Modelling and Verification of Timed Interaction and Migration

Gabriel Ciobanu[1] and Maciej Koutny[2]

[1] Faculty of Computer Science
A.I.Cuza University of Iasi
700483 Iasi, Romania
gabriel@info.uaic.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 where timeouts of interactions and adaptable migrations in a distributed environment with explicit locations can be defined. Timing constraints allow to control the interaction (communication) between co-located mobile processes, and a migration action with variable destination supports flexible movement from one location to another. We define an operational semantics, and outline a structural translation of the proposed process algebra into operationally equivalent finite high level timed Petri nets. The purpose of such a translation is twofold. First, it yields a formal semantics for timed interaction and migration which is both compositional and allows to deal directly with concurrency and causality. Second, it should facilitate the use of simulation and verification tools developed within the area of Petri nets.

Keywords: mobility, timers, process algebra, high-level Petri nets, compositional translation, behavioural consistency.

1 Introduction

The increasing complexity of mobile applications means that the need for their effective analysis and verification is becoming critical. Our aim here is to explore formal modelling of mobile distributed systems where one can also specify time-related aspects of migrating processes. To this end, we first introduce the TiMo (Timed Mobility) model which is a simple process algebra for mobile systems where, in addition to process mobility and interaction, it is possible to add timers to the basic actions. Processes are equipped with input and output capabilities which are active up to pre-defined time deadline and, if not taken, another continuation for the process behaviour is chosen. Another timing constraint allows to specify the earliest and latest time for moving a process from one location to another. We provide the syntax and operational semantics of TiMo which is a discrete time semantics incorporating maximally concurrent executions of the basic actions.

The time model defined for TiMo is similar to that considered in the theory of compositional timed Petri nets [21]. Therefore, in the second part of the paper we outline a structural translation of process algebra terms into behaviourally equivalent finite high-level timed Petri nets, similar to those considered in [12].

Such a dual development yields a formal semantics for explicit mobility and time which is compositional, and at the same time, allows one to deal directly with concurrency and causality which can easily be captured in the Petri net domain. The Petri net representation should also be useful for automatically verifying behavioural properties using suitable model-checking techniques and tools.

To introduce the basic concepts of TiMo, we use the *simple e-shops* running example (SES). In this scenario, we have a client process which initially resides in the *home* location, and wants to find an address of an e-shop where different kinds of electronic items (e-items) can be purchased. To find out the address of a suitable e-shop, the client waits for a couple of time units and then, within 5 time units, moves to the location *info* in order to acquire the relevant address. After 7 time units the e-item loses its importance and the client is no longer interested in acquiring it. The location *info* contains a broker who knows all about the availability of the e-shops stocking the desired e-item. In the first 5 time units the right e-shop is the one at the location $shopA$, and after that for 7 time units that at location $shopB$. It is important to point out that we assume that any interaction between processes can only happen within the same location, and so it is necessary for the client to move to the broker location in order to find out about the i-item. The timers can define a coordination in time and space of the client, and take care of the relative time of interaction of the processes residing at the same location.

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

We assume that the reader is familiar with the basic concepts of process algebras [19], high-level Petri nets [9,17], and timed Petri nets [22].

2 A Calculus 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 time window for a mobile process to move to another location. In TiMo, waiting for a communication on a channel or a movement to new location is no longer indefinite. 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 of 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 that *Chan* is a set of channels, *Loc* is a set of locations, *Var* is a set of location variables, and *Ident* is a finite set of process identifiers (each identifier $I \in \mathit{Ident}$ has a fixed arity $m_I \geq 0$). The syntax of TiMo is given below:

$$\begin{array}{rll} P, Q & ::= a^{\Delta t}\,\mathbf{!}\,\langle v\rangle\,\texttt{then}\,P\,\texttt{else}\,Q & \mid \\ & \quad a^{\Delta t}\,\mathbf{?}\,(u)\,\texttt{then}\,P\,\texttt{else}\,Q & \mid \\ & \quad \mathbf{go}^{t\Delta t'}\,v\,\texttt{then}\,P\,\texttt{else}\,Q & \mid \\ & \quad 0 \;\mid\; I(v_1,\ldots,v_{m_I}) & \mid \\ & \quad P\,|\,Q \;\mid\; \#P & \text{(processes)} \\ \\ M, N & ::= l[\![P]\!] \;\mid\; M\,|\,N & \text{(networks of located processes)} \end{array}$$

In the above description, it is assumed that:

- $a \in \mathit{Chan}$ is a channel,
- $t, t' \in \mathbb{N}$ are integers ($t \leq t'$),
- $v, v_1, \ldots, v_{m_I} \in \mathit{Loc} \cup \mathit{Var}$ are locations and/or location variables,
- $l \in \mathit{Loc}$ is a location, and u is a variable.

