Timed Mobility and Timed Communication for Critical Systems

Bogdan Aman(✉) and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science,
Blvd. Carol I no.11, 700506, Iaşi, Romania
baman@iit.tuiasi.ro, gabriel@info.uaic.ro

Abstract. We present a simple but elegant prototyping language for describing real-time systems including specific features as timeouts, explicit locations, timed migration and timed communication. The parallel execution of a step is provided by multiset labelled transitions. In order to illustrate its features, we describe a railway control system. Moreover, we define some behavioural equivalences matching multisets of actions that could happen in a given range of time (up to a timeout). We define the strong time-bounded bisimulation and the strong open time-bounded bisimulation, and prove that the latter one is a congruence. By using various bisimulations over the behaviours of real-time systems, we can check which behaviours are closer to an optimal and safe behaviour.

1 Introduction

To emphasize real-time aspects in critical systems, we use our prototyping language called rTiMo (real Timed Mobility) having specific features as timeouts, explicit locations, timed migration and timed communication. The timed constraints on migration and communication are used to coordinate interactions among various processes in time-aware systems. A notable advantage of using rTiMo to describe real-time critical systems is the possibility to express natural compositionality, explicit mobility, parallel execution of actions, scalable specification of complex systems in a modular fashion, and behavioural equivalences between matching multisets of actions that could happen in a given range of time (up to a timeout). Moreover, describing processes in rTiMo allows an automatic verification by using the model checking capabilities of Uppaal [1]. Here we emphasize on the behaviours of the critical systems depending not only on the order of actions, but also on the time at which the actions are performed. Thus, correctness and performances issues are closely related. When choosing which behavioural equivalence relation to adopt for a certain time-aware system, we should decide what properties should be preserved by the equivalence relation and how behaves the reliable system taken as reference. On the other hand, all the equivalence relations should be compositional with respect to the main constructs of the language: for example, if two systems are equivalent, then they remain equivalent when composed in parallel with the same third system. This allows compositional reasoning, and so each parallel component can be substituted by equivalent ones.

M. Núñez and M. Güdemann (Eds.): FMICS 2015, LNCS 9128, pp. 146–161, 2015.
DOI: 10.1007/978-3-319-19458-5_10

In critical systems the time issues are essential. A correct evolution depends not only on the actions taken, but also when the actions happen. A system may crash if an action is taken too early or too late. We illustrate how rTiMo works by describing a railway control system, a well-known example of a real-time system [11]. The system used in this paper is composed of two railways that intersect on a mobile bridge, together with several trains that want to cross the bridge. The mobile bridge is used to allow ships sail on the river below. The most important security rule is to avoid collision by prohibiting more than one train to cross the bridge at any given moment. The railway crossing is equipped with a controller that either allows or stops trains from crossing, depending on the state of the bridge (up or down). We use new temporal bisimilarities to define equivalence classes of trains offering similar services with respect to the waiting time (possibly up to an acceptable time difference). By using various bisimulations over the behaviours of real-time critical systems, we can identify which behaviours are closer to an optimal and safe behaviour (i.e., reductions work as expected) and compare it with sub-optimal ones containing faults (unacceptable reductions). The bisimulations can return some useful information about the compared processes: a qualitative indication that a sub-optimal behaviour might be present, and also quantitative information about the possible location or moment of a fault.

2 rTiMo: Syntax and Semantics

In rTiMo the processes can migrate between different locations of a distributed environment consisting of a number of explicit distinct locations. Timing constraints over migration and communication actions are used to coordinate processes in time and space. The passage of time in rTiMo is described with respect to a real-time global clock, while migration and communication actions are performed in a maximal parallel manner. Timing constraints for migration allow one to specify a temporal timeout after which a mobile process must move to another location. Two processes may communicate only if they are present at the same location. In rTiMo, the transitions caused by performing actions with timeouts are alternated with continuous transitions. The semantics of rTiMo is provided by multiset labelled transitions in which multisets of actions are executed in parallel (in one step).

Timing constraints applied to mobile processes allow us to specify how many time units are required by a process to move from one location to another. A timer in rTiMo is denoted by $^{\Delta t}$, where $t \in \mathbb{R}_+$. Such a timer is associated with a migration action such as $go^{\Delta t} bridge$ then P indicating that process P moves to location *bridge* after t time units. A timer $^{\Delta 5}$ associated with an output communication process $a^{\Delta 5}!\langle z\rangle$ then P else Q makes the channel a available for communication (namely it can send z) for a period of 5 time units. It is also possible to restrict the waiting time for an input communication process $a^{\Delta 4}?(x)$ then P else Q along a channel a; if the interaction does not happen before the timeout 4, the process gives up and continues as the alternative process Q.

The syntax of rTiMo is given in Table 1, where the following are assumed:

- a set *Loc* of locations, a set *Chan* of communication channels, and a set *Id* of process identifiers (each $id \in Id$ has its arity m_{id});
- for each $id \in Id$ there is a unique process definition $id(u_1, \ldots, u_{m_{id}}) \stackrel{def}{=} P_{id}$, where the distinct variables u_i are parameters;
- $a \in Chan$ is a communication channel; l is a location or a location variable;
- $t \in \mathbb{R}_+$ is a *timeout* of an action; u is a tuple of variables;
- v is a tuple of expressions built from values, variables and allowed operations.

Table 1. rTiMo Syntax

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

The only variable binding constructor is $a^{\Delta t}?(u)$ then P else Q that binds the variable u within P (but *not* within Q). $fv(P)$ is used to denote the free variables of a process P (and similarly for systems); for a process definition, is assumed that $fv(P_{id}) \subseteq \{u_1, \ldots, u_{m_{id}}\}$, where u_i are the process parameters. Processes are defined 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.

