Behavioural Equivalences over Migrating Processes with Timers

Bogdan Aman[1], Gabriel Ciobanu[1], and Maciej Koutny[2]

[1] Romanian Academy, Institute of Computer Science
and "A.I.Cuza" University, 700506 Iaşi, Romania
bogdan.aman@gmail.com, gabriel@info.uaic.ro
[2] School of Computing Science, Newcastle University
Newcastle upon Tyne, NE1 7RU, United Kingdom
maciej.koutny@newcastle.ac.uk

Abstract. The temporal evolution of mobile processes is governed by independently operating local clocks and their migration timeouts. We define a formalism modelling such distributed systems allowing (maximal) parallel execution at each location. Taking into account explicit timing constraints based on migration and interprocess communication, we introduce and study a number of timed behavioural equivalences, aiming to provide theoretical underpinnings of verification methods. We also investigate relationships between such behavioural equivalences.

Keywords: mobility, timer, process algebra, bisimulation, behaviour, equivalence.

1 Introduction

Process calculi are a family of formalisms used to model distributed systems. They provide algebraic laws allowing a high-level description and analysis of concurrent processes, behavioural equivalences (e.g., bisimulations) between processes, and automated tools for the verification of interaction, communication, and synchronization between processes. During the past couple of decades, a number of calculi supporting process mobility were defined and studied; in particular, π-calculus [16] and mobile ambients [5]. Various specific features were introduced to obtain such formalisms, including explicit locations in distributed π-calculus [14], explicit migration and timers in timed distributed π-calculus [12], and timed mobile ambients [1]. Time is an important aspect of distributed computing systems, and can play a key role in their formal description. Since time is a complex subject, its introduction to the domain of process calculi has received a lot of attention in, e.g., [2,3,6,15,17,21]. Papers like these assume the existence of a global clock which is usually required in the description of complex systems. However, there are several applications and systems for which considering a global clock would be inappropriate. This paper follows such an approach,

H. Giese and G. Rosu (Eds.): FMOODS/FORTE 2012, LNCS 7273, pp. 52–66, 2012.

in which local clocks operate independently, and so processes at different locations evolve asynchronously. Overall, temporal evolution of mobile processes is governed by independent local clocks and their migration timeouts.

The ever increasing complexity of mobile processes calls for the development of effective techniques and tools for the automated analysis and verification of their properties including, in particular, behavioural equivalences between systems. Bisimulation is the most common mathematical concept used to capture behavioural equivalence between processes. The corresponding equivalence relation, called bisimilarity, is used to abstract from certain details of the systems, and is widely accepted as a standard behavioural equivalence for different kinds of computational processes. Several kinds of bisimulations had been defined (e.g, strong or weak bisimulation for π-calculus [16]).

In the paper [9], we defined TiMo, a basic language for mobile systems in which it is possible to add timers to control process mobility and interaction. After that, in [11], a local clock was assigned to each location of a system modelled in TiMo. Each such clock determines the timing of actions executed at the corresponding location. Then, starting from TiMo, we created a flexible software platform supporting the specification of agents and their physical distribution, allowing also a timed migration in a distributed environment [8]. We obtained this implementation by using an advanced software technology, creating a platform for mobile agents with time constraints.

In this paper, we consider TiMo as the specification language for mobile agents with timeouts, and define various behavioural equivalences taking into account the timers which control the execution of communication and migration actions. We study these bisimulations over networks, and then relax them by applying the so-called 'up-to' technique. In [10] we discussed the *TravelShop* example, in which clients buy tickets to predefined destinations from travel agents. One can use the bisimilarities defined in this paper to differentiate, for example, between two travel agents using the same databases for prices, but having different delays for providing the answer (in an urgent situation, the faster one would be preferred). On the other hand, it is possible to define equivalence classes of agents offering similar services with respect to the waiting time (possibly up to an acceptable time difference).

The paper is organised as follows. We start in Section 2 with a brief presentation of TiMo, including its syntax and operational semantics, and introduce an example to illustrate the basic features of TiMo. In Sections 3 and 4, we formally define a number of timed bisimulations, and present some of their properties. Section 5 discusses the 'up-to' technique in the context of the bisimulations introduced in this paper. Conclusion and references end the paper.

2 TiMo

Timing constraints for migration allow one to specify what is the longest time it takes a mobile process to move to another location. A timer denoted by $\Delta 3$ associated to a migration action $\mathtt{go}^{\Delta 3} work$ indicates that the process moves to

location *work* after at most 3 time units. It is also possible to constrain the waiting for a communication on a channel; if a communication action does not happen before a deadline, the process gives up and switches its operation to an alternative. E.g., a timer $\Delta 5$ associated to an output action $a^{\Delta 5}!\langle 10\rangle$ makes the channel available for communication only for the period of 5 time units.

2.1 Syntax

We assume suitable data sets including a set *Loc* of locations and a set *Chan* of communication channels. We use a set *Id* of process identifiers, and each $id \in Id$ has the arity m_{id}. In what follows, we use $\boldsymbol{x}$ to denote a finite tuple of elements $(x_1, \ldots, x_k)$ whenever it does not lead to a confusion.