Moreover, for each process identifier $I \in \mathit{Ident}$, there is a unique definition of the form

$$I(u_1,\ldots,u_{m_I}) = P_I\ , \tag{1}$$

where $u_i \neq u_j$ (for $i \neq j$) are variables acting here as parameters. Given a network of located processes of the form

$$\ldots \quad l[\![\ \ldots\ |\,P\,|\ \ldots\ ]\!] \quad \ldots$$

we call P a *top-level* expression if it does not contain any occurrences of the special symbol #. To improve readability, we will often denote expressions like $\alpha\,\texttt{then}\,0\,\texttt{else}\,0$ and $\alpha\,\texttt{then}\,P\,\texttt{else}\,0$ by α and $\alpha\,\mathbf{.}\,P$, respectively.

Note that since we allow v in the $\mathbf{go}^{t\Delta t'}\,v\,\texttt{then}\,P\,\texttt{else}\,Q$ construct to be a variable and so its value is assigned dynamically though communication with other processes, migration actions support a flexible scheme for movement of processes from one location to another.

Process $a^{\Delta t}\,\mathbf{!}\,\langle v\rangle\,\texttt{then}\,P\,\texttt{else}\,Q$ attempts to send the location given by v over the channel a for t time units. If this happens, then it continues as process P, and otherwise it continues as the alternate process Q. Similarly, process $a^{\Delta t}\,\mathbf{?}\,(u)\,\texttt{then}\,P\,\texttt{else}\,Q$ attempts for t time units to input a location and substitute it for the variable u within its body.

Table 1. Rules of the structural equivalence

(Eq1)	$M \mid N \equiv N \mid M$
(Eq2)	$(M \mid N) \mid N' \equiv M \mid (N \mid N')$
(Eq3)	$l[\![P]\!] \equiv l[\![P \vert 0]\!]$
(Eq4)	$l[\![I(l_1, \ldots, l_{I_A})]\!] \equiv l[\![\{l_1/u_1, \ldots, l_{m_I}/u_{m_I}\}P_I]\!]$
(Eq5)	$l[\![P \vert Q]\!] \equiv l[\![P]\!] \mid l[\![Q]\!]$

Mobility is implemented using processes like $\mathbf{go}^{t\Delta t'}\ u\,\mathtt{then}\,P\,\mathtt{else}\ Q$. It first waits for t time units doing nothing and after that it can move within $t' - t$ time units to the location u and behave as process P. If the move is not carried out in the allowed time window, the process continues as Q at its current location.

Processes are further constructed from the (terminated) process 0 and parallel composition $(P|Q)$. A located process $l[\![P]\!]$ is a process running at location l. Finally, process expressions of the form $\#P$ represent a purely technical notation which is used in our formalisation of structural operational semantics of TiMo; intuitively, it specifies a process P which is temporarily blocked and so cannot execute any action.

The construct $a^{\Delta t}\,\mathbf{?}\,(u)\,\mathtt{then}\,P\,\mathtt{else}\ Q$ binds the location variable u within P (but *not* within Q), and $fv(P)$ are the free variables of a process P (and similarly for networks of located processes). Processes are defined up to the alpha-conversion, and $\{l/u, \ldots\}P$ is obtained from P by replacing all free occurrences of u by l, etc, possibly after alpha-converting P in order to avoid clashes. For a process definition as in (1), we assume that $fv(P_I) \subseteq \{u_1, \ldots, u_{m_I}\}$ and so the free variables of P_I are parameter bound.

A network of located processes is *well-formed* if no variable used in both the network definition and process identifier definitions (1) is both free and bound, no variable ever generates more than one binding, there are no free variables in the network, and there are no occurrences of the special blocking symbol #.

The specification of the running example which captures the essential features of the scenario described in the introduction can then be written down in the following way:

$$SES \;=\; home[\![Client]\!] \mid info[\![Broker]\!]$$

where

$$Client \;= \mathbf{go}^{2\Delta 5}\ info\ .\,(a^{\Delta 2}\,\mathbf{?}\,(shop)\,\mathtt{then}\,\mathbf{go}^{0\Delta 0}\ shop\,\mathtt{else}\,\mathbf{go}^{0\Delta 0}\ home\,)$$
$$Broker = a^{\Delta 5}\,\mathbf{!}\,\langle shopA\rangle\,\mathtt{then}\,0\,\mathtt{else}\ a^{\Delta 7}\,\mathbf{!}\,\langle shopB\rangle$$

Table 2. Rules of the operational semantics, where $\beta \neq \surd$

(Move) $$k[\![\mathbf{go}^{0\Delta t}\, l\, \mathtt{then}\, P\, \mathtt{else}\, Q\,]\!] \xrightarrow{k:l} l[\![\#P]\!]$$

