Verif cation of Real-Time Bounded Distributed Systems With Mobility

Bogdan Aman and Gabriel Ciobanu
Romanian Academy, Institute of Computer Science, Iaşi, Romania
bogdan.aman@gmail.com and gabriel@info.uaic.ro

We introduce and study a prototyping language for describing real-time distributed systems. Its time constraints are expressed as bounded intervals to model the uncertainty of the delay in migration and communication of agents placed in the locations of a distributed system. We provide the operational semantics, and illustrate the new language by a detailed example for which we use software tools for analyzing its temporal properties.

1. INTRODUCTION

Computer systems today are interconnected into large distributed systems. Distributed and concurrent systems are now used in both academic and industrial computing, forcing researchers and practitioners to look for theoretical models and software tools ref ecting the new framework based on mobility and interaction. Programming paradigms have progressed, allowing programmers to implement software in terms of high level abstractions. In distributed systems, such implementations are given by taking care of time scheduling, access to resources, and interaction among processes. When solving problems in distributed systems, it is useful to have an explicit notion of location, explicit migration, local interaction/communication and resource management.

Aiming to bridge the gap between the existing theoretical approach of process calculi and forthcoming realistic programming languages for distributed systems, we have introduced and studied a rather simple and expressive formalism called TiMo as a simplif ed version of timed distributed pi-calculus Ciobanu and Prisacariu (2006). In several aspects, TiMo is a prototyping language for multi-agent systems, featuring mobility and local interaction. The mobility refers to the fact that agents are in locations and that they can migrate from one location to the another, while agents must be in the same location in order to communicate. In this paper we present a revised/improved version of a real-time variant called rTiMo (Real Timed Mobility) in which the timing constraints are expressed as intervals in order to model the uncertainty of the delay in migration and communication. rTiMo supports explicit migration and local communication, together with certain timing constraints over these actions. We provide a relationship between rTiMo and the model checker Uppaal, and so making possible formal verif cation for rTiMo. For the complex distributed systems described by such a language, we show how it is possible to use Uppaal capabilities in order to verify certain properties.

The paper is organized as follows: Section 2 presents the syntax and semantics of rTiMo using interval constraints. Section 3 provides an example, while Sections 5 and 6 contain the modelling and verif cation in Uppaal. Finally, Section 7 presents the related work and concludes the paper.

2. SYNTAX AND SEMANTICS OF RTIMO

The prototyping language rTiMo provides suff cient expressiveness to model in an elegant way the migration and communication in real-time distributed systems. In this paper we present a revised/improved version of rTiMo involving global timing constraints in which the timing constraints are expressed as intervals in order to model the uncertainty of the delay in migration and communication. This realistic approach is used to provide systems a larger degree of non-determinism, for instance in deciding when a process is allowed to move from one location to another one. We achieve this by assuming that a rTiMo migration process can move to another location during a time interval (and not necessarily after exactly a given number of

Processes			
P, Q	$::=$	$a^{[t_1,t_2]}!\langle v\rangle$ then P else Q ı	(output)
		$a^{[t_1,t_2]}?(u)$ then P else Q ı	(input)
		go$^{[t_1,t_2]}l$ then P ı	(move)
		0 ı	(termination)
		$id(v)$ ı	(recursion)
		$P \mid Q$	(parallel)
Located Processes			
L	$::=$	$l[[P]]$	
Systems			
N	$::=$	L ı $L \mid N$	

Table 1: rTIMO *Syntax*

time units). Two processes may communicate only if they are present at the same location, they use the same channel and the time constraints allow them to interact.

In rTIMO, the transitions caused by performing actions with timeouts (migration and communication) are alternated with continuous transitions (time-passing). The semantics of rTIMO is provided by multiset labelled transitions in which multisets of actions are executed in parallel (in one step).

2.1. Syntax of rTIMO

Timing constraints expressed as intervals allow to model the uncertainty of the delay in migration and communication. The syntax of rTIMO is given in Table 1, where the following is assumed:

- a set *Loc* of locations l, a set *Chan* of communication channels a, and a set *Id* of process identif ers (each $id \in Id$ has its arity m_{id});
- for each $id \in Id$ there is a unique process def nition $id(u_1, \ldots, u_{m_{id}}) \overset{def}{=} P_{id}$, where the distinct variables u_i are parameters;
- $[t_1, t_2]$, where $t_1, t_2 \in \mathbb{R}_+$ and $t_1 \leq t_2$ is an execution time *interval* of an action;
- u is a tuple of variables;
- v is a tuple of expressions built from values, variables and allowed operations.
- a time interval $^{[t_1,t_2]}$ associated with a migration action such as go$^{[t_1,t_2]}loc$ then P indicates that process P can move to location loc after t time units, where $t \in [t_1, t_2]$.
- a time interval $^{[t_1,t_2]}$ associated with an output communication process $a^{[t_1,t_2]}!\langle z\rangle$ then P else Q makes the channel a available for communication (by sending z) for a period of $t_2 - t_1$ time units, but only after an idling of t_1 time units. It is also possible to impose an interval for an input communication process $a^{[t_1,t_2]}?(x)$ then P else Q along a channel a. In both cases, if the interaction does not happen in the interval $[t_1, t_2]$, the process gives up and continues as the alternative process Q.

The only variable binding constructor is $a^{[t_1,t_2]}?(u)$ then P else Q; it binds the variable u within P, but *not* within Q. The free variables of a process P are denoted by $fv(P)$; for a process def nition, it is assumed that $fv(P_{id}) \subseteq \{u_1, \ldots, u_{m_{id}}\}$, where u_i are the process parameters. Processes are def ned up-to an alpha-conversion, and $\{v/u, \ldots\}P$ denotes P in which all free occurrences of the variable u are replaced by v, eventually after alpha-converting P in order to avoid clashes.

Since location l, provided by a process go$^{[t_1,t_2]}l$ then P, can be a variable, its value can be assigned dynamically through communication with other processes; this means that migration supports a f exible scheme for the movement of processes from one location to another. Thus, the behaviour can be adapted to various changes of the distributed environment. Processes are further constructed from the (terminated) process 0, and parallel composition $P \mid Q$. A located process $l[[P]]$ specif es a process P running at location l, and a system N is built from its components L. A system N is well-formed if there are no free variables in N.

Remark 1 *In order to focus on the mobility and local interaction aspects of* rTIMO*, we abstract from arithmetical operations, considering by default that the simple ones (comparing, addition, subtraction) are included in the language.*

2.2. Operational Semantics of rTIMO

The f rst component of the operational semantics of rTIMO is the structural equivalence $\equiv$ over systems. The structural equivalence is the smallest congruence such that the equalities of Table 2 hold.

~~(NNULL)~~	~~$N \mid l[[0]] \equiv N$~~
(NCOMM)	$N \mid N' \equiv N' \mid N$
(NASSOC)	$(N \mid N') \mid N'' \equiv N \mid (N' \mid N'')$
(NSPLIT)	$l[[P \mid Q]] \equiv l[[P]] \mid l[[Q]]$

Table 2: rTiMo *Structural Congruence*

Essentially, the role of $\equiv$ is to rearrange a system in order to apply the rules of the operational semantics given in Table 3. Using the equalities of Table 2, a given system N can always be transformed into a f nite parallel composition of located processes of the form $l_1[[P_1]] \mid \dots \mid l_n[[P_n]]$ 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 whole expression $l_1[[P_1]] \mid \dots \mid l_n[[P_n]]$ is called a *component decomposition* of the system N.