The syntax of TiMo [11] is given in Table 1, where P are processes, L located processes, and N networks. Moreover, for each $id \in Id$ there is a unique definition of the form:

$$id(u_1, \ldots, u_{m_{id}} : X_1^{id}, \ldots, X_{m_{id}}^{id}) = P_{id} \ , \tag{1}$$

where P_{id} is a process expression, the u_i's are distinct variables playing the role of parameters, and the X_i^{id}'s are data types. In Table 1, it is assumed that:

- $a \in Chan$ is a channel;
- $lt \in \mathbb{N} \cup \{\infty\}$ is a deadline, where lt stands for *local time*;
- each v_i in $\boldsymbol{v}$ is an expression built from data values and variables;
- each u_i in $\boldsymbol{u}$ is a variable, and each X_i in $\boldsymbol{X}$ is a data type;
- l is a location or a location variable; and
- Ⓢ is a special symbol used to state that a process is temporarily 'stalled' and will be re-activated after a time progress.

The only variable binding constructor is $a^{\Delta lt}?(\boldsymbol{u}{:}\boldsymbol{X})$ then P else P' which binds the variables $\boldsymbol{u}$ within P (but *not* within P'). We use $fv(P)$ to denote the free variables of a process P (and similarly for networks). For a process definition as in (1), we assume that $fv(P_{id}) \subseteq \{u_1, \ldots, u_{m_{id}}\}$ and so the free variables of P_{id} are parameter bound. Processes are defined up to alpha-conversion, and $\{v/u, \ldots\}P$ is obtained from P by replacing all free occurrences of a variable u by v, etc, possible after alpha-converting P in order to avoid clashes. Moreover, if $\boldsymbol{v}$ and $\boldsymbol{u}$ are tuples of the same length then $\{\boldsymbol{v}/\boldsymbol{u}\}P$ denotes $\{v_1/u_1, v_2/u_2, \ldots, v_k/u_k\}P$.

Intuitively, a process $a^{\Delta lt}!\langle \boldsymbol{v}\rangle$ then P else P' attempts to send a tuple of values $\boldsymbol{v}$ over channel a for lt time units. If successful, it continues as process P; otherwise it continues as process P'. Similarly, $a^{\Delta lt}?(\boldsymbol{u}{:}\boldsymbol{X})$ then P else P' is a process that attempts for lt time units to input a tuple of values of type $\boldsymbol{X}$ and substitute them for the variables $\boldsymbol{u}$. Mobility is implemented by a process $\mathsf{go}^{\Delta lt} l$ then P which moves from the current location to the location l within lt time units. Note that since l can be a variable, and so its value is assigned dynamically through the communication with other processes, migration actions support a flexible scheme for the movement of processes from one location to another. Processes are further constructed from the (terminated) process 0 and

Table 1. TiMo syntax

Processes	$P ::= a^{\Delta lt}!\langle \boldsymbol{v} \rangle$ then P else P' ∣	(output)
	$a^{\Delta lt}?(\boldsymbol{u}{:}\boldsymbol{X})$ then P else P' ∣	(input)
	$\mathsf{go}^{\Delta lt}\ l$ then P ∣	(move)
	$P \mid P'$ ∣	(parallel)
	0 ∣	(termination)
	$id(\boldsymbol{v})$	(recursion)
	ⓈP	(stalling)
Located processes	$L ::= l[\![P]\!]$	
Networks	$N ::= L \ ∣\ L \mid N$	

parallel composition $P|P'$. A located process $l[\![P]\!]$ specifies a process P running at location l, and networks are composed out of located processes. 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 Ⓢ in N;
- assuming that id is as in the recursive equation (1), for every $id(\boldsymbol{v})$ 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).

The set of processes is denoted by $\mathcal{P}$, the set of located processes by $\mathcal{L}$, and the set of networks by $\mathcal{N}$.

By delaying the migration to another location, we can model in a simple way the movement time of processes within the network which is, in general, outside the control of a system designer.

2.2 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 first three equalities in Table 2 hold. Its role is to rearrange a network in order to apply the action rules which are also given in Table 2. Using the first three equalities in Table 2, one can always transform a given network N into a finite parallel composition of located processes of the form

$$l_1[\![P_1]\!] \mid \ldots \mid l_n[\![P_n]\!]$$

Table 2. TiMo operational semantics

(NComm)	$N \mid N' \equiv N' \mid N$
(NAssoc)	$(N \mid N') \mid N'' \equiv N \mid (N' \mid N'')$
(NSplit)	$l[\![P \mid P']\!] \equiv l[\![P]\!] \mid l[\![P']\!]$
(Move)	$l[\![\mathsf{go}^{\Delta lt} l' \text{ then } P]\!] \xrightarrow{l'@l} l'[\![\circledS P]\!]$
(Com)	$\dfrac{v_1 \in X_1 \ \ldots \ v_k \in X_k}{l[\![a^{\Delta lt}!\langle \boldsymbol{v} \rangle \text{ then } P \text{ else } Q \mid a^{\Delta lt'}?(\boldsymbol{u}{:}\boldsymbol{X}) \text{ then } P' \text{ else } Q']\!] \xrightarrow{a\langle \boldsymbol{v} \rangle@l} l[\![\circledS P \mid \circledS\{\boldsymbol{v}/\boldsymbol{u}\}P']\!]}$
(Call)	$l[\![id(\boldsymbol{v})]\!] \xrightarrow{id@l} l[\![\circledS\{\boldsymbol{v}/\boldsymbol{u}\}P_{id}]\!]$
(Par)	$\dfrac{N \xrightarrow{\lambda} N'}{N \mid N'' \xrightarrow{\lambda} N' \mid N''}$
(Equiv)	$\dfrac{N \equiv N' \qquad N' \xrightarrow{\lambda} N'' \qquad N'' \equiv N'''}{N \xrightarrow{\lambda} N'''}$
(Time)	$\dfrac{N \not\rightarrow_l}{N \xrightarrow{\surd_l} \phi_l(N)}$

such that no process P_i has the parallel composition operator at its topmost level. Each located process $l_i[\![P_i]\!]$ is called a component of N, and the parallel composition 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 time actions.