(Wait) $$k[\![\mathbf{go}^{0\Delta t}\, l\, \mathtt{then}\, P\, \mathtt{else}\, Q\,]\!] \xrightarrow{\tau} k[\![\#\mathbf{go}^{0\Delta t}\, l\, \mathtt{then}\, P\, \mathtt{else}\, Q\,]\!]$$

(Com) $$k[\![a^{\Delta t}\,\mathbf{!}\,\langle l\rangle\, \mathtt{then}\, P\, \mathtt{else}\, Q \mid a^{\Delta t'}\,\mathbf{?}\,(u)\, \mathtt{then}\, P'\, \mathtt{else}\, Q'\,]\!] \xrightarrow{k:a(l)} l[\![\#P \mid \#\{l/u\}P']\!]$$

(Par) $$\frac{N \xrightarrow{\beta} N'}{N \mid M \xrightarrow{\beta} N' \mid M}$$

(Struc) $$\frac{N \equiv N' \qquad N \xrightarrow{\beta} M \qquad M \equiv M'}{N' \xrightarrow{\beta} M'}$$

(Time) $$\frac{N \not\rightarrow}{N \xrightarrow{\surd} \phi(N)}$$

2.2 Operational Semantics

The first component of the operational semantics of TiMo is the structural equivalence $\equiv$ on networks, similar to that used in [7]. It is the smallest congruence such that the equalities in Table 1 hold (note that $\{l_1/u_1, \ldots, l_{m_I}/u_{m_I}\}$ there denotes simultaneous substitution).

The action rules of the operational semantics are given in Table 2. They are based on the structural equivalence defined in Table 1 and two kinds of transition rules:

$$M \xrightarrow{\beta} N \quad \text{and} \quad M \xrightarrow{\surd} N$$

where the former is recording an execution of an action β, and the latter is a time step. The action β can be $k : l$ or $k : a(l)$ or τ, where k is the location where the action is executed and l is either the location a process has moved to, or the location name communicated between two processes over the channel a. Moreover, τ indicates that a process which could have moved to another location decided to wait in the same location for another time unit.

In the rule (Time), $N \not\to$ denotes the fact that no other rule in Table 2 can be applied. Moreover, $\phi(N)$ is obtained from N in the following *consecutive* stages:

- Each top-level expression $I(l_1, \ldots, l_{I_A})$ is replaced by the corresponding definition $\{l_1/u_1, \ldots, l_{m_I}/u_{m_I}\}P_I$.
- Each top-level expression of the form

$$a^{\Delta 0} \ldots \,\texttt{then}\, P \,\texttt{else}\, Q \qquad \text{or} \qquad \mathbf{go}^{0\Delta 0} \ldots \,\texttt{then}\, P \,\texttt{else}\, Q$$

  is replaced by $\#Q$.
- Each top-level expression of the form $a^{\Delta t}\,!\,\langle l\rangle \,\texttt{then}\, P \,\texttt{else}\, Q$ is replaced by $a^{\Delta(t-1)}\,!\,\langle l\rangle \,\texttt{then}\, P \,\texttt{else}\, Q$.
- Each top-level expressions of the form $a^{\Delta t}\,?\,(u) \,\texttt{then}\, P \,\texttt{else}\, Q$ is replaced by $a^{\Delta(t-1)}\,?\,(u) \,\texttt{then}\, P \,\texttt{else}\, Q$.
- Each top-level expression of the form $\mathbf{go}^{t\Delta t'} \,\texttt{then}\, P \,\texttt{else}\, Q$ is replaced by $\mathbf{go}^{t''\Delta(t'-1)} \,\texttt{then}\, P \,\texttt{else}\, Q$ where $t'' = \max\{0, t-1\}$.
- All occurrences of the special symbol # are deleted.

Note that $\phi(N)$ is a well-formed network of located processes provided that we started from a well-formed network in the first place (see below). For the running example, a possible execution corresponding is given in Table 3.

The way networks of located processes evolve can be regarded as conforming to the maximally concurrent paradigm. If we start with a well-formed network, execution proceeds through alternating executions of time steps and contiguous sequences of actions making up what can be regarded as a maximally concurrent