The operational semantics rules of rTiMo are presented in Table 3. The multiset labelled transitions of form $N \xrightarrow{\Lambda} N'$ use a multiset Λ to indicate the actions executed in parallel in one step. When the multiset Λ contains only one action λ, in order to simplify the notation, $N \xrightarrow{\{\lambda\}} N'$ is simply written as $N \xrightarrow{\lambda} N'$. The transitions of form $N \overset{t}{\rightsquigarrow} N'$ represent a time step of length $t \in \mathbb{R}_+$.

In rule (Move0), the process $\mathsf{go}^{[0,t]}l'$ then P migrates from location l to location l' (illustrated by the label $l \triangleright l'$ of the transition) and then evolves as process P. In rule (Com), an output process $a^{[0,t]}!\langle v\rangle$ then P else Q located at location l, succeeds in sending a tuple of values v over channel a to process $a^{[0,t']}?(u)$ then P' else Q' also located at l. Both processes continue to execute at location l, the f rst one as P and the second one as $\{v/u\}P'$. The label $\{v/u\}@l$ of the rule (Com) illustrates the fact that a communication that lead to the replacement of u by v (denoted by $\{v/u\}$) took place at location l (denoted by $@l$). If a communication action has a timer equal to $[0, 0]$, then by using the rule (Put0) for output action or the rule (Get0) for input action, the generic process $a^{[0,0]} *$ then P else Q where $* \in \{!\langle v\rangle, ?(x)\}$ continues as the process Q. Rule (Call) describes the evolution of a recursion process. The rules (Equiv) and (DEquiv) are used to rearrange a system in order to apply a rule. Rule (Par) is used to compose larger systems from smaller ones by putting them in parallel, and considering the union of multisets of actions. The rules devoted to the passing of time are starting with letter D.

A computational step is captured by a derivation of the form:

$$N \xrightarrow{\Lambda} N_1 \overset{t}{\rightsquigarrow} N'.$$

This means that a step is a parallel execution of individual actions of Λ followed by a time step. Performing a step $N \xrightarrow{\Lambda} N_1 \overset{t}{\rightsquigarrow} N'$ means that N' is directly reachable from N. If there is no applicable action ($\Lambda = \emptyset$), $N \xrightarrow{\Lambda} N_1 \overset{t}{\rightsquigarrow} N'$ is written $N \overset{t}{\rightsquigarrow} N'$ to indicate (only) the time progress.

Proposition 1 *For any systems N, N' and N'', the following hold:*

1. *If $N \overset{t}{\rightsquigarrow} N'$ and $N \overset{t}{\rightsquigarrow} N''$, then $N' \equiv N''$;*
2. *$N \overset{(t+t')}{\rightsquigarrow} N'$ if and only if there is a N'' such that $N \overset{t}{\rightsquigarrow} N''$ and $N'' \overset{t'}{\rightsquigarrow} N'$.*

The f rst item of Proposition 1 states that the passage of time does not introduce any nondeterminism into the execution of a process. Moreover, if a process is able to evolve to a certain time t, then it must evolve through every time moment before t; this ensures that the process evolves continuously.

3. TRAVEL AGENCY EXAMPLE IN RTIMO

To illustrate the syntax and semantics of rTiMo, we use an example describing an understaffed travel agency, presented also in Ciobanu and Rotaru (2013). We assume that the agency has a central off ce (where the executives interact with agents) and six local off ces (where agents interact with customers). However, due to massive layoffs, the agency has only three travel agents available, whose jobs are to communicate special travel packages (destinations and the costs of the travel) to potential customers, and two executives whose only jobs are to assign the travel agents to certain local off ces of the agency each day (not necessarily the same local off ce each day). Also, there are two potential customers who are interested in the recommendations made by the agency, by visiting some of the local agencies (the ones that are close to their homes). We assume that the behaviours of the agency staff and of the potential customers are cyclic, and can be described as rTiMo processes.

The f rst agent (i.e., process $\mathrm{Agent1}$) leaves its home (i.e., the location $\mathrm{home}_{\mathrm{Agent1}}$) and goes to the central off ce of the agency (i.e., location office) in order to be assigned a certain local off ce for the current day (i.e., a location that will replace the location variable newloc). After arriving at the central off ce, it has to communicate with one of the executives on channel b after signing the attendance register, any time between 1 to 5 minutes (depending on

(STOP) $$l[[0]] \overset{\lambda}{\not\rightarrow}$$

(DSTOP) $$l[[0]] \overset{t}{\rightsquigarrow} l[[0]]$$

(DMOVE) $$\frac{t_2 \geq t' > 0}{l[[\mathsf{go}^{[t_1,t_2]} l' \ \mathsf{then}\ P]] \overset{t'}{\rightsquigarrow} l[[\mathsf{go}^{[max\{0,t_1-t'\},t_2-t']} l' \ \mathsf{then}\ P]]}$$

(MOVE0) $$l[[\mathsf{go}^{[0,t]} l' \ \mathsf{then}\ P]] \xrightarrow{l \triangleright l'} l'[[P]]$$

(COM) $$l[[a^{[0,t]}!\langle v\rangle \ \mathsf{then}\ P \ \mathsf{else}\ Q \mid a^{[0,t']}?(u) \ \mathsf{then}\ P' \ \mathsf{else}\ Q']] \xrightarrow{\{v/u\}@l} l[[P \mid \{v/u\}P']]$$

(DPUT) $$\frac{t_2 \geq t' > 0}{l[[a^{[t_1,t_2]}!\langle v\rangle \ \mathsf{then}\ P \ \mathsf{else}\ Q]] \overset{t'}{\rightsquigarrow} l[[a^{[max\{0,t_1-t'\},t_2-t']}!\langle v\rangle \ \mathsf{then}\ P \ \mathsf{else}\ Q]]}$$

(PUT0) $$l[[a^{[0,0]}!\langle v\rangle \ \mathsf{then}\ P \ \mathsf{else}\ Q]] \xrightarrow{a!^{[0,0]}@l} l[[Q]]$$

(DGET) $$\frac{t_2 \geq t' > 0}{l[[a^{[t_1,t_2]}?(u) \ \mathsf{then}\ P \ \mathsf{else}\ Q]] \overset{t'}{\rightsquigarrow} l[[a^{[max\{0,t_1-t'\},t_2-t']}?(u) \ \mathsf{then}\ P \ \mathsf{else}\ Q]]}$$

(GET0) $$l[[a^{[0,0]}?(u) \ \mathsf{then}\ P \ \mathsf{else}\ Q]] \xrightarrow{a?^{[0,0]}@l} l[[Q]]$$

(DCALL) $$\frac{l[[\{v/x\}P_{id}]] \overset{t}{\rightsquigarrow} l[[P'_{id}]]}{l[[id(v)]] \overset{t}{\rightsquigarrow} l[[P'_{id}]]} \ \text{where } id(v) \overset{def}{=} P_{id}$$

(CALL) $$\frac{l[[\{v/x\}P_{id}]] \xrightarrow{id@l} l[[P'_{id}]]}{l[[id(v)]] \xrightarrow{id@l} l[[P'_{id}]]} \ \text{where } id(v) \overset{def}{=} P_{id}$$

(DPAR) $$\frac{N_1 \overset{t}{\rightsquigarrow} N'_1 \qquad N_2 \overset{t}{\rightsquigarrow} N'_2}{N_1 \mid N_2 \overset{t}{\rightsquigarrow} N'_1 \mid N'_2}$$

(PAR) $$\frac{N_1 \xrightarrow{\Lambda_1} N'_1 \quad N_2 \xrightarrow{\Lambda_2} N'_2}{N_1 \mid N_2 \xrightarrow{\Lambda_1 \cup \Lambda_2} N'_1 \mid N'_2}$$

(DEQUIV) $$\frac{N \equiv N' \qquad N' \overset{t}{\rightsquigarrow} N'' \qquad N'' \equiv N'''}{N \overset{t}{\rightsquigarrow} N'''}$$