Table 2 introduces two kinds of rules,

$$N \xrightarrow{\lambda} N' \text{ and } N \xrightarrow{\surd_l} N' \,.$$

The former is an execution of an action λ, and the latter a time step at location l. In the rule (Time), $N \not\rightarrow_l$ means that the rules (Call) and (Com) as well as (Move) with $\Delta lt = \Delta 0$ cannot be applied to N for location l. It can be noticed that in rule (Time) we use negative premises, i.e., an activity is performed in the absence of other actions. This is due to the fact that sequencing the evolution

in time units can only be defined using negative premises, as done for sequencing processes in [4,13]. Moreover, $\phi_l(N)$ is obtained by taking the component decomposition of N and simultaneously replacing all components:

$$l[\![a^{\Delta lt}\omega \text{ then } P \text{ else } Q]\!] \quad \text{by} \quad \begin{cases} l[\![Q]\!] & \text{if } lt = 0 \\ l[\![a^{\Delta lt-1}\omega \text{ then } P \text{ else } Q]\!] & \text{otherwise} \end{cases}$$

$$l[\![\mathsf{go}^{\Delta lt} l' \text{ then } P]\!] \quad \text{by} \quad l[\![\mathsf{go}^{\Delta lt-1} l' \text{ then } P]\!]$$

where ω stands for $!\langle \boldsymbol{v} \rangle$ or $?(\boldsymbol{u}{:}\boldsymbol{X})$. After that, all the occurrences of the symbol Ⓢ in N are erased since processes that were unfolded or interacted with other processes or migrated need to be activated (note that the number of the symbols Ⓢ to be erased cannot exceed the number of the components of the network).

The rules of Table 2 express executions of individual actions. A complete computational step is captured by a derivation of the form

$$N \stackrel{\Lambda@l}{\Longrightarrow} N' \ ,$$

where $\Lambda = \{\lambda_1, \ldots, \lambda_m\}$ $(m \geq 0)$ is a finite multiset of actions for some location l (i.e., actions λ_i of the form $l'@l$ or $a\langle v \rangle @l$ or $id@l$) such that

$$N \stackrel{\lambda_1}{\longrightarrow} N_1 \ldots N_{m-1} \stackrel{\lambda_m}{\longrightarrow} N_m \stackrel{\surd_l}{\longrightarrow} N' \ .$$

That means that 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 Λ at location l. If there is only a time progression at a location l, we write $N \stackrel{\emptyset@l}{\Longrightarrow} N'$.

In terms of executing TiMo specifications on an abstract machine, one can imagine the latter as a device transforming well-formed networks into well-formed networks. At any stage, the machine selects one location l as the active one. Then, it executes all interprocess communications within location l as well as all migrations with expired (zero) timers in a maximally concurrent way. This is followed by the execution of arbitrarily many migrations with unexpired timers at location l. Finally, one decrements all the top-most timers in all the network components at location l which have not yet been involved in the current computational step.

2.3 An Example

The *TravelShop* example discussed in [10] is rather involved, so in this paper we use its simplified version to illustrate the operational semantics of TiMo. In the *UrgentTravel* example a client process attempts to initiate an unspecified *travel* process as soon as it receives a flight offer.

The scenario involves three locations and three processes. The role of each location is as follows: *office* is a location where the *client* process starts its work, and $agency_i$ (for $i = 1, 2$) is a travel agency where the client can find out about the price of tickets. The role of each process is as follows:

- *client* resides in the *office* location, and is determined to pay for a flight as soon as it receives an offer from one of two travel agencies. After sending an email to each agency, it awaits for the quickest response to initiate the *travel* process.
- $agent_i$ (for $i = 1, 2$) resides in the $agency_i$ location, and replies to emails received from clients.

We use timers in order to impose deadlines on the execution of communications and migrations. Each location has its local clock which determines the timing of actions executed at that location. The process specifications that capture the essential features of the above scenario are:

$$agent_i = a^{\Delta 5}!\langle offer_i \rangle \text{ then } agent_i \text{ else } agent_i$$

$$\begin{aligned} client \;&= d^{\Delta 6}?(y) \text{ then } travel(y) \text{ else } 0 \\ &\mid \mathsf{go}^{\Delta 2} agency_1 \text{ then } (a^{\Delta 1}?(x) \text{ then } (\mathsf{go}^{\Delta 2} office \text{ then } d^{\Delta 1}!\langle x \rangle) \text{ else } 0) \\ &\mid \mathsf{go}^{\Delta 3} agency_2 \text{ then } (a^{\Delta 1}?(x) \text{ then } (\mathsf{go}^{\Delta 3} office \text{ then } d^{\Delta 1}!\langle x \rangle) \text{ else } 0) \end{aligned}$$

Note that in the above definitions we slightly simplified the notation and used:

- $d^{\Delta t}!\langle x \rangle$ instead of $d^{\Delta t}!\langle x \rangle$ then 0 else 0
- $d^{\Delta 6}?(y)$ instead of $d^{\Delta 6}?(y{:}1..1000)$
- $a^{\Delta 1}?(x)$ instead of $a^{\Delta 1}?(x{:}1..1000)$.

Table 3 shows a typical execution of the following network modelling our scenario:

$$UrgentTravel = office[\![client]\!] \mid agency_1[\![agent_1]\!] \mid agency_2[\![agent_2]\!]$$

3 Timed Bisimulations in TiMo

In what follows, we define various behavioural equivalences for networks of located processes. Similarly as in timed distributed π-calculus [7], we start by extending the standard notion of strong bisimilarity to take into account timed transitions.