Mobility is provided by a process $go^{\Delta t} l$ then P that describes the migration from the current location to the location indicated by l after t time units. Since l can be a variable, and so its value is assigned dynamically through communication with other processes, this form of migration supports a flexible scheme for the movement of processes from one location to another. Thus, the behaviour can adapt 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 $l[[P]]$ specifies a process P running at location l, and a system is built from its components $L \mid N$. A system N is well-formed if there are no free variables in N.

Operational Semantics. The first 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 in Table 2 hold. 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 finite parallel composition of located processes of the form $l_1[[P_1]] \mid \ldots \mid l_n[[P_n]]$ 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 $l_1[[P_1]] \mid \ldots \mid l_n[[P_n]]$ is called a *component decomposition* of the system N.

Table 2. rTiMo Structural Congruence

(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)	$l[[P \mid Q]] \equiv l[[P]] \mid l[[Q]]$

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.

Table 3. rTiMo Operational Semantics

(Stop)	$l[[0]] \overset{\lambda}{\not\rightarrow}$ (DStop) $l[[0]] \overset{t}{\rightsquigarrow} l[[0]]$
(DMove)	if $t \geq t'$ then $l[[go^{\Delta t} l' \text{ then } P]] \overset{t'}{\rightsquigarrow} l[[go^{\Delta t - t'} l' \text{ then } P]]$
(Move0)	$l[[go^{\Delta 0} l' \text{ then } P]] \xrightarrow{l \triangleright l'} l'[[P]]$
(Com)	$l[[a^{\Delta t}!\langle v\rangle \text{ then } P \text{ else } Q \mid a^{\Delta t'}?(u) \text{ then } P' \text{ else } Q']] \xrightarrow{\{v/u\}@l} l[[P \mid \{v/u\}P']]$
(DPut)	if $t \geq t' > 0$ then $l[[a^{\Delta t}!\langle v\rangle \text{ then } P \text{ else } Q]] \overset{t'}{\rightsquigarrow} l[[a^{\Delta t - t'}!\langle v\rangle \text{ then } P \text{ else } Q]]$
(Put0)	$l[[a^{\Delta 0}!\langle v\rangle \text{ then } P \text{ else } Q]] \xrightarrow{a!^{\Delta 0}@l} l[[Q]]$
(DGet)	if $t \geq t' > 0$ then $l[[a^{\Delta t}?(u) \text{ then } P \text{ else } Q]] \overset{t'}{\rightsquigarrow} l[[a^{\Delta t - t'}?(u) \text{ then } P \text{ else } Q]]$
(Get0)	$l[[a^{\Delta 0}?(u) \text{ then } P \text{ else } Q]] \xrightarrow{a?^{\Delta 0}@l} l[[Q]]$
(DCall)	if $l[[\{v/x\}P_{id}]] \overset{t}{\rightsquigarrow} l[[P'_{id}]]$ and $id(v) \overset{def}{=} P_{id}$ then $l[[id(v)]] \overset{t}{\rightsquigarrow} l[[P'_{id}]]$
(Call)	if $l[[\{v/x\}P_{id}]] \xrightarrow{id@l} l[[P'_{id}]]$ and $id(v) \overset{def}{=} P_{id}$ then $l[[id(v)]] \xrightarrow{id@l} l[[P'_{id}]]$
(DPar)	if $N_1 \overset{t}{\rightsquigarrow} N'_1$, $N_2 \overset{t}{\rightsquigarrow} N'_2$ and $N_1 \mid N_2 \overset{\lambda}{\not\rightarrow}$ then $N_1 \mid N_2 \overset{t}{\rightsquigarrow} N'_1 \mid N'_2$
(Par)	if $N_1 \xrightarrow{\Lambda_1} N'_1$ and $N_2 \xrightarrow{\Lambda_2} N'_2$ then $N_1 \mid N_2 \xrightarrow{\Lambda_1 \cup \Lambda_2} N'_1 \mid N'_2$
(DEquiv)	if $N \equiv N'$, $N' \overset{t}{\rightsquigarrow} N''$ and $N'' \equiv N'''$ then $N \overset{t}{\rightsquigarrow} N'''$
(Equiv)	if $N \equiv N'$, $N' \xrightarrow{\Lambda} N''$ and $N'' \equiv N'''$ then $N \xrightarrow{\Lambda} N'''$

In rule (Move0), the process $go^{\Delta 0} l'$ then P migrates from location l to location l' and evolves as process P. In rule (Com), a process $a^{\Delta t}!\langle v\rangle$ then P else Q located at location$l$, succeeds in sending a tuple of values v over channel a to process $a^{\Delta t}?(u)$ then P' else Q' also located at l. Both processes continue to execute at location l, the first one as P and the second one as $\{v/u\}P'$. If a communication action has a timer equal to 0, then by using the rule (Put0) for output action or the rule (Get0) for input action, the generic process $a^{\Delta 0} * \text{ then } P \text{ else } Q$ 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 D. For instance, in rule (DPar), $N_1 \mid N_2 \stackrel{\lambda}{\not\rightarrow}$ means that no action λ (i.e, an action labelled by $l' \triangleright l$, $\{v/u\}@l$, $id@l$, $go^{\Delta 0}@l$, $a?^{\Delta 0}@l$ or $a!^{\Delta 0}@l$) can be applied in the system $N_1 \mid N_2$. Negative premises are used to denote the fact that the passing to a new step is performed based on the absence of actions; the use of negative premises does not lead to an inconsistent set of rules.

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

$$N \stackrel{\Lambda}{\longrightarrow} N_1 \stackrel{t}{\rightsquigarrow} N'.$$

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

Proposition 1. *For all systems N, N' and N'', the following statements hold:*

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

The first 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 Modelling Critical Systems by Using rTiMo

The use of rTiMo for specifying critical systems is illustrated by considering a railway bridge controller, a real-time problem concerned with the control of accessing a mobile bridge by several trains according to the rule that the bridge can be accessed only by one train at a time. The system is defined as a number of trains (we use three trains), two railways (each divided into two sections on each side of the bridge), and a mobile bridge that can allow ships to sail on the river below (the bridge is up) or not (the bridge is down). This is a simplified version of the system described in [11]. Since not all the actions can take place simultaneously, their delays are modelled by timers.