(EQUIV) $$\frac{N \equiv N' \qquad N' \xrightarrow{\Lambda} N'' \qquad N'' \equiv N'''}{N \xrightarrow{\Lambda} N'''}$$

Table 3: rTiMo *Operational Semantics*

the availability of one of the executives). Since the agent can use different means of transportations, and depending on the traff c, it can take between 5 to 10 minutes for the agent to reach the central off ce of the agency (this movement is described by the action $\texttt{go}^{[5,10]}\mathrm{office}\ \texttt{then}\ \mathrm{P}$). The agent then moves to the given location (it can take between 3 to 5 minutes depending on the local off ce it has to reach) and advertises (over channel a) the f rst destination on the agency's list (i.e., location dest_1), in the form of a holiday pack for 100 monetary units. Finally, after selling one travel package, the agent returns home (it can take between 5 to 8 minutes depending on the local off ce it departs from). The second and the third agent (i.e., processes $\mathrm{Agent2}$ and $\mathrm{Agent3}$) are similar to the f rst, but they have different homes (i.e., the locations $\mathrm{home}_{\mathrm{Agent2}}$ and $\mathrm{home}_{\mathrm{Agent3}}$), and advertise different destinations (i.e., locations dest_2 and dest_3), in the form of holiday packs for 200 and 300 monetary units, respectively. Formally, we have:

$$
\begin{aligned}
&\mathrm{AgentX}(\mathrm{home}_{\mathrm{AgentX}} : \mathrm{Loc}) = \\
&\quad \texttt{go}^{[5,10]}\mathrm{office}\ \texttt{then}\ \mathrm{AgentX}(\mathrm{office} : \mathrm{Loc}) \\
&\mathrm{AgentX}(\mathrm{office} : \mathrm{Loc}) = \\
&\quad b^{[1,5]}?(\mathrm{newloc} : \mathrm{Loc}) \\
&\qquad \texttt{then}\ (\texttt{go}^{[3,5]}\ \mathrm{newloc} \\
&\qquad\quad \texttt{then}\ \mathrm{AgentX}(\mathrm{newloc} : \mathrm{Loc})) \\
&\qquad \texttt{else}\ \mathrm{AgentX}(\mathrm{office} : \mathrm{Loc}) \\
&\mathrm{AgentX}(\mathrm{office_i} : \mathrm{Loc}) = \\
&\quad a_i^{[1,20]}!\langle \mathrm{dest}_{\mathrm{X}}, 100 \cdot X\rangle \\
&\qquad \texttt{then}\ \texttt{go}^{[1,3]}\ \mathrm{home}_{\mathrm{AgentX}} \\
&\qquad\quad \texttt{then}\ \mathrm{AgentX}(\mathrm{home}_{\mathrm{AgentX}} : \mathrm{Loc}) \\
&\qquad \texttt{else}\ \texttt{go}^{[1,3]}\ \mathrm{home}_{\mathrm{AgentX}} \\
&\qquad\quad \texttt{then}\ \mathrm{AgentX}(\mathrm{home}_{\mathrm{AgentX}} : \mathrm{Loc}),
\end{aligned}
$$

where $1 \leq i \leq 6$ and $X \in \{1, 2, 3\}$ refers to the number of the agent.

The two executives (i.e., processes $\mathrm{Executive1}$ and $\mathrm{Executive2}$) reside at the central off ce (i.e., location office), and each chooses a local off ce (i.e., in a cyclic manner, from the locations office_1, office_3, office_5, for process $\mathrm{Executive1}$, and the locations office_2, office_4, office_6 for process $\mathrm{Executive2}$) that will be assigned to the next agent that comes to the central off ce (over channel b in a period of time between 1 and 5 time units for $\mathrm{Executive1}$ and a period of time between 2 and 4 time units for $\mathrm{Executive2}$, namely after each executive resolves some off ce paperwork that make different periods for the two executives). Formally, we have:

$$
\begin{aligned}
&\mathrm{Executive1}(\mathrm{office}_1 : \mathrm{Loc}) = \\
&\quad b^{[1,5]}!\langle \mathrm{office}_1\rangle \\
&\qquad \texttt{then}\ \mathrm{Executive1}(\mathrm{office}_3 : \mathrm{Loc}) \\
&\qquad \texttt{else}\ \mathrm{Executive1}(\mathrm{office}_1 : \mathrm{Loc}) \\
&\mathrm{Executive1}(\mathrm{office}_3 : \mathrm{Loc}) = \\
&\quad b^{[1,5]}!\langle \mathrm{office}_3\rangle \\
&\qquad \texttt{then}\ \mathrm{Executive1}(\mathrm{office}_5 : \mathrm{Loc}) \\
&\qquad \texttt{else}\ \mathrm{Executive1}(\mathrm{office}_3 : \mathrm{Loc}) \\
&\mathrm{Executive1}(\mathrm{office}_5 : \mathrm{Loc}) = \\
&\quad b^{[1,5]}!\langle \mathrm{office}_5\rangle \\
&\qquad \texttt{then}\ \mathrm{Executive1}(\mathrm{office}_1 : \mathrm{Loc}) \\
&\qquad \texttt{else}\ \mathrm{Executive1}(\mathrm{office}_5 : \mathrm{Loc})
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Executive2}(\mathrm{office}_2 : \mathrm{Loc}) = \\
&\quad b^{[2,4]}!\langle \mathrm{office}_2\rangle \\
&\qquad \texttt{then}\ \mathrm{Executive2}(\mathrm{office}_4 : \mathrm{Loc}) \\
&\qquad \texttt{else}\ \mathrm{Executive2}(\mathrm{office}_2 : \mathrm{Loc}) \\
&\mathrm{Executive2}(\mathrm{office}_4 : \mathrm{Loc}) = \\
&\quad b^{[2,4]}!\langle \mathrm{office}_4\rangle \\
&\qquad \texttt{then}\ \mathrm{Executive2}(\mathrm{office}_6 : \mathrm{Loc}) \\
&\qquad \texttt{else}\ \mathrm{Executive2}(\mathrm{office}_4 : \mathrm{Loc}) \\
&\mathrm{Executive2}(\mathrm{office}_6 : \mathrm{Loc}) = \\
&\quad b^{[2,4]}!\langle \mathrm{office}_6\rangle \\
&\qquad \texttt{then}\ \mathrm{Executive2}(\mathrm{office}_2 : \mathrm{Loc}) \\
&\qquad \texttt{else}\ \mathrm{Executive2}(\mathrm{office}_6 : \mathrm{Loc})
\end{aligned}
$$

The f rst customer (i.e., process $\mathrm{Client1}$) leaves home (i.e., location $\mathrm{home}_{\mathrm{C1}}$) when he knows the agencies should be open and visits all of the three local off ces of the agency that are closest to his home (i.e., the locations office_1, office_2, and office_3), in order, receives travel offers from the agents found at those local off ces, and chooses the cheapest travel destination. Then, he goes to the desired destination, spends a certain amount of time there, after which he returns home. The second customer (i.e., process $\mathrm{Client2}$) has the same behaviour as the f rst, except that he has a different home (i.e., location $\mathrm{home}_{\mathrm{Client2}}$), the off ces closest to his home are locations office_4, office_5 and office_6, and that he chooses the most expensive travel destination. For simplicity, we consider that both clients have the same intervals of performing similar actions. Formally, we have:

$\mathrm{Client1}(\mathrm{home}_{\mathrm{Client1}} : \mathrm{Loc}) =$

go$^{[12,13]}$ office_1

then $\mathrm{Client1}(\mathrm{office}_1 : \mathrm{Loc})$

$\mathrm{Client1}(\mathrm{office}_1 : \mathrm{Loc}) =$

$a_1^{[0,4]}?(\mathrm{dest}_{\mathrm{Client1,1}} : \mathrm{Loc}, \mathrm{cost}_{\mathrm{Client1,1}} : \mathbb{N})$

then (go$^{[1,2]}$ office_2

then $\mathrm{Client1}(\mathrm{office}_2 : \mathrm{Loc})$)

else (go$^{[1,2]}$ office_2

then $\mathrm{Client1}(\mathrm{office}_2 : \mathrm{Loc})$)

$\mathrm{Client1}(\mathrm{office}_2 : \mathrm{Loc}) =$

$a_2^{[0,4]}?(\mathrm{dest}_{\mathrm{Client1,2}} : \mathrm{Loc}, \mathrm{cost}_{\mathrm{Client1,2}} : \mathbb{N})$

then (go$^{[1,3]}$ office_3

then $\mathrm{Client1}(\mathrm{office}_3 : \mathrm{Loc})$)

else (go$^{[1,3]}$ office_3

then $\mathrm{Client1}(\mathrm{office}_3 : \mathrm{Loc})$)

$\mathrm{Client1}(\mathrm{office}_3 : \mathrm{Loc}) =$

$a_3^{[0,4]}?(\mathrm{dest}_{\mathrm{lientC1,3}} : \mathrm{Loc}, \mathrm{cost}_{\mathrm{Client1,3}} : \mathbb{N})$

then (go$^{[1,5]}$ $\mathrm{next}_{\mathrm{Client1}}$

then $\mathrm{Client1}(\mathrm{next}_{\mathrm{Client1}} : \mathrm{Loc})$)

else (go$^{[1,5]}$ $\mathrm{next}_{\mathrm{Client1}}$

then $\mathrm{Clent1}(\mathrm{next}_{\mathrm{Client1}} : \mathrm{Loc})$)

$\mathrm{Client1}(\mathrm{dest}_{\mathrm{Client1,i}} : \mathrm{Loc}) =$

go$^{[1,5]}$ $\mathrm{home}_{\mathrm{Client1}}$

then $\mathrm{Client1}(\mathrm{home}_{\mathrm{Client1}} : \mathrm{Loc})$, for $1 \le i \le 3$

where

$$\mathrm{next}_{\mathrm{Client1}} = \begin{cases} \mathrm{dest}_{\mathrm{Client1,i}} & \text{if } \mathrm{cost}_{\mathrm{Client1,i}} = \\ & min_{j\in\{1,2,3\}}\mathrm{cost}_{\mathrm{Client_1,j}} \in \mathbb{N} \\ \mathrm{home}_{\mathrm{Client1}} & \text{otherwise.} \end{cases}$$

$\mathrm{Client2}(\mathrm{home}_{\mathrm{Client2}} : \mathrm{Loc}) =$

go$^{[12,13]}$ office_4

then $\mathrm{Client1}(\mathrm{office}_4 : \mathrm{Loc})$

$\mathrm{Client2}(\mathrm{office}_1 : \mathrm{Loc}) =$

$a_4^{[0,4]}?(\mathrm{dest}_{\mathrm{Client2,1}} : \mathrm{Loc}, \mathrm{cost}_{\mathrm{Client2,1}} : \mathbb{N})$

then (go$^{[1,2]}$ office_2

then $\mathrm{Client2}(\mathrm{office}_5 : \mathrm{Loc})$)

else (go$^{[1,2]}$ office_2

then $\mathrm{Client2}(\mathrm{office}_5 : \mathrm{Loc})$)

$\mathrm{Client2}(\mathrm{office}_5 : \mathrm{Loc}) =$

$a_5^{[0,4]}?(\mathrm{dest}_{\mathrm{Client2,2}} : \mathrm{Loc}, \mathrm{cost}_{\mathrm{Client2,2}} : \mathbb{N})$

then (go$^{[1,3]}$ office_6

then $\mathrm{Client2}(\mathrm{office}_6 : \mathrm{Loc})$)

else (go$^{[1,3]}$ office_6

then $\mathrm{Client2}(\mathrm{office}_6 : \mathrm{Loc})$)

$\mathrm{Client2}(\mathrm{office}_6 : \mathrm{Loc}) =$

$a_6^{[0,4]}?(\mathrm{dest}_{\mathrm{Client2,3}} : \mathrm{Loc}, \mathrm{cost}_{\mathrm{Client2,3}} : \mathbb{N})$

then (go$^{[1,5]}$ $\mathrm{next}_{\mathrm{Client2}}$

then $\mathrm{Client2}(\mathrm{next}_{\mathrm{Client2}} : \mathrm{Loc})$)

else (go$^{[1,5]}$ $\mathrm{next}_{\mathrm{Client2}}$

then $\mathrm{Client2}(\mathrm{next}_{\mathrm{Client2}} : \mathrm{Loc})$)

$\mathrm{Client2}(\mathrm{dest}_{\mathrm{Client2,i}} : \mathrm{Loc}) =$

go$^{[1,5]}$ $\mathrm{home}_{\mathrm{Client2}}$

then $\mathrm{Client2}(\mathrm{home}_{\mathrm{Client2}} : \mathrm{Loc})$, for $1 \le i \le 3$

where

$$\mathrm{next}_{\mathrm{Client2}} = \begin{cases} \mathrm{dest}_{\mathrm{Client2,i}} & \text{if } \mathrm{cost}_{\mathrm{Client2,i}} = \\ & max_{j\in\{1,2,3\}}\mathrm{cost}_{\mathrm{Client2,j}} \in \mathbb{N} \\ \mathrm{home}_{\mathrm{Client2}} & \text{otherwise.} \end{cases}$$

The initial state of the corresponding rTiMo network N is:

$$\begin{gathered}\mathrm{home}_{\mathrm{Agent1}}[[\mathrm{Agent1}(\mathrm{home}_{\mathrm{Agent1}})]] \mid \\ \mid \mathrm{home}_{\mathrm{Agent2}}[[\mathrm{Agent2}(\mathrm{home}_{\mathrm{Agent2}})]] \mid \\ \mid \mathrm{home}_{\mathrm{Agent3}}[[\mathrm{Agent3}(\mathrm{home}_{\mathrm{Agent3}})]] \mid \\ \mid \mathrm{office}[[\mathrm{Executive1}(\mathrm{office}_1) \mid \mathrm{Executive2}(\mathrm{office}_2)]] \mid \\ \mid \mathrm{home}_{\mathrm{Client1}}[[\mathrm{Client1}(\mathrm{home}_{\mathrm{Client1}})]] \mid \\ \mid \mathrm{home}_{\mathrm{Client2}}[[\mathrm{Client2}(\mathrm{home}_{\mathrm{Client2}})]]\end{gathered}$$

4. TIMED SAFETY AUTOMATA

Towards a necessary automated verif cation of complex distributed systems described by rTiMo , we provide f rst a relationship between rTiMo and timed safety automata Alur and Dill (1994). Then, taking into consideration the existing software tools, we relate rTiMo to UPPAAL . UPPAAL is an integrated tool environment for modelling, validation and verif cation of real-time systems. More details about the semantics of the input language of UPPAAL can be found at http://www.uppaal.org/. Modelling and verif cation of real-time systems by using UPPAAL are presented in Hessel et all (2008). Such a system is modelled as a network of several parallel timed automata. All the clocks are evaluated to real-numbers and progress synchronously. The model uses variables just as in programming languages: they are read, written, and are subject to linear expressions. A state of the system is def ned by the locations of the network, the clock constraints, and the values of the variables. In any state, every automaton from the network may f re an edge (if it satisf es the restrictions) separately or synchronize with another automaton, leading to a new state.