Definition 1 (strong timed bisimulation)
Let $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{N}$ be a binary relation on networks of processes.

1. *$\mathcal{R}$ is a* strong timed simulation *(ST simulation) if*

   $$(N_1, N_2) \in \mathcal{R} \;\wedge\; N_1 \xrightarrow{\psi} N_1' \;\; \textit{implies} \;\; \exists N_2' \in \mathcal{N} :\; N_2 \xrightarrow{\psi} N_2' \;\wedge\; (N_1', N_2') \in \mathcal{R}\,.$$

   where ψ is any action allowed by the operational semantics.
2. *$\mathcal{R}$ is a* strong timed bisimulation *(ST bisimulation) if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong timed simulations.*
3. *The* strong timed bisimilarity $\sim$ *is the union of all ST bisimulations.*

Table 3. Applying operational semantics

UrgentTravel

$\xrightarrow{\emptyset@office}\ \xrightarrow{\emptyset@office}\ \xrightarrow{\emptyset@office}\ \xrightarrow{\emptyset@agency_1}\ \xrightarrow{\emptyset@agency_1}\ \xrightarrow{\emptyset@agency_2}$

$$
\begin{array}{l}
\quad office[\![d^{\Delta 4}?(y)\ \mathsf{then}\ travel(y)\ \mathsf{else}\ 0 \\
\qquad\quad |\ \mathsf{go}^{\Delta 0} agency_1\ \mathsf{then}\ (a^{\Delta 1}?(x)\ \mathsf{then}\ (\mathsf{go}^{\Delta 2} office\ \mathsf{then}\ d^{\Delta 1}!\langle x\rangle)\ \mathsf{else}\ 0) \\
\qquad\quad |\ \mathsf{go}^{\Delta 1} agency_2\ \mathsf{then}\ (a^{\Delta 1}?(x)\ \mathsf{then}\ (\mathsf{go}^{\Delta 3} office\ \mathsf{then}\ d^{\Delta 1}!\langle x\rangle)\ \mathsf{else}\ 0)[\![\\
|\ \ agency_1[\![a^{\Delta 4}!\langle offer_1\rangle\ \mathsf{then}\ agent_1\ \mathsf{else}\ agent_1]\!] \\
|\ \ agency_2[\![a^{\Delta 5}!\langle offer_2\rangle\ \mathsf{then}\ agent_2\ \mathsf{else}\ agent_2]\!]
\end{array}
$$

$\xrightarrow{\{agency_1@office,\ agency_2@office\}@office}$

$\xrightarrow{\{a\langle offer_1\rangle@agency_1\}@agency_1}\ \xrightarrow{\{a\langle offer_2\rangle@agency_2\}@agency_2}$

$$
\begin{array}{l}
\quad office[\![d^{\Delta 3}?(y)\ \mathsf{then}\ travel(y)\ \mathsf{else}\ 0]\!] \\
|\ \ agency_1[\![agent_1 \mid \mathsf{go}^{\Delta 2} office\ \mathsf{then}\ d^{\Delta 1}!\langle offer_1\rangle]\!] \\
|\ \ agency_2[\![agent_2 \mid \mathsf{go}^{\Delta 3} office\ \mathsf{then}\ d^{\Delta 1}!\langle offer_2\rangle]\!]
\end{array}
$$

$\xrightarrow{\{office@agency_1\}@agency_1}$

$$
\begin{array}{l}
\quad office[\![d^{\Delta 3}?(y)\ \mathsf{then}\ travel(y)\ \mathsf{else}\ 0 \mid d^{\Delta 1}!\langle offer_1\rangle]\!] \\
|\ \ agency_1[\![a^{\Delta 5}!\langle offer_1\rangle\ \mathsf{then}\ agent_1\ \mathsf{else}\ agent_1]\!] \\
|\ \ agency_2[\![agent_2 \mid \mathsf{go}^{\Delta 3} office\ \mathsf{then}\ d^{\Delta 1}!\langle offer_2\rangle]\!]
\end{array}
$$

$\xrightarrow{\{d\langle offer_1\rangle@office\}@office}$

$$
\begin{array}{l}
\quad office[\![travel(offer_1) \mid 0]\!] \\
|\ \ agency_1[\![a^{\Delta 5}!\langle offer_1\rangle\ \mathsf{then}\ agent_1\ \mathsf{else}\ agent_1]\!] \\
|\ \ agency_2[\![agent_2 \mid \mathsf{go}^{\Delta 3} office\ \mathsf{then}\ d^{\Delta 1}!\langle offer_2\rangle]\!]
\end{array}
$$

Essentially, the above definition treats timed transitions just as any other transitions, and therefore coincides with the original notion of bisimilarity for labelled transition systems. It is easy to check that $\sim$ is an equivalence relation, and also the largest strong timed bisimulation. From the point of view of the behaviour of TiMo networks, a crucial result is that strong timed bisimularity can be used to compare their evolutions in terms of complete computational steps of well-formed networks.

Theorem 1. *Let N_1 and N_2 be two well-formed networks. Then:*

$$N_1 \sim N_2 \ \wedge \ N_1 \stackrel{\Lambda@l}{\Longrightarrow} N_1' \quad \textit{implies} \quad \exists N_2' \in \mathcal{N} : \ N_2 \stackrel{\Lambda@l}{\Longrightarrow} N_2' \ \wedge \ N_1' \sim N_2' \ .$$

Together with the fact that, for every well-formed network N, $N \stackrel{\Lambda@l}{\Longrightarrow} N'$ implies that N' is also well-formed (see [11]), this means that the strong timed bisimilarity is an adequate tool for comparing the behaviour of (well-formed) networks.