The initial system is described in rTiMo by:

$$railway1a[[train1 \mid train3]] \mid railway1b[[0]] \mid railway2a[[train2]] \mid railway2b[[0]] \mid bridge[[operate \mid control_1]],$$

where the processes placed inside locations are as defined below.

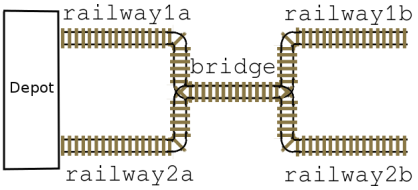

Fig. 1. A railway crossing

In what follows branches that continue with a 0 process are omitted.

Waiting indefinitely on a channel a is abstracted by using the timer $\Delta\infty$.

$$
\begin{aligned}
train1 = {} & go^{\Delta 15} bridge \text{ then } appr^{\Delta 20}!\langle train1, railway1a, railway1b\rangle \\
& \mid stop^{\Delta 25}?(x) \text{ then } move^{\Delta\infty}?(y) \\
& \qquad \text{then } (go^{\Delta 3} bridge \text{ then } go^{\Delta 7} railway1b \text{ then } train1' \\
& \qquad\quad \mid go^{\Delta 10} bridge \text{ then } leave^{\Delta 1}!\langle train1\rangle) \\
& \qquad \text{else } (go^{\Delta 2} bridge \text{ then } go^{\Delta 7} railway1b \text{ then } train1' \\
& \qquad\quad \mid go^{\Delta 9} bridge \text{ then } leave^{\Delta 1}!\langle train1\rangle) \\
train2 = {} & go^{\Delta 11} bridge \text{ then } appr^{\Delta 20}!\langle train2, railway2a, railway2b\rangle \\
& \mid stop^{\Delta 21}?(x) \text{ then } move^{\Delta\infty}?(y) \text{ then } (go^{\Delta 2} bridge \\
& \qquad \text{then } go^{\Delta 6} railway2b \text{ then } train2' \\
& \qquad\quad \mid go^{\Delta 8} bridge \text{ then } leave^{\Delta 1}!\langle train2\rangle) \\
& \qquad \text{else } (go^{\Delta 1} bridge \text{ then } go^{\Delta 6} railway1b \text{ then } train2' \\
& \qquad\quad \mid go^{\Delta 7} bridge \text{ then } leave^{\Delta 1}!\langle train2\rangle) \\
train3 = {} & go^{\Delta 1} bridge \text{ then } appr^{\Delta 20}!\langle train3, railway1a, railway2b\rangle \\
& \mid stop^{\Delta 11}?(x) then \ move^{\Delta\infty}?(y) \\
& \qquad \text{then } (go^{\Delta 1} bridge \text{ then } go^{\Delta 5} railway2b \text{ then } train3' \\
& \qquad\quad \mid go^{\Delta 6} bridge \text{ then } leave^{\Delta 1}!\langle train3\rangle) \\
& \qquad \text{else } (go^{\Delta 0.5} bridge \text{ then } go^{\Delta 5} railway1b \text{ then } train3' \\
& \qquad\quad \mid go^{\Delta 5.5} bridge \text{ then } leave^{\Delta 1}!\langle train3\rangle) \\
operate = {} & down^{\Delta\infty}?(v) \text{ then } up^{\Delta\infty}?(w) \text{ then } operate \\
control_1 = {} & appr^{\Delta\infty}?(x_1, y_1, z_1) \text{ then } (down^{\Delta 1}!\langle x_1\rangle \mid control_0 \mid control_2) \\
control_0 = {} & leave^{\Delta\infty}?(x) \text{ then } (up^{\Delta 1}!\langle x_1\rangle \mid unblock^{\Delta\infty}!\langle x_1\rangle) \\
control_i = {} & appr^{\Delta\infty}?(x_i, y_i, z_i) \\
& \qquad then \ [go^{\Delta 0} y_i \text{ then } stop^{\Delta 1}!\langle x_i\rangle \ \mid go^{\Delta 10i} y_i \text{ then } move^{\Delta 1}!\langle x_i\rangle \\
& \qquad\quad \mid wait^{\Delta 10i}?(x) \text{ then } 0 \text{ else } (down^{\Delta 10}!\langle x_1\rangle \mid control_0) \\
& \qquad\quad \mid unblock^{\Delta 1}?(z) \text{ then } control_{i-1} \text{ else } control_{i+1}]
\end{aligned}
$$

A train movement is abstractly modelled using *go* actions to describe the migrations between the locations of the system (the parts of the railways and the bridge). The synchronization between the controllers operating the bridge and the trains is modelled by communication actions. When a train is approaching, it communicates with the $control_i$ on channel *appr*, announcing the name of the train, the current location and the destination. The timer $\Delta 20$ means that the $control_i$ has to acknowledge in at most 20 units of time that the train is approaching. If the bridge is occupied, the train has to be stopped in 10 units of time from the receiving of the approach message; otherwise the train goes to location *bridge*. When the $control_i$ decides to stop a train, it does it by

synchronizing on channel *stop* at the train location. After a train was stopped, it waits for the synchronization on the channel *move* allowing it to cross the bridge. It can be noticed that if a train is stopped, then it takes a longer period of time to cross the *bridge*.

The *bridge* has to ensure the following safety properties: the *bridge* is down whenever a train is at the crossing, and it prevents trains crossing when another train is engaged in crossing. The controller $control_i$ interacts with the incoming trains, instruct them what to do (e.g., *stop* or *move*) and sends messages to control the operation of the bridge (either *up* or *down*).