Table 3. An execution for the running example

$$SES \xrightarrow{\surd}\xrightarrow{\surd}\xrightarrow{home:info}$$

$$info[\![\#P \mid a^{\Delta 3}\,!\,\langle shopA\rangle \,\texttt{then}\, 0 \,\texttt{else}\, a^{\Delta 7}\,!\,\langle shopB\rangle\,]\!] \xrightarrow{\surd}$$

$$info[\![P \mid a^{\Delta 2}\,!\,\langle shopA\rangle \,\texttt{then}\, 0 \,\texttt{else}\, a^{\Delta 7}\,!\,\langle shopB\rangle\,]\!] \xrightarrow{info:a(shopA)}$$

$$info[\![\#\mathbf{go}^{0\Delta 0}\, shopA \mid \#0]\!] \xrightarrow{\surd}$$

$$info[\![\mathbf{go}^{0\Delta 0}\, shopA \mid 0]\!] \xrightarrow{info:shopA}$$

$$shopA[\![\#0]\!] \mid info[\![0]\!] \xrightarrow{\surd}$$

$$shopA[\![0]\!] \mid info[\![0]\!]$$

where $\qquad P = a^{\Delta 2}\,?\,(shop) \,\texttt{then}\, \mathbf{go}^{0\Delta 0}\, shop \,\texttt{else}\, \mathbf{go}^{0\Delta 0}\, home$

step (note the role of the special blocking symbols $\#$). Now, if N is a well-formed networks and we have

$$N \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_k}\xrightarrow{\surd} M$$

then M is also a well-formed network. Moreover, we write $N \xrightarrow{\Gamma} M$, where Γ is the multiset comprising all β_i's different from τ, and call M *directly reachable* from N. In other words, we capture the cumulative effect of execution.

Then the labelled transition system $\mathsf{ts}(N)$ of a well-formed network of located processes N has as its states all well-formed networks directly and indirectly reachable from N together with all possible arcs labelled by the multisets of observed actions. The initial state is itself the network N.

We can define a barbed (observation) predicate $N \downarrow_l$ for networks of located processes expressing that N has as component a (non-zero) process located at l. For timed mobile ambients we have defined a barbed predicate $\downarrow_{n@k}$ and a global predicate $\downarrow_n$ in [2]. However here we have only locations, and so we cannot define a global predicate. Also, the definition of $N \downarrow_l$ is rather trivial; it only describes something which can be immediately seen from the description of N.

3 An Algebra of Nets

We now outline an algebra of high-level timed Petri nets which we will then use to render TiMo networks of located processes. We focus on nets modelling finite networks, as this translation still includes all the essential novel features, and the case of networks involving process identifiers (more precisely, recursive process identifiers, as any non-recursive network specification can be transformed so that no process identifier is used) can be treated similarly as in [12]. The current development, resulting in *tm-nets*, has been inspired by the box algebra [5,6,13] and timed box algebra [21].

We use coloured tokens and read-arcs allowing any number of transitions to simultaneously check for the presence of a resource stored in a place [9]. The latter feature is crucial as we aim at defining a step semantics for tm-nets based on their maximally concurrent executions.

There are two kinds of places in tm-nets:

– *Control flow places*
  These model control flow and are labelled by their status symbols: (i) the *internal* places by i; (ii) the *entry* places by e; and (iii) 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 describe later on. The tokens carried by control flow places are of the form $l{:}t$ where l is the current location of the thread represented by the token, and t is the age of the token.
– *Location places*
  Each such place is labelled by a location (or location variable) and is used as a store for process locations which can then be accessed by process threads.

There are two kinds of arcs used in tm-nets, the standard directed arcs and read arcs used for token testing, which can be labelled by one of the following annotations: $L{:}T$, $L{:}T'$, L', $L{:}0$, $L'{:}0$, where L, L' are (Petri net) location variables and T, T' are time variables.

An (unmarked) *tm-net* is a triple $\Sigma = (S^{\mathit{flow}} \uplus S^{loc}, Tr, \iota)$, where: S^{flow} and S^{loc} are finite disjoint sets of, respectively, control-flow and location *places*; Tr is a finite set of *transitions* disjoint from S^{flow} and S^{loc}; ι is an *annotation function* defined for the places, transitions, and arcs between places and transitions. The arcs may be either directed (transferring tokens) or undirected (checking for the presence of tokens). We assume that:

– For every s in S^{flow}, $\iota(s) \in \{\mathsf{e}, \mathsf{i}, \mathsf{x}, \mathsf{x}'\}$ is a label giving the status of the place, determining its role during the application of composition operators. In what follows, ${}^{\circ}\Sigma$ is the set of all entry places of Σ (forming its entry interface).
– For every s in S^{loc}, $\iota(s)$ is a location or location variable. At most one location place with a given label is allowed to be present in Σ.
– For every tr in Tr, $\iota(tr)$ is a pair $(\lambda(tr), \gamma(tr))$ where $\lambda(tr)$ is its *label* (a term with variables representing what is visible from the outside when the transition fires) and $\gamma(tr)$ is a *guard* (giving a timing constraint for the executability of the transition).
– For every arc $\mathfrak{a}$, either undirected ($\mathfrak{a} = \{s, tr\}$) or directed from a place to a transition ($\mathfrak{a} = (s, tr)$) or from a transition to a place ($\mathfrak{a} = (tr, s)$), $\iota(\mathfrak{a})$ is an annotation which is a set of terms with variables.[1]

As usual, if V are the variables occurring in the annotation of a transition tr and on the arcs adjacent to tr, we shall denote by $\flat$ a *binding* assigning to each variable in V a value in its domain. We shall only consider *legal bindings*, i.e., such that for each arc $\mathfrak{a}$ between t and s, if $\hbar \in \iota(\mathfrak{a})$, the evaluation of $\hbar$ under the binding $\flat$ (denoted $\flat(\hbar)$) delivers a value allowed in s. The observed label of a transition fired under binding $\flat$ is $\flat(\lambda(tr))$.