The above definition of equivalence compares the evolution of whole networks, but does not provide means for reasoning about equivalence of compositionally defined networks. Consider, for example, two networks:

$$N_1 = l[\![a^{\Delta lt}!\langle 1\rangle \ \mathsf{then}\ 0\ \mathsf{else}\ 0]\!] \ \text{ and } \ N_2 = l[\![0]\!] \ .$$

Clearly, $N_1 \sim N_2$ as both networks allow only the transition $\stackrel{\emptyset@l}{\Longrightarrow}$. However, when we compose them with $N = l[\![a^{\Delta lt}?(u : \mathbb{N})\ \mathsf{then}\ 0\ \mathsf{else}\ 0]\!]$ then:

$$N_1 \mid N \not\sim N_2 \mid N$$

as the first composition can execute transition $\stackrel{\{a\langle 1\rangle@l\}@l}{\Longrightarrow}$ whereas the second one can only execute $\stackrel{\emptyset@l}{\Longrightarrow}$.

To be able to reason about networks in a compositional way, one may augment (only for the purpose of dealing with equivalences) the operational semantics of processes with two additional rules relating to communication, which intuitively represent individual evolutions of interacting processes:

$$\text{(Snd)} \qquad l[\![a^{\Delta lt}!\langle \boldsymbol{v}\rangle \ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] \stackrel{a!\langle \boldsymbol{v}\rangle@l}{\longrightarrow} l[\![\circledS P]\!]$$

$$\text{(Rcv)} \quad \frac{v_1 \in X_1 \ \ldots \ v_k \in X_k}{a^{\Delta lt}?(\boldsymbol{u}{:}\boldsymbol{X})\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!] \stackrel{a?\langle \boldsymbol{v}\rangle@l}{\longrightarrow} l[\![\{\boldsymbol{v}/\boldsymbol{u}\}P]\!]}$$

All the previous rules remain unchanged. In particular, $N \not\rightarrow_l$ in the rule (Time) still means that the rules (Call) and (Com) as well as (Move) with $\Delta lt = \Delta 0$ cannot be applied to N for location l; in other words the two new rules, (Snd) and (Rcv), are not taken into account. The transitions of the extended operational semantics will be denoted by $\stackrel{\psi}{\rightarrow_e}$ rather than $\stackrel{\psi}{\rightarrow}$.

Definition 2 (strong extended timed bisimulation)

Let $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{N}$ be a binary relation on networks of processes.

1. *$\mathcal{R}$ is a* strong extended timed simulation *(SET simulation) if*

   $$(N_1, N_2) \in \mathcal{R} \wedge N_1 \stackrel{\psi}{\rightarrow_e} N_1' \ \textit{implies} \ \exists N_2' \in \mathcal{N} : N_2 \stackrel{\psi}{\rightarrow_e} N_2' \wedge (N_1', N_2') \in \mathcal{R} \ .$$

   where ψ is any action allowed by the extended operational semantics.

2. $\mathcal{R}$ *is a* strong extended timed bisimulation *(SET bisimulation) if both* $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are strong extended timed simulations.*
3. *The* strong extended timed bisimilarity $\sim^e$ *is the union of all SET bisimulations.*

The strong extended timed bisimilarity is compositional, and it implies strong timed bisimilarity.

Theorem 2. *Let* N_1, N_1', N_2 *and* N_2' *be well formed networks. Then:*

$$N_1 \sim^e N_2 \text{ and } N_1' \sim^e N_2' \text{ implies } N_1 \mid N_1' \sim^e N_2 \mid N_2' .$$

Proposition 1. *Let* N *and* N' *be well formed networks. Then:*

$$N \sim^e N' \text{ implies } N \sim N' .$$

It therefore follows, in the context of Theorem 1, that strong extended timed bisimilarity provides an adequate tool for comparing behaviours of compositionally defined networks considered up to certain time deadline.

4 Bounded Timed Bisimulations in TiMo

The above notion of equivalence takes into account the timed behaviour requiring an exact match of transitions of two networks, for their entire evolution. Sometimes these requirements are too strong. According to [18] where a similar approach is presented, real-time distributed systems usually require a certain behaviour within a given threshold of time units. That is why we will now restrict equivalences up-to some threshold time values specified individually for each location $l \in Loc$, defining *bounded* timed equivalences.

In what follows we assume that $Loc = \{l_1, \ldots, l_n\}$. We then introduce some additional notations and terminology:

- $\mathbb{T} = \{(t_1@l_1, \ldots, t_n@l_n) \mid t_1, \ldots t_n \in \mathbb{N}\}$ comprises tuples in which each location l_i has an associated number of time units in which it will be observed. We use $\widehat{t}$ to denote $(t_1@l_1, \ldots, t_n@l_n)$, and $\widehat{t}_{l_i}$ to denote t_i.
- For every $\widehat{t} = (t_1@l_1, \ldots, t_n@l_n) \in \mathbb{T}$ and $l_i \in Loc$,

$$\widehat{t} \ominus l_i = (t_1@l_1, \ldots, t_{i-1}@l_{i-1}, t_i - 1@l_i, t_{i+1}@l_{i+1}, \ldots, t_n@l_n) .$$