In what follows are written some evolution steps for the system described above. For each process are written only the actions to be applied next (e.g., the $train1$ process is represented as $train1 = go^{\Delta 15} bridge \ldots \mid stop^{\Delta 25}?(x) \ldots$). In order to follow easily the evolution, the reductions are performed one after another instead of an entire multiset of reductions; the whole parallel step is delimited by the time steps $\overset{t}{\rightsquigarrow}$. To ease the reading, we bold the actions or the processes that are executed in the next step. We illustrate only a few reductions just to give an idea how the system evolves.

$$
\begin{aligned}
&railway1a[[train1 \mid train3]] \mid railway2a[[train2]] \mid railway1b[[0]] \mid railway2b[[0]] \\
&\quad \mid bridge[[operate \mid control_1]] \\
&\overset{1}{\rightsquigarrow} railway1a[[go^{\Delta 14} bridge \ldots \mid stop^{\Delta 24}?(x) \ldots \\
&\qquad\qquad \mid \mathbf{go^{\Delta 0} bridge} \ldots \mid stop^{\Delta 10}?(x) \ldots]] \\
&\quad \mid railway2a[[go^{\Delta 10} bridge \ldots \mid stop^{\Delta 20}?(x) \ldots]] \\
&\quad \mid railway1b[[0]] \mid railway2b[[0]] \mid bridge[[operate \mid control_1]] \\
&\xrightarrow{railway1a \triangleright bridge} railway1a[[go^{\Delta 14} bridge \ldots \mid stop^{\Delta 24}?(x) \ldots \mid stop^{\Delta 10}?(x) \ldots]] \\
&\quad \mid railway2a[[go^{\Delta 10} bridge \ldots \mid stop^{\Delta 20}?(x) \ldots]] \mid railway1b[[0]] \mid railway2b[[0]] \\
&\quad \mid bridge[[operate \mid \mathbf{control_1} \mid \mathbf{appr^{\Delta 20}!\langle train3, railway1a, railway2b\rangle}]] \\
&\xrightarrow{\{(train3, railway1a, railway2b)/(x_1, y_1, z_1)\}@bridge} \\
&\quad railway1a[[go^{\Delta 14} bridge \ldots \mid stop^{\Delta 24}?(x) \ldots \mid stop^{\Delta 10}?(x) \ldots]] \\
&\quad \mid railway2a[[go^{\Delta 10} bridge \ldots \mid stop^{\Delta 20}?(x) \ldots]] \mid railway1b[[0]] \mid railway2b[[0]] \\
&\quad \mid bridge[[\mathbf{operate} \mid \mathbf{down^{\Delta 1}!\langle train3\rangle}} \\
&\quad \mid \{(train3, railway1a, railway2b)/(x_1, y_1, z_1)\} control_0 \\
&\quad \mid \{(train3, railway1a, railway2b)/(x_1, y_1, z_1)\} control_2]] \\
&\xrightarrow{\{train3/v\}@bridge} railway1a[[go^{\Delta 14} bridge \ldots \mid stop^{\Delta 24}?(x) \ldots \mid stop^{\Delta 10}?(x) \ldots]] \\
&\quad \mid railway2a[[go^{\Delta 10} bridge \ldots \mid stop^{\Delta 20}?(x) \ldots]] \mid railway1b[[0]] \mid railway2b[[0]] \\
&\quad \mid bridge[[up^{\Delta \infty}?(w) \ldots \mid \{(train3, railway1a, railway2b)/(x_1, y_1, z_1)\} control_0 \\
&\qquad\qquad \mid \{(train3, railway1a, railway2b)/(x_1, y_1, z_1)\} control_2]] \\
&\overset{10}{\rightsquigarrow} \ldots
\end{aligned}
$$

An important advantage of using rTiMo to describe time-aware systems is the possibility to verify certain interesting real-time properties such as safety properties (a specified error cannot occur) and bounded liveness properties (configuration reachability within a certain amount of time) by using the model checking capabilities of the software tool Uppaal . This is possible due to the relationship between rTiMo and timed safety automata presented in [1], allowing a natural use of the software tool Uppaal for verification of critical systems described in rTiMo .

4 Real-Time Behavioral Equivalences in rTiMo

Bisimulation is one of the important notion related to concurrent complex systems [15]. We focus here on behavioural equivalences over multiset labelled transition systems; unlike the classical definition in which two systems are equivalent if they match each other's actions, in this paper we consider that two systems are equivalent if they match each other's multiset of actions. Moreover, this could happen in a certain range of time (up to a timeout). An advantage of equivalences defined in this way is that one could aim at obtaining a correspondence between processes that otherwise would not be equivalent (by using already existing equivalences where the order of compared actions has to be the same, at the same moment of time). The multisets of actions could be considered as timely equivalent if they are in a similar interval of time, without imposing a strict moment for each action. Bisimilarity could be also useful when reasoning about behavioural equivalences of processes: given a process, one can check if it is behaving as intended (optimal behaviour) or not (sub-optimal behaviour). Two processes are said to be equivalent if they are able to "simulate" each others' actions, step by step, and continue to be equivalent after each such step [14].

When choosing which equivalence relation to adopt for a given system, one needs to decide what properties should be preserved by the equivalence relation. It is an advantage if the equivalence relations are compositional with respect to the main constructs of the formalism, and so allowing the components to be substituted by equivalent ones without any side-effect.