4.1. Syntax

Assume a f nite set of real-valued variables $\mathcal{C}$ ranged over by x, y, ... standing for clocks, and a f nite alphabet Σ ranged over by a, b, ... standing for actions. A clock constraint g is a conjunctive formula of constraints of the form $x \sim m$ or $x - y \sim m$, for $x, y \in \mathcal{C}$, $\sim \in \{\leq, <, =, >, \geq\}$, and $m \in \mathbb{N}$. The set of clock constraints is denoted by $\mathcal{B}(\mathcal{C})$.

Definition 1 *A* **timed safety automaton** $\mathcal{A}$ *is a tuple* $\langle N, n_0, E, I\rangle$, *where*

- N *is a f nite set of nodes;*
- n_0 *is the initial node;*
- $E \subseteq N \times \mathcal{B}(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times N$ *is the set of edges;*
- $I : N \to \mathcal{B}(\mathcal{C})$ *assigns invariants to nodes.*

$n \xrightarrow{g,a,r} n'$ *is a shorthand notation for* $\langle n, g, a, r, n'\rangle \in E$. *Node invariants are restricted to constraints of the form:* $x \leq m$ *or* $x < m$ *where* $m \in \mathbb{N}$.

4.2. Networks of Timed Automata

A network of timed automata is the parallel composition $\mathcal{A}_1 \mid \ldots \mid \mathcal{A}_n$ of a set of timed automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ combined into a single system using the parallel composition operator and with all internal actions hidden. Synchronous communication inside the network is by handshake synchronisation of input and output actions. In this case, the action alphabet Σ consists of $a?$ symbols (for input actions), $a!$ symbols (for output actions), and τ symbols (for internal actions). A detailed example is found in Henzinger (1994).

A network can perform delay transitions (delay for some time), and action transitions (follow an enabled edge). An action transition is enabled if the clock assignment also satisf es all integer guards on the corresponding edges. In synchronisation transitions, the resets on the edge with an output-label are performed before the resets on the edge with an input-label. To model urgent synchronisation transitions that should be taken as soon as they are enabled (the system may not delay), a notion of urgent channels is used.

Let u, v, ... denote clock assignments mapping $\mathcal{C}$ to non-negative reals $\mathbb{R}_+$. $g \models u$ means that the clock values u satisfy the guard g. For $d \in \mathbb{R}_+$, the clock assignment mapping all $x \in \mathcal{C}$ to $u(x) + d$ is denoted by $u + d$. Also, for $r \subseteq \mathcal{C}$, the clock assignment mapping all clocks of r to 0 and agreeing with u for the other clocks in $\mathcal{C} \backslash r$ is denoted by $[r \mapsto 0]u$. Let n_i stand for the ith element of a node vector n, and $n[n'_i/n_i]$ for the vector n with n_i being substituted with n'_i.

A network state is a pair $\langle n, u\rangle$, where n denotes a vector of current nodes of the network (one for each automaton), and u is a clock assignment storing the current values of all network clocks and integer variables.

Definition 2 *The operational semantics of a timed automaton is a transition system where states are pairs* $\langle n, u\rangle$ *and transitions are def ned by the rules:*

- $\langle n, u\rangle \xrightarrow{d} \langle n, u{+}d\rangle$ *if* $u \in I(n)$ *and* $(u{+}d) \in I(n)$, *where* $I(n) = \bigwedge I(n_i)$;
- $\langle n, u\rangle \xrightarrow{\tau} \langle n[n'_i/n_i], u'\rangle$ *if* $n_i \xrightarrow{g,\tau,r} n'_i$, $g \models u$, $u' = [r \mapsto 0]u$ *and* $u' \in I(n[n'_i/n_i])$;
- $\langle n, u\rangle \xrightarrow{\tau} \langle n[n'_i/n_i][n'_j/n_j], u'\rangle$ *if there exist* $i \neq j$ *such that*
  1. $n_i \xrightarrow{g_i,a?,r_i} n'_i$, $n_j \xrightarrow{g_j,a!,r_j} n'_j$, $g_i \wedge g_j \models u$,
  2. $u' = [r_i \mapsto 0]([r_j \mapsto 0]u)$ *and* $u' \in I(n[n'_i/n_i][n'_j/n_j])$.

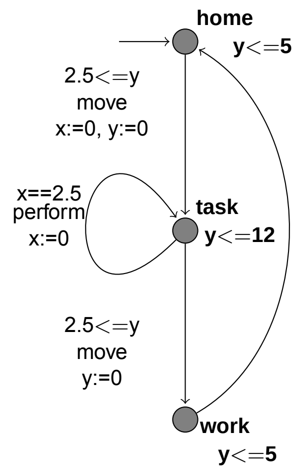


Figure 1: *Timed Safety Automata*

Graphically, a timed safety automata can be represented as a graph having a finite set of nodes and a finite set of labelled edges (representing transitions), using real-timed variables (representing the clocks of the system). The clocks are initialised with zero when the system starts, and then increased synchronously with the same rate. The behaviour of the automaton is restricted by using clock constraints, i.e. guards on edges, and local timing constraints called *node invariants* (e.g., see Figure 1). An automaton is allowed to stay in a node as long as the timing conditions of that node are satisfied. A transition can be taken when the edge guards are satisfied by clocks values. When a transition is taken, clocks may be reset to zero.

Building a timed automaton for each located process of rTiMo leads to the next result about the equivalence between an rTiMo network N and its corresponding timed automaton $\mathcal{A}_N$.

Theorem 1 *Given an* rTiMo *network N with channels appearing only once in output actions, there exists a network $\mathcal{A}_N$ of several parallel timed automata with a bisimilar behaviour.*

A bisimilar behaviour is given by:

- at the start of execution, all clocks in rTiMo and their corresponding timed automata are set to 0;
- the consumption of a go action in a node l is matched by a τ edge obtained by translation;
- a communication rule is matched by a synchronization between the edges obtained by translations;
- the passage of time is similar in both formalisms: in rTiMo the global clock is used to decrement by d all timers in the network when no action is possible, while in the timed automata all local clocks are decremented synchronously with the same value d.

Thus, the size of a timed safety automata $\mathcal{A}_N$ is polynomial with respect to the size of a TiMo network N, and the state spaces have the same number of states.

5. MODELLING THE TRAVEL AGENCY EXAMPLE IN UPPAAL

The model of the travel agency of Section 3 has three templates:

- $\mathrm{Agent(int\ dest)}$ is the model of an agent with one integer parameter dest, as shown in Figure 2. Using the parameter dest we can initialize the three agents from Section 3 by creating the processes $\mathrm{A1} = \mathrm{Agent}(1)$, $\mathrm{A2} = \mathrm{Agent}(2)$ and $\mathrm{A3} = \mathrm{Agent}(3)$, where each agent sells a travel package to a different destination dest. It is also possible to instantiate any number of agents, or agents that sell the same travel package.
- $\mathrm{Executive(int\ o1, int\ o2, int\ o3)}$ is the model of an executive with three integer parameters $\mathrm{o1}$, $\mathrm{o2}$ and $\mathrm{o3}$, shown in Figure 2. The three parameters are used to initiate the two executives described in Section 3 by creating the processes $\mathrm{E1} = \mathrm{Executive}(1, 3, 5)$ and $\mathrm{E2} = \mathrm{Executive}(2, 4, 6)$, where each executive is given the office locations that he/she can assign to agents. As for the agents, it is possible to create more executives than the ones presented in Section 3.
- $\mathrm{Client(int\ id, int\ o1, int\ o2, int\ o3)}$ is the model of a client with four parameters, shown in Figure 3. The parameters are used to initiate the two clients of Section 3 by creating the processes $\mathrm{C1} = \mathrm{Client}(1, 1, 2, 3)$ and $\mathrm{C2} = \mathrm{Client}(2, 4, 5, 6)$. The id parameter is used to uniquely identify a client, while the other parameters are used to identify three local offices each client is allowed to visit before making a travel decision. As for Agent and $\mathrm{Executive}$, any number of clients can be created.