  Intuitively, $\widehat{t} \ominus l_i$ records that one time unit has passed at location l_i, and the remaining observation time has been updated accordingly.
- Any relation $\mathcal{R} \subseteq \mathcal{N} \times \mathbb{T} \times \mathcal{N}$ is a *timed relation* over networks.
- The *inverse of a timed relation* $\mathcal{R}$ is

$$\mathcal{R}^{-1} = \{(N', \widehat{t}, N) \mid (N, \widehat{t}, N') \in \mathcal{R}\} .$$

- If $\mathcal{R}$ is a timed relation and $\widehat{t} \in \mathbb{T}$ then the $\widehat{t}$*-projection* of $\mathcal{R}$ is:

$$\mathcal{R}_{\widehat{t}} = \{(N_1, N_2) \mid (N_1, \widehat{t}, N_2) \in \mathcal{R}\} .$$

Definition 3 (strong bounded timed bisimulation)

Let $\mathcal{R} \subseteq \mathcal{N} \times \mathbb{T} \times \mathcal{N}$ be a timed relation over $\mathcal{N}$.

1. *$\mathcal{R}$ is a* strong bounded timed simulation *(SBT simulation) if*

$$\left\{ \begin{array}{c} (N_1, \widehat{t}, N_2) \in \mathcal{R} \\ N_1 \xrightarrow{\surd_l} N_1' \text{ and } \widehat{t}_l > 0 \end{array} \right\} \quad \text{implies} \quad \exists N_2' \in \mathcal{N} : \left\{ \begin{array}{c} N_2 \xrightarrow{\surd_l} N_2' \\ (N_1', \widehat{t} \ominus l, N_2') \in \mathcal{R} \end{array} \right\}$$

   and, for each λ of the form $l'@l$ or $a\langle \boldsymbol{v} \rangle @l$ or $id@l$,

$$\left\{ \begin{array}{c} (N_1, \widehat{t}, N_2) \in \mathcal{R} \\ N_1 \xrightarrow{\lambda} N_1' \text{ and } \widehat{t}_l > 0 \end{array} \right\} \quad \text{implies} \quad \exists N_2' \in \mathcal{N} : \left\{ \begin{array}{c} N_2 \xrightarrow{\lambda} N_2' \\ (N_1', \widehat{t}, N_2') \in \mathcal{R} \end{array} \right\}$$

2. *$\mathcal{R}$ is a* strong bounded timed bisimulation *(SBT bisimulation) if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong bounded timed simulations.*
3. *The* strong bounded timed bisimilarity $\simeq$ *is the union of all SBT bisimulations.*

One can see that $\simeq$ is the largest SBT bisimulation. Moreover, SBT bisimulations enjoy properties similar to those satisfied by equivalence relations.

Proposition 2. *The inverse, composition and union of SBT bisimulations are SBT bisimulations, where the composition of timed relations $\mathcal{R}$ and $\mathcal{R}'$ comprises all triples $(N, \widehat{t}, N'')$ for which there is $N' \in \mathcal{N}$ satisfying $(N, \widehat{t}, N') \in \mathcal{R}$ and $(N', t, N'') \in \mathcal{R}'$.*

Strong bounded timed bisimilarity is such that being equivalent up-to a certain time bound implies equivalence up-to any smaller time bound.

Proposition 3. *Let $N \simeq_{\widehat{t}} N'$ be two well-formed networks. Then $N \simeq_{\widehat{t'}} N'$, for every $\widehat{t'} \in \mathbb{T}$ satisfying $t_1 \leq t_1', \ldots, t_n \leq t_n'$.*

Finally, we have a crucial result that strong bounded timed bisimularity can be used to compare the complete computational steps of two networks.

Theorem 3. *Let N_1 and N_2 be two well-formed networks. Then:*

$$\left\{ \begin{array}{c} N_1 \simeq_{\widehat{t}} N_2 \\ N_1 \xRightarrow{\Lambda @ l} N_1' \text{ and } \widehat{t}_l > 0 \end{array} \right\} \quad \text{implies} \quad \exists N_2' \in \mathcal{N} : \left\{ \begin{array}{c} N_2 \xRightarrow{\Lambda @ l} N_2' \\ N_1' \simeq_{\widehat{t} \ominus l} N_2' \end{array} \right\}$$

Similarly as strong time bisimilarity is not preserved by network composition, strong bounded time bisimilarity is not preserved by network composition. However, as in the previous case, one can consider two additional rules, (Snd) and (Rcv), and obtain the extended version of $\simeq_{\widehat{t}}$.

Definition 4 (strong extended bounded timed bisimulation).

Let $\mathcal{R} \subseteq \mathcal{N} \times \mathbb{T} \times \mathcal{N}$ be a timed relation over $\mathcal{N}$.

1. $\mathcal{R}$ *is a* strong extended bounded timed simulation *(SEBT simulation) if*

$$\left\{\begin{array}{c}(N_1,\widehat{t},N_2)\in\mathcal{R}\\ N_1\xrightarrow{\surd_l}_e N_1' \text{ and } \widehat{t}_l>0\end{array}\right\} \text{ implies } \exists N_2'\in\mathcal{N}:\left\{\begin{array}{c}N_2\xrightarrow{\surd_l}_e N_2'\\ (N_1',\widehat{t}\ominus l,N_2')\in\mathcal{R}\end{array}\right\}$$

and, for each ψ *of the form* $l'@l$ *or* $a\langle\boldsymbol{v}\rangle@l$ *or* $a!\langle\boldsymbol{v}\rangle@l$ *or* $a?\langle\boldsymbol{v}\rangle@l$ *or* $id@l$,

$$\left\{\begin{array}{c}(N_1,\widehat{t},N_2)\in\mathcal{R}\\ N_1\xrightarrow{\psi}_e N_1' \text{ and } \widehat{t}_l>0\end{array}\right\} \text{ implies } \exists N_2'\in\mathcal{N}:\left\{\begin{array}{c}N_2\xrightarrow{\psi}_e N_2'\\ (N_1',\widehat{t},N_2')\in\mathcal{R}\end{array}\right\}$$

2. $\mathcal{R}$ *is a* strong extended bounded timed bisimulation *(SEBT bisimulation) if both* $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are strong extended bounded timed simulations.*
3. *The* strong extended bounded timed bisimilarity *is the union* $\simeq^e$ *of all SEBT bisimulations.*

The strong extended bounded timed bisimilarity is compositional, and it implies strong bounded timed bisimilarity.