Definition 1. *A* **timed bisimulation** $\mathcal{R}$ *over* rTiMo *systems using a set Act of actions is a symmetrical binary relation satisfying the conditions:*

- *for all* $(N_1, N_2) \in \mathcal{R}$*, if* $N_1 \xrightarrow{\lambda} N_1'$ *for* $\lambda \in Act$ *and* N_1'*, then* $N_2 \xrightarrow{\lambda} N_2'$ *and* $(N_1', N_2') \in \mathcal{R}$ *for some* N_2'*.*
- *for all* $(N_1, N_2) \in \mathcal{R}$*, if* $N_1 \overset{t}{\rightsquigarrow} N_1'$ *for* $t \in \mathbb{R}_+$ *and* N_1'*, then* $N_2 \overset{t}{\rightsquigarrow} N_2'$ *and* $(N_1', N_2') \in \mathcal{R}$ *for some* N_2'*.*

Two rTiMo *systems are* **timed bisimilar** *iff there is a timed bisimulation relation containing them.*

In a similar way as in timed distributed π-calculus [4], the standard notion of bisimilarity is extended to take into account timed transitions and multisets of actions. For a set A, A^* denotes the set of all multisets over A. For a multiset of actions $\Lambda = \{\lambda_1, \ldots, \lambda_n\} \in A^*$, the sequence $\xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_n}$ is denoted by $\xrightarrow{\Lambda}$.

The **identity relation** over the set $\mathcal{L}$ of located processes is $\mathrm{id} \overset{def}{=} \{(L, L) \mid L \in \mathcal{L}\}$. The **inverse of a relation** $\mathcal{R}$ is $\mathcal{R}^{-1} \overset{def}{=} \{(L_2, L_1) \mid (L_1, L_2) \in \mathcal{R}\}$. The **composition of relations** $\mathcal{R}_1$ and $\mathcal{R}_2$ is $\mathcal{R}_1\mathcal{R}_2 \overset{def}{=} \{(L, L'') \mid \exists L' \in \mathcal{L}$ such that$(L, L') \in \mathcal{R}_1$ and $(L', L'') \in \mathcal{R}_2\}$.

Definition 2. *A binary relation* $\mathcal{R} \subseteq \mathcal{L} \times \mathcal{L}$ *is called a* **timed simulation** *(T simulation) if when* $(l[[P]], l[[Q]]) \in \mathcal{R}$ *and* $\Lambda \in \{id@l, \{v/u\}@l, go^{\Delta 0}@l, a?^{\Delta 0}@l, a!^{\Delta 0}@l\}^*$*:*

- *if* $l[[P]] \xrightarrow{\Lambda} \xrightarrow{l \triangleright l'} l'[[P']] \mid l[[P'']]$ *then* $\exists Q', Q''$ *s.t.* $l[[Q]] \xrightarrow{\Lambda} \xrightarrow{l \triangleright l'} l'[[Q']] \mid l[[Q'']]$, $(l'[[P']], l'[[Q']]) \in \mathcal{R}$ *and* $(l[[P'']], l[[Q'']]) \in \mathcal{R}$;
- *if* $l[[P]] \xrightarrow{\Lambda} \overset{t}{\rightsquigarrow} l[[P']]$ *then* $\exists Q'$ *s.t.* $l[[Q]] \xrightarrow{\Lambda} \overset{t}{\rightsquigarrow} l[[Q']]$ *and* $(l[[P']], l[[Q']]) \in \mathcal{R}$.

If $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are timed simulations, then* $\mathcal{R}$ *is called a* **timed bisimulation** *(T bisimulation).* **Strong timed bisimilarity** *(ST bisimulation)* $\sim$ *is defined by*

$$\sim \overset{def}{=} \{(l[[P]], l[[Q]]) \in \mathcal{L} \times \mathcal{L} \mid \exists T \text{ bisimulation } \mathcal{R} \text{ and } (l[[P]], l[[Q]]) \in \mathcal{R}\}.$$

This definition treats timed transitions as normal transitions, and so it coincides with the original notion of bisimulation over a labelled transition system.

Remark 1. $\sim$ is an equivalence relation, and also the largest ST bisimulation.

Example 1. Inspired by the railway system of Subsection 3, consider the following simplified two located processes:

$$\begin{aligned} L_1 = railway1a[[stop^{\Delta 5}?(x)\ then\ (go^{\Delta 3} bridge\ then\ go^{\Delta 7} railway1b) \\ else\ (go^{\Delta 2} bridge\ then\ go^{\Delta 7} railway1b)]] \\ L_2 = railway1a[[stop^{\Delta 5}?(x)\ then\ (go^{\Delta 2} bridge\ then\ go^{\Delta 6} railway1b) \\ else\ (go^{\Delta 1} bridge\ then\ go^{\Delta 6} railway1b)]] \end{aligned}$$

If the trains reach *bridge* after different numbers of time units, the two located processes are not strong timed bisimilar, i.e., $(L_1 \not\sim L_2)$, because they have different evolutions in time (after 7 units of time).

$$\begin{aligned} & L_1 \overset{5}{\rightsquigarrow} railway1a[[stop^{\Delta 0}?(x)\ then\ (go^{\Delta 3} bridge\ then\ go^{\Delta 7} railway1b) \\ & \qquad else\ (go^{\Delta 2} bridge\ then\ go^{\Delta 7} railway1b)]] && \text{(DPar)} \\ & \xrightarrow{stop?^{\Delta 0}@railway1a} railway1a[[go^{\Delta 3} bridge\ then\ go^{\Delta 7} railway1b]] && \text{(Get0)} \\ & \overset{2}{\rightsquigarrow} railway1a[[go^{\Delta 1} bridge\ then\ go^{\Delta 7} railway1b]] && \text{(DPar)} \\ & \overset{1}{\rightsquigarrow} railway1a[[go^{\Delta 0} bridge\ then\ go^{\Delta 7} railway1b]] && \text{(DPar)} \\ & L_2 \overset{5}{\rightsquigarrow} railway1a[[stop^{\Delta 0}?(x)\ then\ (go^{\Delta 2} bridge\ then\ go^{\Delta 6} railway1b) \\ & \qquad else\ (go^{\Delta 1} bridge\ then\ go^{\Delta 6} railway1b)]] && \text{(DPar)} \\ & \xrightarrow{stop?^{\Delta 0}@railway1a} railway1a[[go^{\Delta 2} bridge\ then\ go^{\Delta 6} railway1b]] && \text{(Get0)} \\ & \overset{2}{\rightsquigarrow} railway1a[[go^{\Delta 0} bridge\ then\ go^{\Delta 6} railway1b]] && \text{(DPar)} \\ & \xrightarrow{railway1a \triangleright bridge} bridge[[go^{\Delta 7} railway1b]] && \text{(Move0)} \end{aligned}$$