Thus, the initial system is

$$\mathrm{system\ A1, A2, A3, E1, E2, C1, C2.}$$

We explain in detail the Agent template of Figure 2 (the others are constructed in a similar manner). It has five locations: home, office, $\mathrm{office\_b}$, $\mathrm{office\_o}$

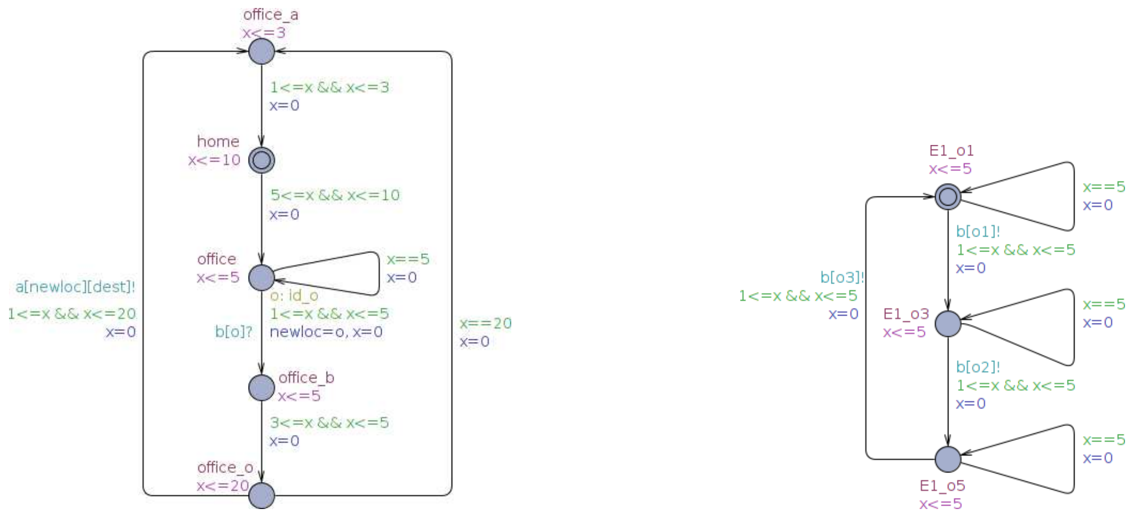


Figure 2: *The Agent (left) and Executive (right) Templates*

and $\mathrm{office\_a}$. The initial location is home, which corresponds to the fact that an agent is at home. The location has the invariant $x \ <= \ 10$ (taken from the rTiMo action $\mathrm{go}^{[5,10]}$) which has the effect that the location must be left within 10 time units. The outgoing transition towards location office is guarded by the constraints $5 \ <= \ x$ and $x \ <= 10$, which correspond to the above mentioned go rTiMo action. Once at location office, the agent can either synchronize on channel $b[o]$ with an executive or, if the channel expires, create another instance in order to be able to receive an off ce location from an executive. After the communication is performed, the agent is at location $\mathrm{office\_b}$ (meaning that it successfully received the location $\mathrm{newloc} = \mathrm{o}$ of the off ce he is detached to), and is ready to move to the assigned location $\mathrm{office_o}$. After arriving at $\mathrm{office_o}$, he awaits for a client for at most 20 time units, to which he must communicate the travel package on channel $\mathrm{a[newloc][dest]!}$. Regardless of the fact that he interacts with a client or not, he moves within 20 time units to the location $\mathrm{office\_b}$ where he is ready to go home in order to prepare for a new working day. Thus, an agent has a cyclic evolution (a similar behaviour as one of the executives and of the clients).

6. VERIFYING PROPERTIES OF TRAVEL AGENCY BY USING UPPAAL

According to the results and descriptions presented in the previous section, we can verify time bounded distributed systems with mobility presented as rTiMo networks by using UPPAAL. UPPAAL can be used to check temporal properties of networks of timed automata, properties expressed in Computation Tree Logic (CTL). If ϕ and ψ are boolean expressions over predicates on nodes, integer variables and clock constraints, then the formulas have the following forms:

A $[\,]\ \phi$ - Invariantly ϕ; A $\langle\,\rangle\ \phi$ - Always Eventually ϕ;

E $[\,]\ \phi$ - Potentially Always ϕ; E $\langle\,\rangle\ \phi$ - Possibly ϕ;

$\phi \rightsquigarrow \psi$ - ϕ always leads to ψ. This is a shorthand for A $[\,]$ $(\phi \Rightarrow$ A $\langle\,\rangle\ \psi)$

The formulas can be of two types: path formulae (quantify over paths or traces of the model) and state formulae (individual states). Path formulae can be classif ed into reachability (E $\langle\,\rangle\ \phi$), safety (A $[\,]\ \phi$ and E $[\,]\ \phi$) and liveness (A $\langle\,\rangle\ \phi$ and $\phi \rightsquigarrow \psi$). Reachability properties are used to check whether there exist a path starting at the initial state, such that ϕ is eventually satisf ed along that path. Safety properties are used to verify that something bad will never happen, while liveness properties check whether something will eventually happen.

We present various properties that could be analyzed and verif ed for our example from Section 3. We have used an Intel PC with 8 GB memory, 2.50 GHz $\times$ 4 CPU and 64-bit Ubuntu 14.04 LTS to run the experiments. The results are presented for each analyzed property.

Example 1 *Given the uncertainty of the delay in migration and communication, the size of the potential interactions in* rTiMo *systems grows exponential making the software verif cation a necessity. We use* UPPAAL *to perform this kind of verif cations for the travel agency example presented in Section 3, for both safety and liveness properties. Here we present only some of the formulas/properties verif ed by using* UPPAAL*.*