A *marking* $\mathcal{M}$ of a tm-net Σ is a function assigning to each place s a multiset of tokens.[2] Below we use $\oplus$ and $\ominus$ to denote respectively multiset sum and difference. Moreover, if $\mathcal{M}$ and $\mathcal{M}'$ are multisets over the same set of elements Z then $\mathcal{M} \geq \mathcal{M}'$ means that $\mathcal{M}(z) \geq \mathcal{M}'(z)$, for all $z \in Z$.

In what follows, a *marked* tm-net is a pair $(\Sigma, \mathcal{M})$ where Σ is a tm-net and $\mathcal{M}$ is its *initial* marking.

3.1 Firing Rule for tm-Nets

Let $\mathcal{M}$ be a marking of a tm-net Σ. As usual for Petri nets, we first need to say what it means for a transition of Σ to be enabled at the marking $\mathcal{M}$.

[1] In the nets resulting from the translation, all such sets are either empty or singletons.
[2] 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.

Given a transition tr of Σ and a binding $\flat$ for tr, we denote by $\mathcal{M}^{\flat}_{tr,in}$ and $\mathcal{M}^{\flat}_{tr,out}$ two markings of Σ defined in such a way that, for every place s,

$$\mathcal{M}^{\flat}_{tr,in}(s) = \bigoplus_{\hbar \in \iota((s,tr))} \{\flat(\hbar)\} \quad \text{and} \quad \mathcal{M}^{\flat}_{tr,out}(s) = \bigoplus_{\hbar \in \iota((tr,s))} \{\flat(\hbar)\} \; .$$

Intuitively, $\mathcal{M}^{\flat}_{tr,in}$ represents all the tokens which are consumed by the firing of tr under the binding $\flat$, and $\mathcal{M}^{\flat}_{tr,out}$ represents all the tokens which are produced by the firing of tr under the same binding.

A transition tr is *enabled* under the binding $\flat$ at marking $\mathcal{M}$ if the following are satisfied:

– $\flat(\gamma(tr))$ evaluates to $true$,
– for every place s, $\mathcal{M}(s) \geq \mathcal{M}^{\flat}_{tr,in}(s)$,
– for every place s and every $\hbar \in \iota(\{s, tr\})$, $\flat(\hbar) \in \mathcal{M}(s)$.

An enabled transition tr may then be *fired*, which transforms $\mathcal{M}$ into a new marking $\mathcal{M}'$ in such a way that, for each place s:

$$\mathcal{M}'(s) \;=\; \mathcal{M}(s) \ominus \mathcal{M}^{\flat}_{tr,in}(s) \oplus \mathcal{M}^{\flat}_{tr,out}(s) \; .$$

This is denoted by $(\Sigma, \mathcal{M}) \xrightarrow{\flat(\lambda(tr))} (\Sigma, \mathcal{M}')$. Actions of this type will be used in the generation of the labelled transition system generated by a tm-net executed from its initial marking.

To define the desired labelled transition system, we still need to incorporate two aspects: maximally concurrent execution of actions, and the timing aspects. The first one is captured using the step semantics of Petri nets. More precisely, a *computational step* of $(\Sigma, \mathcal{M})$ is derived in the following way:

– We select a set of transitions U such that they form a valid *step* of transitions in the usual Petri net sense (i.e., each transition can be fired in isolation, and there are enough resources for all of them to be executed simultaneously). Moreover, none of the transitions is labelled by τ, and one cannot extend U by adding a transition labelled by $L{:}a(L')$ and still have a valid step for the marking $\mathcal{M}$. The firing of U results in an intermediate marking $\mathcal{M}''$.
– We change $\mathcal{M}''$ by replacing each token of the form $l{:}t$ by $l{:}(t+1)$ which gives a new marking $\mathcal{M}'''$.
– We take the set U' of all enabled τ-labelled transitions (in the tm-nets resulting from the translation described below, U' is always a valid step). The firing of U' results in a marking $\mathcal{M}'$.

We then denote $(\Sigma, \mathcal{M}) \xrightarrow{\Gamma} (\Sigma, \mathcal{M}')$ where Γ is the multiset comprising all the labels generated by the transitions in U.

Note that the above definition treats some transitions as urgent, and the other as non-urgent to reflect the fact that in the process algebra some of the actions can be postponed while the others cannot.