Strong timed equivalences require an exact matching between the multisets of transitions of two located processes, for the entire evolution. Sometimes these requirements are too strong. According to [12], there are problems in computer science and artificial intelligence where only the timed behaviour within a given amount of time t is of interest. Sometimes one needs to see if two critical systems have the same behaviour for a predefined period of time and not for their entire evolution (e.g., trains that behave equivalently only between two locations, regardless of what happens for the rest of their evolutions). That is why in what follows the equivalences are restricted to a given time range $[0, t]$, thus defining **time-bounded** equivalences.

Definition 3. *The binary relations* $\mathcal{R}_t \subseteq \mathcal{L} \times \mathcal{L}$, $t \in \mathbb{R}_+$ *over located processes are called* **time-bounded simulations** *(TB simulations) if for* $t \in \mathbb{R}_+$, *whenever* $(l[[P]], l[[Q]]) \in \mathcal{R}_t$ *and* $\Lambda \in \{id@l, \{v/u\}@l, go^{\Delta 0}@l, a?^{\Delta 0}@l, a!^{\Delta 0}@l\}^*$:

- *if* $l[[P]] \xrightarrow{\Lambda} \xrightarrow{l \triangleright l'} l'[[P']] \mid l[[P'']]$ *then* $\exists Q', Q''$ *such that* $l[[Q]] \xrightarrow{\Lambda} \xrightarrow{l \triangleright l'} l'[[Q']] \mid l[[Q'']]$, $(l'[[P']], l'[[Q']]) \in \mathcal{R}_t$ *and* $(l[[P'']], l[[Q'']]) \in \mathcal{R}_t$;
- $\forall t' \leq t, t' \in \mathbb{R}_+$ *if* $l[[P]] \xrightarrow{\Lambda} \overset{t'}{\rightsquigarrow} l[[P']]$ *then* $\exists Q'$ *such that* $l[[Q]] \xrightarrow{\Lambda} \overset{t'}{\rightsquigarrow} l[[Q']]$ *and* $(l[[P']], l[[Q']]) \in \mathcal{R}_{t-t'}$.

If $\mathcal{R}_t$ *and* $\mathcal{R}_t^{-1}$, *with* $t \in \mathbb{R}_+$, *are time-bounded simulations, then* $\mathcal{R}_t$ *is a* **time-bounded bisimulation** *(TB bisimulation).* **Time-bounded bisimilarities** $\simeq_t$ *are defined by*

$\simeq_t \overset{def}{=} \{(l[[P]], l[[Q]]) \in \mathcal{L} \times \mathcal{L} \mid \exists$ *TB bisimulation* $\mathcal{R}_t$ *and* $(l[[P]], l[[Q]]) \in \mathcal{R}_t\}$.
$(l[[P]], l[[Q]]) \in \simeq_t$ *can be written also as* $l[[P]] \simeq_t l[[Q]]$.

Example 2. Consider the two located processes of Example 1. Even if those systems have different definitions, they are time-bounded bisimilar before time unit 7 ($L_1 \simeq_7 L_2$) since they have the same evolutions during this period at location $railway1a$. Hence L_1 and L_2 cannot be identified by timed bisimulation, but this is possible by using time-bounded bisimulation for the time range $[0, 7]$. However, if $t > 7$, then $L_1 \not\simeq_t L_2$.

Time-bounded bisimulation satisfies the following properties showing that an equivalence $\simeq_t$ includes the equivalence $\simeq_u$ for any $u \leq t$. This result is consistent with the continuity of time. This means that if two processes are time-bounded equivalent in a finite time range $[0, t]$, then they are time-bounded equivalent in any finite time range $[0, u]$, $u \leq t$.

Lemma 1. *For any processes* P *and* Q, *location* l, *and any* $u, t \in \mathbb{R}_+$:
If $l[[P]] \simeq_t l[[Q]]$, *then for any* $u \leq t$ *it holds that* $l[[P]] \simeq_u l[[Q]]$.

A useful question to ask about an rTiMo located process is the reachability of a given process within a given amount of time. In what follows l and l' denote the same or different locations:

Definition 4. *Given* $t \in \mathbb{R}_+$ *and* $l[[P]], l'[[Q]] \in \mathcal{L}$, *the* t*-bounded reachability problem asks if there exists a computation leading from* $l[[P]]$ *to* $l'[[Q]]$ *in at most* t *units of time.*

The next lemma states that time-bounded bisimulation is adequate to check t-bounded reachability on arbitrary located processes.

Lemma 2. *If* $l[[P]] \simeq_t l[[Q]]$, *then* $l'[[R]]$ *is reachable from* $l[[P]]$ *in at most* t *units of time iff* $l'[[R]]$ *is reachable from* $l[[Q]]$ *in at most* t *units of time.*

Using the TB bisimulations $\simeq_t$, a specific relation of bisimilarity is defined, called strong time-bounded bisimilarity, satisfying Proposition 2.

Definition 5. Strong time-bounded bisimilarity *(STB bisimulation), denoted* $\simeq$, *is defined by:*

$\simeq = \{(l[[P]], l[[Q]]) \in \mathcal{L} \times \mathcal{L} \mid \exists t \in \mathbb{R}_+$ *and a TB bisim.* $\simeq_t$ s.t. $(l[[P]], l[[Q]]) \in \simeq_t\}$.