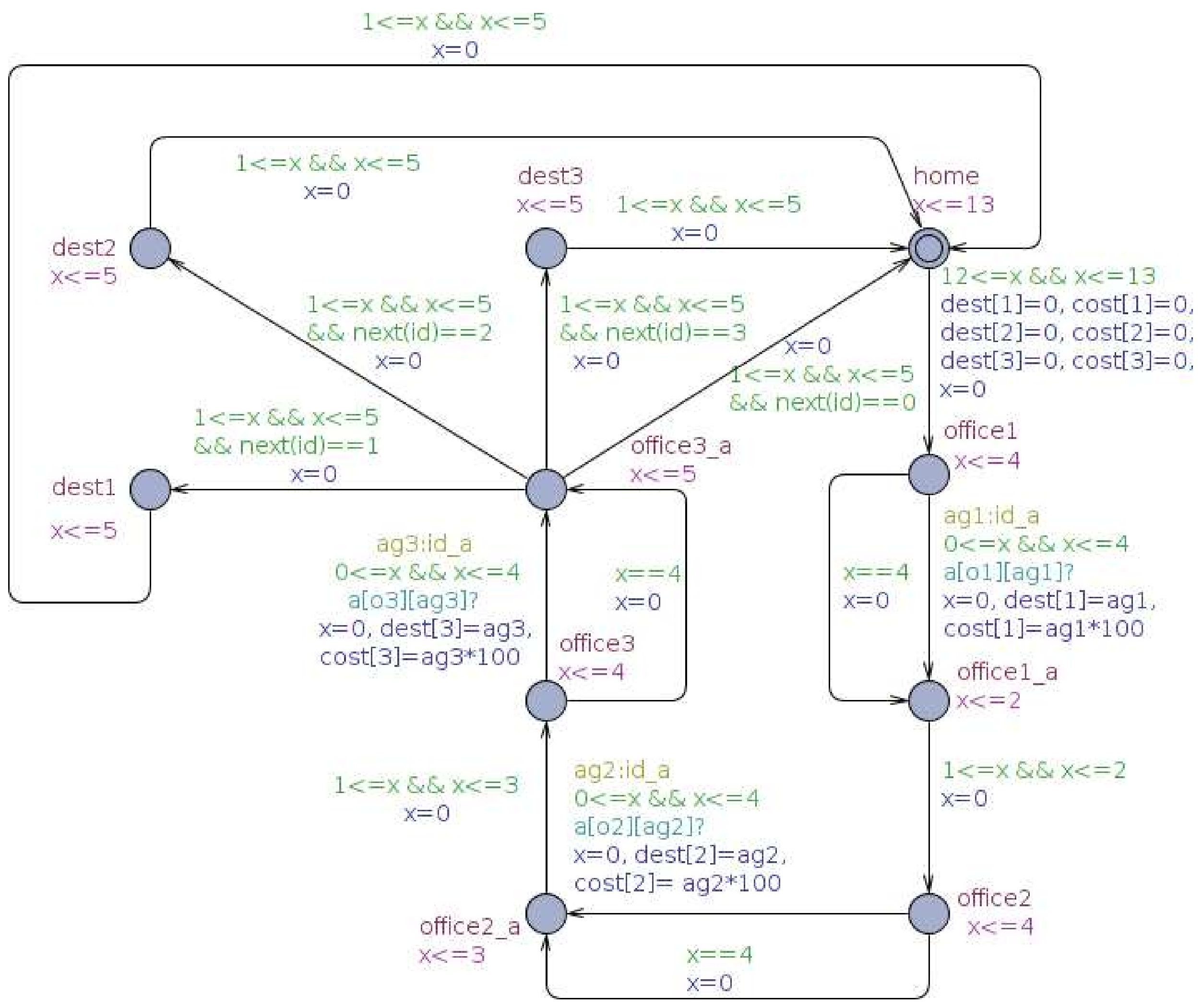
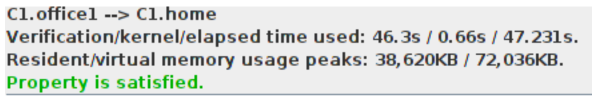


Figure 3: *The Client Template*

- $\mathrm{C1.office1} - - > \mathrm{C1.home}$

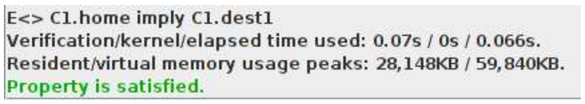
  The formulae $\mathrm{C1.office1} \rightsquigarrow \mathrm{C1.home}$*, shorthand for* $\mathsf{A}[\,](\mathrm{C1.office1} \Rightarrow \mathsf{A}\langle\,\rangle \mathrm{C1.home}$*, describe that, once the client* C1 *is in the* office1 *location, then it will always reach the* home *location. This implies that after leaving location* office1*, even whether the client visits or not the locations* office2 *and* office3*, the client* C1 *goes to the desired location, after which returns* home*.*



```
C1.office1 --> C1.home
Verification/kernel/elapsed time used: 46.3s / 0.66s / 47.231s.
Resident/virtual memory usage peaks: 38,620KB / 72,036KB.
Property is satisfied.
```

- $\mathsf{E}\langle\,\rangle$ C1.home imply C1.dest1

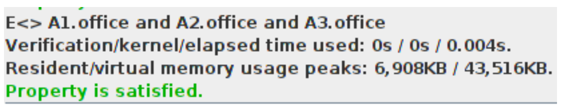
  This formulae is used to check whether once the client C1 *is in the* home *location, then it possibly reaches the* dest1 *location. This implies that if the client* C1 *leaves the* home *location, one of its travels takes him to location* dest1*. In a similar manner it can be checked that there are evolutions in which the client* C1 *visits one of the locations* dest2 *or* dest3*.*



```
E<> C1.home imply C1.dest1
Verification/kernel/elapsed time used: 0.07s / 0s / 0.066s.
Resident/virtual memory usage peaks: 28,148KB / 59,840KB.
Property is satisfied.
```

- $\mathsf{E}\langle\,\rangle$ A1.office and A2.office and A3.office

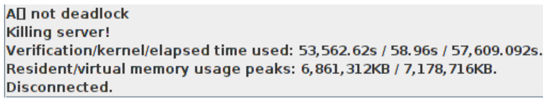
  This is checking whether or not the agents A1*,* A2 *and* A3 *reach the* office *location at the same time. Due to the uncertainty of the delay in migration and communication and of the fact that each agent has different timing constraints, having all agents at location* office *may not happen in all the possible evolutions.*



```
E<> A1.office and A2.office and A3.office
Verification/kernel/elapsed time used: 0s / 0s / 0.004s.
Resident/virtual memory usage peaks: 6,908KB / 43,516KB.
Property is satisfied.
```

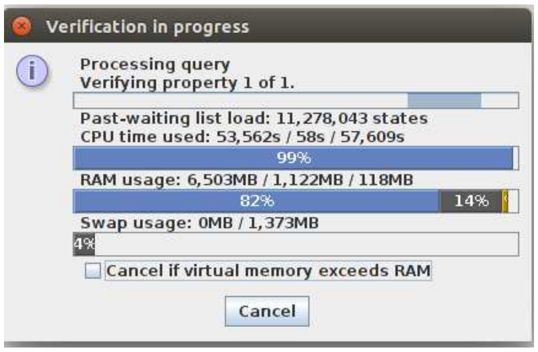
- A[ ] not deadlock

  This is checking that there exists no deadlock. This implies that, whatever are the interactions between the involved participants, the evolution never stops. This means that after a working day the agents go the next day to work, while the clients continue to look for travel packages in the following days.



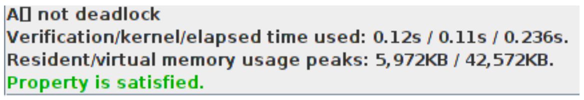
```
A[] not deadlock
Killing server!
Verification/kernel/elapsed time used: 53,562.62s / 58.96s / 57,609.092s.
Resident/virtual memory usage peaks: 6,861,312KB / 7,178,716KB.
Disconnected.
```

For this property, we have stopped the verif cation process after 57609 seconds due to the fact that the RAM was fully used (as it can be seen below).



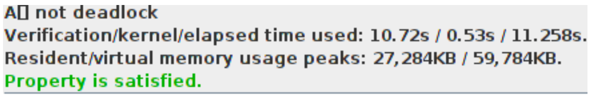
```
Verification in progress
Processing query
Verifying property 1 of 1.
Past-waiting list load: 11,278,043 states
CPU time used: 53,562s / 58s / 57,609s
99%
RAM usage: 6,503MB / 1,122MB / 118MB
82%  14%
Swap usage: 0MB / 1,373MB
4%
Cancel if virtual memory exceeds RAM
Cancel
```

Since the verif cation of the previous property was stopped due to insuff cient RAM, we try a similar verif cation of "no deadlock" for smaller systems. For systems with a smaller number of agents, executives and clients the "no deadlock" property is satisf ed. For one agent, one executive and one client the Uppaal *verif cation returned:*



```
A[] not deadlock
Verification/kernel/elapsed time used: 0.12s / 0.11s / 0.236s.
Resident/virtual memory usage peaks: 5,972KB / 42,572KB.
Property is satisfied.
```

