hence there is at most one transition \(tr\) which can modify the marking of \(s\), and \(tr\) can only be executed at most once. In addition, if such a \(tr\) does exist, then \(s\) is initially empty. Another observation concerning the above data place \(s\) is that if it is initially empty and \(tr'\) is a transition connected to it by a read arc, then \(s\) cannot block the enabledness of \(tr'\). This again follows from the assumed well-formedness of \(N\).

We then note that the terminated process is a basic process expression and its translation \(\mathbb{K}\,(\text{stop})\) cannot execute any actions. Moreover, the replacement of \(a^{\Delta t} \omega\) then \(P\) else \(Q\) by \(Q\) is simulated by \(\tau\)-labelled transitions (see Figure 4).

Finally, the replacement of \(a^{\Delta t} \omega\) then \(P\) else \(Q\) by \(a^{\Delta t - 1} \omega\) then \(P\) else \(Q\), and the replacement of \(g^{\Delta t} v\) by \(g^{\Delta t - 1} v\) are simulated by the local time progression of markings.

As a result, the evolutions of process expressions and the corresponding \texttt{ltm}-nets can simulate each other. It is therefore possible to conduct behavioural analyses for each of the two representations, and their results are applicable after suitable interpretations to the other representation as well. For example, by analysing the control-flow tokens in a given marking of the \texttt{ltm}-net representation, we can easily detect whether any process currently resides in a given network location.
5.3 Extending the Translation

We outline now three ways of extending the translation presented earlier in this section.

First, if we allow any data values and data variables to occur in the translated network, each data place of the constructed LTM-net needs to be assigned the type of the corresponding TiMo data value or data variable.

Second, one might want to allow communication of the channels and their dynamic acquisition by migrating processes. This can be achieved by allowing $a$ in the two communication constructs to be a channel variable as well. Then one can translate the generalised input and output prefixes as shown in Figure 6, and in Phase III of the translation synchronise all $?$-labelled transitions with all the $!$-labelled transitions.

![Diagram](image_url)

Fig. 6. Modified translations for generalised action prefixes, where $v$ and $v'$ are channels or channel variables, and $C$ is an arc-variable. If $v$ (or $v'$) is a channel, then the corresponding data place is initialised with a single $v$ token; otherwise it is initially empty.

To deal with recursive TiMo processes, one can adapt the approach introduced in [14]. A key idea is to view a TiMo expression as consisting of a main program together with a number of procedure declarations corresponding to the declarations of process identifiers. The main program is executed once, while each procedure can be invoked several times. Each such invocation is carried out through the execution of a special call-transition corresponding to the (CALL) rule in Table 2. What is crucial, however, is that each invocation is uniquely identified by a structured token derived from the sequence of recursive calls.
along the execution path leading to that invocation. That this sequence is sufficient to identify an invocation follows from the fact that a given call-transition may be activated many times, but each time with a different sequence. The approach uses the notion of a *trail* $\sigma$ to denote a finite (possibly empty) sequence of call-transitions of an iETM-net. The control-flow places carry tokens of the form $l.t:\sigma$. The empty trail corresponds to the tokens flowing through the control-flow places in the translation for finite networks.

Procedure invocation is then possible if each of the input places $s_i$ of a call-transition $\delta$ labelled with $id@L$ contains tokens of the form $l_i.t:\sigma$. The execution results in removing these tokens and inserting a new token $l:0;\sigma\delta$ in each in-
tial (entry) place of the net corresponding to the definition of $\text{id}$. In a similar way, trails are used in the data places to identify in an unambiguous way the invocation to which a given data token belongs, and to match them with the corresponding control-flow tokens.

Figure 7 shows the result of the translation for the following modification of the simplified version of the running example:

$$ ha [ g o^{\Delta 5} \inf \quad a^{\Delta 6} ? (\text{shop}) \quad \text{then} \quad go^{\Delta 8} \text{ shop else stop }] \mid \inf \| br $$

where the process identifier $br$ is defined by:

$$ br \equiv a^{\Delta 6} ! (sA) \quad \text{then} \quad br \quad \text{else stop} $$

Note that the broker stops if there is no request for the address of an e-shop for six local time units since the last communication.

6 Conclusion and Related Work

In this paper we introduced a process algebra called TiMo having processes able to migrate between different locations and timing constraints used to control migration and communication. We used local clocks, provided an operational semantics for the proposed formalism of distributed systems with mobility, and succeeded in translating finite TiMo specifications into a class of high level Petri nets with time. Note also that other useful process operators, such as action hiding, could easily be incorporated into the proposed framework by suitable action renaming or filtering technique. We are not aware of any approach combining in a similar way mobility with timing constraints and local clocks, though our work is clearly related to an extensive body of literature using time in the framework given by process algebras.

Process algebras have been used to model and study distributed concurrent systems in an algebraic framework. A number of highly successful models have been formulated within this framework, including $\lambda C P$ [4], $ccs$ [18], $csp$ [16], distributed $\pi$-calculus [15], and mobile ambients [7]. Several process algebras with timing features were proposed (for instance [1, 11]), but without being able to express process mobility. Time and mobility together are expressed in other formalisms such as the timed $\pi$-calculus [3], timed distributed $\pi$-calculus [10], and timed mobile ambients [2]. Timed distributed $\pi$-calculus uses a similar approach as TiMo, namely using timers to restrict the interaction between processes and to control the availability of various resources; however, it uses a global clock which decrements all the timers [10]. In the timed distributed $\pi$-calculus, the notion of space is flat. A more realistic account of physical distribution is obtained using a hierarchical representation of space, and this is given in [2] by the timed mobile ambients.
References