Theorem 4. *Let* N_1, N_1', N_2 *and* N_2' *be well formed networks and* $\widehat{t}\in\mathbb{T}$. *Then:*

$$N_1\simeq^e_{\widehat{t}} N_2 \text{ and } N_1'\simeq^e_{\widehat{t}} N_2' \text{ implies } N_1\mid N_1'\simeq^e_{\widehat{t}} N_2\mid N_2'\ .$$

Proposition 4. *Let* N_1 *and* N_2 *be well formed networks and* $\widehat{t}\in\mathbb{T}$. *Then:*

$$N_1\simeq^e_{\widehat{t}} N_2 \text{ implies } N_1\simeq_{\widehat{t}} N_2\ .$$

It therefore follows, in the context of Theorem 3, that strong extended timed bisimilarity provides an adequate tool for comparing behaviours of compositionally defined networks.

5 Relaxing Timed Bisimulations

In what follows we use the 'up-to' technique presented in [19] in the context of bounded timed bisimulations. The standard proof technique to establish that N_1 and N_2 are bisimilar is to find a bisimulation $\mathcal{R}$ s.t.such that $(N_1,N_2)\in\mathcal{R}$ and $\mathcal{R}$ is closed under transitions of the operational semantics; in particular, that the derivatives (N_1',N_2') of (N_1,N_2) are also in $\mathcal{R}$. Sometimes it is difficult to find directly such a relation $\mathcal{R}$. Instead, there is an useful alternative technique, the so-called bisimulation 'up-to' some relation $\mathcal{R}'$: for a relation $\mathcal{R}$, which is not a bisimulation, if $(N_1,N_2)\in\mathcal{R}$, then one requires that the derivatives (N_1',N_2') are in $\mathcal{R}'$. Under certain conditions one can then establish that N_1 and N_2 are bisimilar. For such a technique, a general framework working for untimed operational semantics was presented in [20]. We cannot make a direct use of that framework, but we can adapt it in a straightforward manner to our setting.

We begin by introducing a notion of 'progressing' a timed relation towards another timed relation.

Definition 5 (strong progress)

Let $\mathcal{R}$ and $\mathcal{R}'$ be two timed relations. Then $\mathcal{R}$ strongly progresses *to $\mathcal{R}'$ if*

$$\left\{ \begin{array}{c} (N_1, \widehat{t}, N_2) \in \mathcal{R} \\ N_1 \xrightarrow{\surd_l} N_1' \ \wedge \ \widehat{t}_l > 0 \end{array} \right\} \implies \exists N_2' \in \mathcal{N} : \left\{ \begin{array}{c} N_2 \xrightarrow{\surd_l} N_2' \\ (N_1', \widehat{t} \ominus l, N_2') \in \mathcal{R}' \end{array} \right\}$$

and, for each λ of the form $l'@l$ or $a\langle \boldsymbol{v} \rangle @l$ or $id@l$,

$$\left\{ \begin{array}{c} (N_1, \widehat{t}, N_2) \in \mathcal{R} \\ N_1 \xrightarrow{\lambda} N_1' \ \wedge \ \widehat{t}_l > 0 \end{array} \right\} \implies \exists N_2' \in \mathcal{N} : \left\{ \begin{array}{c} N_2 \xrightarrow{\lambda} N_2' \\ (N_1', \widehat{t}, N_2') \in \mathcal{R}' \end{array} \right\}$$

We denote this by $\mathcal{R} \rightsquigarrow \mathcal{R}'$.

The above definition is similar to that of SBT bisimulation, except that the derivatives $(N_1', \widehat{t}, N_2')$ and $(N_1', \widehat{t} \ominus l, N_2')$ must be in $\mathcal{R}'$ rather than $\mathcal{R}$.

Proposition 5. *If $\mathcal{R} \rightsquigarrow \mathcal{R}'$ and $\mathcal{R}'$ is an SBT bisimulation, then $\mathcal{R}$ is also an SBT bisimulation.*

Therefore, to establish that $N_1 \simeq_{\widehat{t}} N_2$ it is enough to find a relation $\mathcal{R}$ with $(N_1, \widehat{t}, N_2) \in \mathcal{R}$ which strongly progresses to a known SBT bisimulation $\mathcal{R}'$. The choice of $\mathcal{R}'$ depends on the particular equivalence we are trying to establish. One of the most common cases is when $\mathcal{R}' = \simeq$. However, in general we may not have a relation $\mathcal{R}'$ known to be a bisimulation. Nevertheless, we may find that $\mathcal{R}$ progresses to a relation $\mathcal{R}' = \mathcal{F}(\mathcal{R})$ for some mapping $\mathcal{F}$ over relations. The idea is that if $\mathcal{R}$ progresses to $\mathcal{F}(\mathcal{R})$ and $\mathcal{F}$ satisfies certain conditions, then $\mathcal{R}$ is included in $\simeq$. Thus, to establish $N_1 \simeq_{\widehat{t}} N_2$ we need to find such an $\mathcal{F}$ whenever $\mathcal{R}$ contains $(N_1, \widehat{t}, N_2)$.