Proposition 2. *The following statements hold:*

1. $\simeq$ *is a TB bisimulation;*
2. $\simeq$ *is closed to identity, inverse, composition and union;*
3. $\simeq$ *is the largest TB bisimulation;*
4. $\simeq$ *is an equivalence.*

Using the fact that $\simeq$ is an equivalence can be used to partition a state space into equivalence classes such that states in the same class are observationally equivalent with respect to the system's behaviour. This leads to a reduction of the state space prior to model checking.

4.1 Strong Open Time-Bounded Equivalences

Bisimulation as a congruence is a desirable feature for any (real-time) language for critical systems because it can be used in checking compositionally whether two critical systems are behaving similarly. This means that the specifications related by a bisimulation relation $\mathcal{R}$ can be used interchangeably as parts of a larger process without affecting the overall behaviour of the latter (as depicted in Figure 2). In this paper behavioural equivalences are based on the observable transitions of processes, rather than on their states (as done in timed automata and time/timed Petri nets).

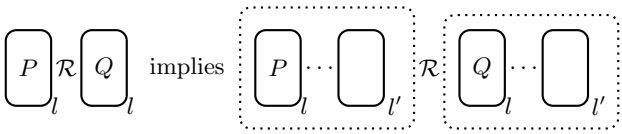

Fig. 2. Interchangeably equivalent parts of a larger system

In this section we define such a relation for rTIMO, a relation inspired by the *open bisimilarity* [14]. In this kind of bisimilarity, all names that occur in a system are potentially replaceable (all free names are treated as variables). The newly defined open bisimilarity is necessary since, according to the following example, the TB equivalence is not closed under arbitrary substitutions.

Example 3. Consider that *train*1 from Subsection 3 is located in the depot (has not entered yet either *railway*1*a* or *railway*2*a* of the two railways to which the depot is connected) and wants to reach *railway*2*b*. In order to reach the destination it is necessary to use either *railway*1*a* or *railway*2*a*. It has to take a decision on which of the these two railways its journey starts. In order to decide on this aspect it sends a *query* to the depot in order to receive an answer that can help it in making this decision. There are two situations: the *train*1 decides either to use the received information (process P_1) or not (process P_2).

$P_1 = newrail^{\Delta 0}?(railway1a)$ then $query^{\Delta 2}!\langle railway1a\rangle$
$\quad | \; query^{\Delta 5}?(u)$ then $(go^{\Delta 3}u$ then $train1)$.
$P_2 = newrail^{\Delta 0}?(railway1a)$ then $query^{\Delta 2}!\langle railway1a\rangle$
$\quad | \; query^{\Delta 5}?(u)$ then $(go^{\Delta 3}railway1a$ then $train1)$.

For any $t \in \mathbb{R}_+$, it holds that $station[[P_1]] \simeq_t station[[P_2]]$ because

$$station[[P_1]] \xrightarrow{newrail?^{\Delta 0}@station} \xrightarrow{\{railway1a/u\}@station} station[[go^{\Delta 3}railway1a \text{ then } train1]], \text{ and}$$

$$station[[P_2]] \xrightarrow{newrail?^{\Delta 0}@station} \xrightarrow{\{railway1a/u\}@station} station[[go^{\Delta 3}railway1a \text{ then } train1]].$$

After communicating on channel *query*, results the same process and thus $station[[P_1]] \simeq_t station[[P_2]]$. However, if in these two processes, P_1 and P_2, the free names are rewritten by the substitution $\sigma = \{railway2a/railway1a\}$ (meaning that name $railway2a$ is communicated instead of $railway1a$), then

$$station[[P_1\sigma]] \xrightarrow{newrail?^{\Delta 0}@station} \xrightarrow{\{railway1a/u\}@station} station[[go^{\Delta 3}railway1a \text{ then } train1]], \text{ and}$$

$$station[[P_2\sigma]] \xrightarrow{newrail?^{\Delta 0}@station} \xrightarrow{\{railway1a/u\}@station} station[[go^{\Delta 3}railway2a \text{ then } train1]].$$

These processes have different behaviours, and so

$$station[[P_1]] \not\simeq_t station[[P_2]].$$

Following the style presented in [15], we define the following bisimilarity that becomes a congruence by closing the bisimilarity under arbitrary substitutions.

Definition 6. *The binary relations $\mathcal{R}_t^o \subseteq \mathcal{L} \times \mathcal{L}$, $t \in \mathbb{R}_+$ over located processes are called* **open time-bounded simulations** *(OTB simulations) if for $t \in \mathbb{R}_+$, whenever $(l[[P]], l[[Q]]) \in \mathcal{R}_t^o$, then for any substitution σ and $\Lambda \in \{id@l, \{v/u\}@l, go^{\Delta 0}@l, a?^{\Delta 0}@l, a!^{\Delta 0}@l\}^*$ it holds:*

- *if $l[[P\sigma]] \xrightarrow{\Lambda} \xrightarrow{l \triangleright l'} l'[[P']] \mid l[[P'']]$ then $\exists Q', Q''$ such that $l[[Q\sigma]] \xrightarrow{\Lambda} \xrightarrow{l \triangleright l'} l'[[Q']] \mid l[[Q'']]$, $(l'[[P']], l'[[Q']]) \in \mathcal{R}_t^o$ and $(l[[P'']], l[[Q'']]) \in \mathcal{R}_t^o$;*
- *$\forall t' \leq t$, $t' \in \mathbb{R}_+$ if $l[[P\sigma]] \xrightarrow{\Lambda} \overset{t'}{\rightsquigarrow} l[[P']]$ then $\exists Q'$ such that $l[[Q\sigma]] \xrightarrow{\Lambda} \overset{t'}{\rightsquigarrow} l[[Q']]$ and $(l[[P']], l[[Q']]) \in \mathcal{R}_{t-t'}^o$.*

If $\mathcal{R}_t^o$ and $(\mathcal{R}_t^o)^{-1}$ are open time-bounded simulations for $t \in \mathbb{R}_+$, then $\mathcal{R}_t^o$ are called **open time-bounded bisimulations** *(OTB bisimulations).* **Open time-bounded bisimilarities** *$\simeq_t^o$ are defined by*

$$\simeq_t^o \overset{def}{=} \{(l[[P]], l[[Q]]) \in \mathcal{L} \times \mathcal{L} \mid \exists \, OTB \text{ bisimulation } \mathcal{R}_t^o \text{ and } (l[[P]], l[[Q]]) \in \mathcal{R}_t^o\}.$$

$(l[[P]], l[[Q]]) \in \simeq_t^o$ can be written as $l[[P]] \simeq_t^o l[[Q]]$.