We then can form the transition system $\mathsf{ts}(\Sigma, \mathcal{M})$ of a tm-net $(\Sigma, \mathcal{M})$ in the usual way, using the Γ-labelled executions based on the last definition.

3.2 Composing tm-Nets

Among the various operations which can be defined for the tm-nets, two are needed for the translation of a relatively simple TiMo process algebra. The first is a ternary *action* operation ($\Sigma_1 \, \texttt{then} \, \Sigma_2 \, \texttt{else} \, \Sigma_3$), and the other one a binary *parallel composition* ($\Sigma_1 | \Sigma_2$). We now assume that

$$\Sigma_i = (S_i^{flow} \uplus S_i^{loc}, Tr_i, \iota_i) \qquad (i = 1, 2, 3)$$

are unmarked tm-nets with disjoint sets of places and transitions (note that one can always rename the identities of the nodes of different tm-nets to make sure that this condition is satisfied).

action composition. The composition $\Sigma_1 \, \texttt{then} \, \Sigma_2 \, \texttt{else} \, \Sigma_3$ is defined if Σ_1 has a unique e-place, a unique x-place s_1, and a unique x'-place r_1. It is obtained in the following:

- Σ_1, Σ_2 and Σ_3 are put side by side.
- For every $s_2 \in {}^\circ\Sigma_2$, we create a new place s_2' with the status i and such that each arc $\mathfrak{a}$ between s_i and $tr \in Tr_i$, for $i \in \{1, 2\}$, is replaced by an arc of the same kind (directed to or from, or undirected) and with the same annotation, between s_2' and tr. Then s_1 and the e-places of Σ_2 are deleted. The same is then done for r_1 and the e-places of Σ_3.
- Location places with the same label are 'merged' into a location place with the same label, and with all the arcs and annotations linking them to the transitions in Σ_1, Σ_2 and Σ_3 being inherited by the new location place.

parallel composition. The composition $\Sigma_1 | \Sigma_2$ is obtained through the following procedure:

- Σ_1 and Σ_2 are put side by side.
- Location places with the same label are merged as in the previous case.

4 From Networks of Located Processes to Nets

We translate the following well-formed network of located processes:

$$N \;=\; l_1[\![P_1]\!] \;|\; \ldots \;|\; l_n[\![P_n]\!]$$

The translation is carried out in the following three phases:

Phase I. For each $i \leq n$, we first translate P_i compositionally into $\mathbb{K}(P_i)$, assuming that actions are translated as follows:

$$\mathbb{K}(\alpha \, \texttt{then} \, P \, \texttt{else} \, Q) = \mathbb{K}(\alpha) \, \texttt{then} \, \mathbb{K}(P) \, \texttt{else} \, \mathbb{K}(Q) \ .$$

where $\mathbb{K}(\alpha)$ is given in Figure 1. Moreover, the translation for the terminated process 0 consists of just two isolated control flow places, one e-place and one x-place.

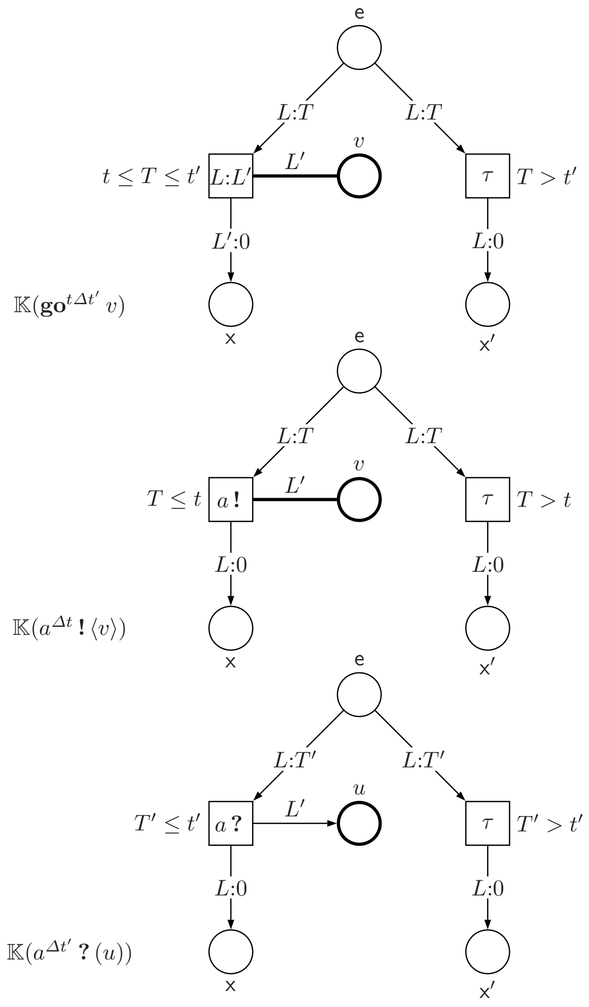

Fig. 1. Basic translations. Note that location places and read arcs are represented by thicker lines.