Suitable mappings $\mathcal{F}$ are characterised in [20] as being *strongly safe* which, in our context, means that for any timed relations $\mathcal{R}$ and $\mathcal{R}'$, if $\mathcal{R} \subseteq \mathcal{R}'$ and $\mathcal{R} \rightsquigarrow \mathcal{R}'$, then $\mathcal{F}(\mathcal{R}) \subseteq \mathcal{F}(\mathcal{R}')$ and $\mathcal{F}(\mathcal{R}) \rightsquigarrow \mathcal{F}(\mathcal{R}')$. More details about the 'up-to' techniques and safe functions can be found in [20].

6 Conclusion

This paper presents an approach in which local clocks operate independently, and so processes at different locations evolve asynchronously. On the other hand, processes operating at the same location evolve synchronously, and temporal evolution of mobile processes is governed by independent local clocks and their migration timeouts. A computational step captures the cumulative effect of the concurrent execution of a group of actions executed at one location.

In process calculi such as distributed π-calculus, timed distributed π-calculus and other formalisms with explicit migration operators, bisimulations are used to compare behaviours of mobile processes evolving in distributed systems with

explicit locations. Bisimulations are behavioural equivalences used to study the properties of a concurrent system by verifying its bisimilarity with a system known to enjoy those properties. Moreover, given the model of a system, bisimulations can be used to consider equivalent simplified models.

In this paper, we defined behavioural equivalences between migrating process in distributed systems in terms of local time and locations. In particular, the strong timed bisimilarity can be used to compare the complete computational steps of two networks. Moreover, two networks that are strong bounded timed bisimilar up to certain deadlines $\widehat{t}$ remain equivalent provided that their execution is restricted to the time limit given by $\widehat{t}$. We also defined extended versions of both equivalences which can support compositional reasoning.

Acknowledgement. The work of Bogdan Aman and Gabriel Ciobanu was supported by a grant from the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0919. In addition, the work of Bogdan Aman was supported by POSDRU/89/1.5/S/49944.

References

1. Aman, B., Ciobanu, G.: Timed Mobile Ambients for Network Protocols. In: Suzuki, K., Higashino, T., Yasumoto, K., El-Fakih, K. (eds.) FORTE 2008. LNCS, vol. 5048, pp. 234–250. Springer, Heidelberg (2008)
2. Baeten, J.C.M., Bergstra, J.A.: Discrete time process algebra: Absolute time, relative time and parametric time. Fundam. Inform. 29(1-2), 51–76 (1997)
3. Berger, M.: Towards Abstractions for Distributed Systems. Ph.D. thesis, Department of Computing, Imperial College (2002)
4. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can't be traced. In: POPL, pp. 229–239 (1988)
5. Cardelli, L., Gordon, A.D.: Mobile Ambients. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, pp. 140–155. Springer, Heidelberg (1998)
6. Chen, L.: Timed Processes: Models, Axioms and Decidability. Ph.D. thesis, School of Informatics, University of Edinburgh (1993)
7. Ciobanu, G.: Behaviour Equivalences in Timed Distributed π-Calculus. In: Wirsing, M., Banâtre, J.-P., Hölzl, M., Rauschmayer, A. (eds.) Soft-Ware Intensive Systems. LNCS, vol. 5380, pp. 190–208. Springer, Heidelberg (2008)
8. Ciobanu, G., Juravle, C.: Flexible software architecture and language for mobile agents. Concurrency and Computation: Practice and Experience 24(6), 559–571 (2012)
9. Ciobanu, G., Koutny, M.: Modelling and Verification of Timed Interaction and Migration. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 215–229. Springer, Heidelberg (2008)
10. Ciobanu, G., Koutny, M.: Timed Migration and Interaction with Access Permissions. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 293–307. Springer, Heidelberg (2011)
11. Ciobanu, G., Koutny, M.: Timed mobility in process algebra and petri nets. J. Log. Algebr. Program. 80(7), 377–391 (2011)
12. Ciobanu, G., Prisacariu, C.: Timers for distributed systems. Electr. Notes Theor. Comput. Sci. 164(3), 81–99 (2006)

13. Groote, J.F.: Transition system specifications with negative premises. Theoretical Computer Science 118, 263–299 (1993)
14. Hennessy, M.: A distributed π-calculus. Cambridge University Press (2007)
15. Hennessy, M., Regan, T.: A process algebra for timed systems. Inf. Comput. 117(2), 221–239 (1995)
16. Milner, R.: Communicating and mobile systems - the π-calculus. Cambridge University Press (1999)
17. Nicollin, X., Sifakis, J.: The algebra of timed processes, atp: Theory and application. Inf. Comput. 114(1), 131–178 (1994)
18. Posse, E., Dingel, J.: Theory and Implementation of a Real-Time Extension to the π-Calculus. In: Hatcliff, J., Zucca, E. (eds.) FMOODS/FORTE 2010. LNCS, vol. 6117, pp. 125–139. Springer, Heidelberg (2010)
19. Sangiorgi, D.: A theory of bisimulation for the π-calculus. Acta Inf. 33(1), 69–97 (1996)
20. Sangiorgi, D., Walker, D.: The π-Calculus - a theory of mobile processes. Cambridge University Press (2001)
21. Yi, W.: A Calculus of Real-Time Systems. Ph.D. thesis, Department of Computer Science, Chalmers University of Technology (1991)