Adding another agent takes more time to verify, but the property still holds:



```
A[] not deadlock
Verification/kernel/elapsed time used: 10.72s / 0.53s / 11.258s.
Resident/virtual memory usage peaks: 27,284KB / 59,784KB.
Property is satisfied.
```

Several other properties of rTiMo *systems can be verif ed by using* Uppaal *.*

7. CONCLUSION AND RELATED WORK

In this paper we presented time bounded extension of rTiMo , suitable to work in complex distributed systems with mobility. It is different from all previous approaches since it encompasses specif c features as real-time timeouts given as intervals, explicit locations, time bounded migration and communication. The parallel execution of a step is provided by multiset labelled transitions. We have presented an example of applying rTiMo to an understaffed travel agency, illustrating that rTiMo provides an appropriate framework for modelling and reasoning about time bounded distributed systems with migration and interaction/communication. We have shown that we can model and verify real-time systems (e.g., the travel agency) corresponding to rTiMo networks by using Uppaal . As rTiMo is a prototype language, a f exible representation of a travel agency is given as a number of parallel processes that are instances of the AX, EX and CX. The implementation of rTiMo processes in Uppaal is natural due to the fact that in Uppaal it is possible to use templates. For the running example this allows, by proper instantiations, to create any number of agents, executives and clients that can interact when placed in parallel. It is easy to note that the formalism presented in Aman and Ciobanu (2013) represents a strict subclass of the formalism presented in this paper.

Several proposals of process calculi for real-time modelling and verif cation have been presented in the literature: timed CSP Reed and Roscoe (1988), timed ACP Baeten and Bergstra (1991) and several timed extensions of CCS Moller and Tofts (1990); Yi et all (1994). Aiming to bridge the gap between the existing theoretical approach of process calculi and forthcoming realistic programming languages for distributed systems, we have introduced and studied a rather simple and expressive formalism called TiMo as a simplif ed version of timed distributed pi-calculus Ciobanu and Prisacariu (2006). In several aspects, TiMo is a prototyping language for multi-agent systems, featuring mobility and local interaction. Starting with a f rst version proposed in Ciobanu and Koutny (2008), several variants were developed during the last years; we mention here the access permissions given by a type system in perTiMo Ciobanu and Koutny (2011a), as well as a probabilistic extension pTiMo Ciobanu and Rotaru (2013). Inspired by TiMo , a f exible software platform was introduced in Ciobanu and Juravle (2009, 2012) to support the specif cation of agents allowing timed migration in a distributed environment Ciobanu (2010). Interesting properties of distributed systems described by TiMo refer to process migration, time constraints, bounded liveness and optimal reachability Aman et. all (2012); Ciobanu and Koutny (2011b). A verif cation tool called TiMo@PAT Ciobanu and Zheng (2013) was developed by using Process Analysis Toolkit (PAT), an extensible platform for model checkers. A probabilistic temporal logic called PLTM was introduced in Ciobanu and Rotaru (2013) to verify complex properties making explicit reference to specif c locations, temporal constraints over local clocks and multisets of actions.

Acknowledgements. Many thanks to the reviewers for their useful comments. The work was supported by a grant of the Romanian National Authority for Scientif c Research, project number PN-II-ID-PCE-2011-3-0919.

REFERENCES

Alur, R. and Dill, D.L. (1994) A Theory of Timed Automata. *Theoretical Computer Science* **126**, 183–235.

Aman, B. and Ciobanu, G. (2013) Real-Time Migration Properties of rTiMo Verif ed in Uppaal. In Hierons, R., Merayo, M.and Bravetti, M. (Eds.), SEFM 2013. *Lecture Notes in Computer Science* **8137**, 31–45.

Aman, B., Ciobanu, G. and Koutny, M. (2012) Behavioural Equivalences over Migrating Processes with Timers. In Giese, H. and Rosu, G. (Eds.) FMOODS/FORTE 2012, *Lecture Notes in Computer Science* **7273**, 52–66.

Baeten, J.C.M. and Bergstra, J.A. (1991) Real Time Process Algebra. *Journal of Formal Aspects of Computing Science* **3(2)**, 142–188.

Ciobanu, G. (2008) Behaviour Equivalences in Timed Distributed π-Calculus. In Wirsing, M., Banâtre, J.-P., Hölzl, M. and Rauschmayer, A. (Eds.), *Lecture Notes in Computer Science* **5380**, 190–208.

Ciobanu, G. (2010) Finding Network Resources by Using Mobile Agents. *Intelligent Distributed Computing IV. Studies in Computational Intelligence* **315**, 305–313.

Ciobanu, G. and Juravle, C. (2009) A Software Platform for Timed Mobility and Timed Interaction. In Lee, D., Lopes, A. and Poetzsch-Heffter, A. (Eds.) FMOODS/FORTE 2009, *Lecture Notes in Computer Science* **5522**, 106–121.

Ciobanu, G. and Juravle, C. (2012) Flexible Software Architecture and Language for Mobile Agents. *Concurrency and Computation: Practice and Experience***24**, 559–571.

Ciobanu, G. and Koutny, M. (2008) Modelling and Verif cation of Timed Interaction and Migration. In Fiadeiro, J.L., Inverardi, P. (Eds.) FASE 2008, *Lecture Notes in Computer Science* **4961**, 215–229.

Ciobanu, G. and Koutny, M. (2011) Timed Migration and Interaction With Access Permissions. In Butler, M., Schulte, W. (eds.) FM 2011, *Lecture Notes in Computer Science* **6664**, 293–307.

Ciobanu, G. and Koutny, M. (2011) Timed Mobility in Process Algebra and Petri nets. *The Journal of Logic and Algebraic Programming* **80(7)**, 377–391.

Ciobanu, G. and Prisacariu, C. (2006) Timers for Distributed Systems. In Di Pierro, A. and Wiklicky, H. (Eds.) QAPL 2006,*Electronic Notes in Theoretic Computer Science* **164(3)**, 81–99.

Ciobanu, G. and Rotaru, A. (2013) A Probabilistic Logic for pTiMo. In Liu, Z., Woodcock, J. and Zhu, H. (Eds.) ICTAC 2013, *Lecture Notes in Computer Science* **8049**, 141–158.

Ciobanu, G. and Zheng, M. (2013) Automatic Analysis of TiMo Systems in PAT. In *Proc. 18th International Conference on Engineering of Complex Computer Systems (ICECCS 2013)*, IEEE Computer Society, 121–124.

Hennessy, M. (2007) *A Distributed π-calculus*. Cambridge University Press.

Henzinger, T.A., Nicollin, X., Sifakis, J. and Yovine,S. (1994) Symbolic Model Checking for Real-time Systems. *Information and Computation* **111**, 192–224.

Hessel, A., Larsen, K.G., Mikucionis, M. Nielsen, B. Pettersson, P. and Skou, A. (2008) Testing Real-Time Systems Using Uppaal. In Hierons, R.M., Bowen, J.P., Harman, M. (Eds.) FORTEST, *Lecture Notes in Computer Science* **4949**, 77–117.

Milner, R., Parrow, J. and Walker, D. (1992) A Calculus of Mobile Processes (i-ii). *Information and Computation* **100**, 1–77.

Moller, F. and Tofts, C. (1990) A Temporal Calculus of Communicating Systems. In Baeten, J.C.M., Klop, J.W. (Eds.) CONCUR 1990, *Lecture Notes in Computer Science* **458**, 401–415.

Reed, G.M. and Roscoe, A.W. (1988) A Timed Model for Communicating Sequential Processes. *Theoretical Computer Science* **58(1-3)**, 249–261.

Yi, W. , Pettersson, P. and Daniels, M. (1994) Automatic Verif cation of Real-time Communicating Systems by Constraint-solving. In *International Conference on Formal Description Techniques*, 223–238.