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

- into each e-labelled place originating from $\mathbb{K}(P_i)$ we insert a single token $l_i{:}0$,
- into each l-labelled location place (where l is a location rather than a location variable) we insert a single token l.

Phase III. For each pair of transitions, tr and tr', respectively labelled by $a\,\mathbf{!}$ and $a\,\mathbf{?}$, we create a new synchronisation transition which inherits the connectivity of the both tr and tr'. The guard of the new transition is the conjunction of the guards of tr and tr', and the label is $L{:}a(L')$. After that all transitions labelled by $a\,\mathbf{!}$ or $a\,\mathbf{?}$ are deleted, yielding the result of the whole translation denoted by $\mathbb{PN}(N)$.

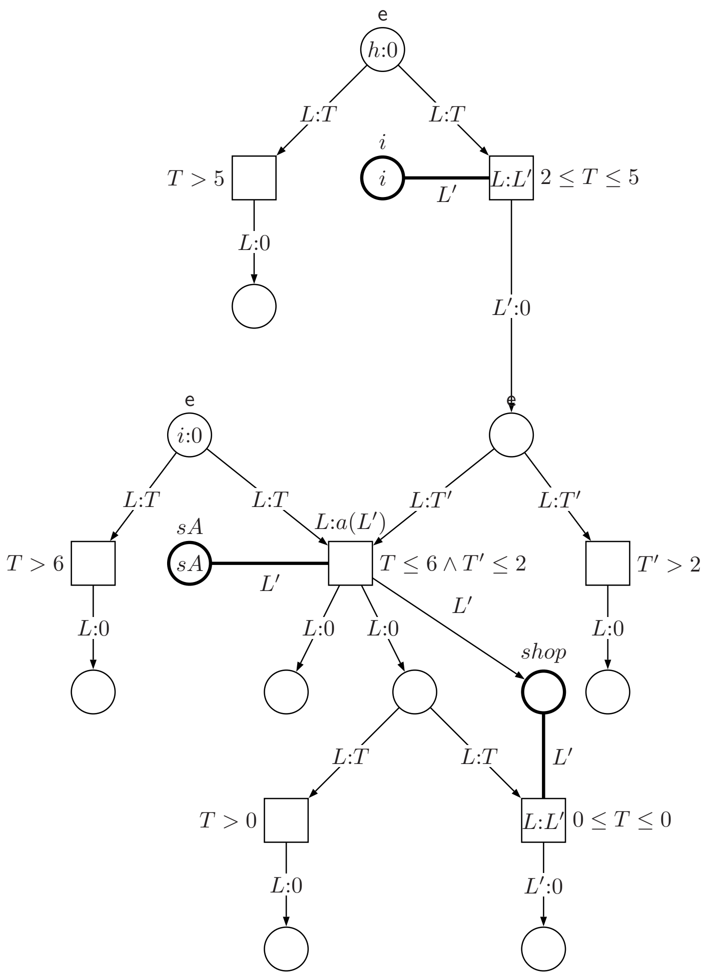

Fig. 2. An example of translation from TiMo to tm-nets. To improve readability, the exit places are not shown (they are all isolated and unmarked), and the labels of the internal places as well as τ labels are omitted.

Figure 2 shows the result of the three-phrase translation for a slightly simplified version of the running example:

$$h[\![\mathbf{go}^{2\Delta 5}\ i\ \mathbf{.}\,(a^{\Delta 2}\,\mathbf{?}\,(shop)\,\mathtt{then}\,\mathbf{go}^{0\Delta 0}\,shop\,\mathtt{else}\ 0\,)]\!]\ |\ i[\![a^{\Delta 6}\,\mathbf{!}\,\langle sA\rangle]\!]$$

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

Theorem 1. *The transition system of* $\mathbb{PN}(N)$ *is strongly bisimilar in the sense of [18] to the transition system of* N.

As a consequence, the evolutions of process expressions and the corresponding tm-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 tm-net representation, we can easily detect whether any process currently resides in a given network location.

The proof of Theorem 1 has similarities with the proof of a result of [12]. The main idea is to observe that the way translation from process expressions to Petri nets has been defined ensures 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 very closely. A notable difference is the fact that in the tm-net model the fact that the second branch of the action construct has been taken is signified by a τ-transition, whereas in the process algebra a rewriting is applied. Therefore, firing of such a τ-transition is not recorded in the labelled transition system generated by the tm-net semantics.

5 Conclusions and Related Work

Process algebras have long 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 ACP [4], CCS [18], CSP [16], distributed π-calculus [15], and mobile ambients [8]. However, none was able to capture properties of timing in distributed systems in a natural way. Process algebras with timing features were presented in [1,11,14,20], but without being able to express process mobility. Mobility can be expressed by other formalisms, such as the timed π-calculus [3], timed distributed π-calculus [10], and timed mobile ambients [2]. Timed distributed π-calculus uses a relative time of interaction given by timers, and a global clock which decrements the timers [10]. Timers are used to restrict the interaction between components, and both typing and timers are used to control the availability of resources. In the timed distributed π-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.

In this paper we introduced a simple TiMo process algebra where we have explicit mobility and can specify timers for the basic actions. We succeeded in translating finite TiMo specifications into the class of tm-nets which are high level Petri nets with time. In the future work, we plan to treat other salient features of distributed systems, including general data transfer and typing of channels.

Acknowledgement

We would like to thank the anonymous referees for their very helpful and constructive comments. This research was supported by the EC IST grant 511599 (Rodin), the NSFC project 60433010, and the CEEX grant Idei 402/2007.

References

1. Aceto, L., Murphy, D.: Timing and Causality in Process Algebra. Acta Informatica 33, 317–350 (1996)
2. Aman, B., Ciobanu, G.: Mobile Ambients with Timers and Types. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) ICTAC 2007. LNCS, vol. 4711, pp. 50–63. Springer, Heidelberg (2007)
3. Berger, M.: Basic Theory of Reduction Congruence forTwo Timed Asynchronous π-Calculi. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 115–130. Springer, Heidelberg (2004)
4. Bergstra, J.A., Klop, J.W.: Process Theory based on Bisimulation Semantics. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. LNCS, vol. 354, pp. 50–122. Springer, Heidelberg (1989)
5. Best, E., Fraczak, W., Hopkins, R.P., Klaudel, H., Pelz, E.: M-nets: an Algebra of High Level Petri Nets, with an Application to the Semantics of Concurrent Programming Languages. Acta Informatica 35, 813–857 (1998)
6. Best, E., Devillers, R., Koutny, M.: Petri Net Algebra. In: EATCS Monographs on TCS, Springer, Heidelberg (2001)
7. Bettini, L., et al.: The KLAIM Project: Theory and Practice. In: Priami, C. (ed.) GC 2003. LNCS, vol. 2874, Springer, Heidelberg (2003)
8. Cardelli, L., Gordon, A.: Mobile Ambients. Teoretical Computer Science 240, 170–213 (2000)
9. Christensen, S., Hansen, N.D.: Coloured Petri Nets Extended with Place Capacities, Test Arcs and Inhibitor Arcs. In: Ajmone Marsan, M. (ed.) ICATPN 1993. LNCS, vol. 691, Springer, Heidelberg (1993)
10. Ciobanu, G., Prisacariu, C.: Timers for Distributed Systems. Electronic Notes in Theoretical Computer Science 164, 81–99 (2006)
11. Corradini, F.: Absolute Versus Relative Time in Process Algebras. Information and Computation 156, 122–172 (2000)
12. Devillers, R., Klaudel, H., Koutny, M.: A Petri Net Semantics of a Simple Process Algebra for Mobility. Electronic Notes in Theoretical Computer Science 154, 71–94 (2006)
13. Devillers, R., Klaudel, H., Koutny, M., Pommereau, F.: Asynchronous Box Calculus. Fundamenta Informaticae 54, 295–344 (2003)
14. Gorrieri, R., Roccetti, M., Stancampiano, E.: A Theory of Processes with Durational Actions. Theoretical Computer Science 140, 73–94 (1995)
15. Hennessy, M., Regan, T.: A Process Algebra for Timed Systems. Information and Computation 117, 221–239 (1995)
16. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Englewood Cliffs (1985)
17. Jensen, K., Rozenberg, G. (eds.): High-level Petri Nets. Theory and Application. Springer, Heidelberg (1991)

18. Milner, R.: Communication and Concurrency. Prentice Hall, Englewood Cliffs (1989)
19. Milner, R.: Communicating and Mobile Systems: the π-calculus. Cambridge University Press, Cambridge (1999)
20. Moller, F., Tofts, C.: A temporal Calculus of Communicating Systems. In: Groote, J.F., Baeten, J.C.M. (eds.) CONCUR 1991. LNCS, vol. 527, pp. 401–415. Springer, Heidelberg (1991)
21. Niaouris, A.: An Algebra of Petri Nets with Arc-Based Time Restrictions. In: Liu, Z., Araki, K. (eds.) ICTAC 2004. LNCS, vol. 3407, pp. 447–462. Springer, Heidelberg (2005)
22. Starke, P.: Some Properties of Timed Nets under the Earliest Firing Rule. In: Rozenberg, G. (ed.) APN 1989. LNCS, vol. 424, Springer, Heidelberg (1990)