The intuition is that two located processes are equivalent whenever all possible instantiations (substitutions of their free names) have matching transitions.

The next result is consistent with the continuity of time: an open time-bounded equivalence $\simeq_0^t$ includes the open time-bounded equivalence $\simeq_0^u$, for any $u \leq t$. This means that if two processes are open time-bounded equivalent in a finite time range $[0, t]$, then they are open time-bounded equivalent in any finite time range $[0, u]$, $u \leq t$.

Lemma 3. *For any processes P and Q, location l, and any $u, t \in \mathbb{R}_+$: if $l[[P]] \simeq_t^o l[[Q]]$, then for any $u \leq t$ it holds that $l[[P]] \simeq_u^o l[[Q]]$.*

Using the OTB bisimulations $\simeq_t^o$, a specific relation of bisimilarity is defined, called strong open time-bounded bisimilarity; this relation satisfies the statements of Proposition 3.

Definition 7. Strong open time-bounded bisimilarity *(SOTB bisimulation), denoted $\simeq^o$, is defined by:*

$$\simeq^o = \{(l[[P]], l[[Q]]) \in \mathcal{L} \times \mathcal{L} \mid \exists t \in \mathbb{R}_+ \text{ and a OTB bisim. } \simeq_t^o \text{ s.t. } (l[[P]], l[[Q]]) \in \simeq_t^o\}.$$

The following results present some properties of the SOTB equivalences.

Proposition 3. *The following statements hold:*

1. *$\simeq^o$ is a OTB bisimulation;*
2. *$\simeq^o$ is closed to identity, inverse, composition and union;*
3. *$\simeq^o$ is the largest OTB bisimulation;*
4. *$\simeq^o$ is an equivalence.*

Definition 8. *A binary relation $\mathcal{R}$ is said to be* **closed under substitutions** *if whenever $(l[[P]], l[[Q]]) \in \mathcal{R}$, then $(l[[P\sigma]], l[[Q\sigma]]) \in \mathcal{R}$ for any substitution σ. Formally,*

$$\text{clos}(\mathcal{R}) \stackrel{def}{=} \{(l[[P\sigma]], l[[Q\sigma]]) \mid (l[[P]], l[[Q]]) \in \mathcal{R} \text{ and } \sigma \text{ is an arbitrary substitution}\}.$$

The connections between $\simeq$ and $\simeq^o$ are illustrated in the next result. The second item states that $\simeq^o$ is included in $\simeq$, namely if $l[[P]] \simeq^o l[[Q]]$ implies that $l[[P]] \simeq l[[Q]]$.

Lemma 4. *1. If $\simeq$ is closed under substitution, then $\simeq = \simeq^o$.*
2. $\simeq^o \subseteq \simeq$.

The following result states that the SOTB equivalence $\simeq^o$ is preserved by migration, communication and parallel composition.

Lemma 5. *For $P, P', Q, Q' \in \mathcal{P}$ and $l, l' \in Loc$, if $l[[P]] \simeq^o l[[P']]$ then*

1. *$l[[P \mid Q]] \simeq^o l[[P' \mid Q]]$;*
2. *$l[[a^{\Delta t'}?(u)$ then P else $Q]] \simeq^o l[[a^{\Delta t'}?(u)$ then P' else $Q]]$;*
3. *$l[[go^{\Delta t'} l'$ then $P]] \simeq^o l[[go^{\Delta t'} l'$ then $P']]$.*

As a consequence of Lemmas 3 and 5, the main result of the paper is obtained.

Theorem 1. *$\simeq^o$ is a congruence.*

When choosing which bisimulation to adopt in certain situation one needs to decide what kind of properties should be preserved by the equivalence relation. If the bisimulation is not a congruence then the bisimilar systems can still be distinguished by putting them in appropriate contexts. On the other hand, if the bisimulation is a congruence, this means that the systems that are related by a congruence relation, e.g., $\simeq^o$ in our case, can be used interchangeably as parts of a larger system without affecting the overall behaviour of the latter (as depicted in Figure 2). For this reason, usually one needs to ensure that he defines equivalences that are in fact congruences. In this way theories can be constructed that support modular description and verification of critical systems. Thus, it should be possible to use the congruence relation $\simeq^o$ in computer simulations and model checkers for real-time systems with timed migration and communication.

5 Conclusion and Related Work

Several proposals for real-time modelling and verification have been presented in the literature (e.g., [16]). A comprehensive overview of the development of an algebraic theory of processes with time is given in [3]. In this paper we used a prototyping language rTIMO for describing real-time critical systems. It emphasizes the essential aspects, and is different from all these previous approaches since it encompasses specific features as timeouts, explicit locations, timed migration and communication. Starting from a first version of TIMO proposed in [6], several variants were developed during the last years in order to model various complex systems; we mention the access permissions given by a type system in perTIMO [7]. TIMO is a simpler version of timed distributed π-calculus [8]. Inspired by TIMO , a flexible software platform was introduced in [5] to support the specification of agents allowing timed migration in a distributed environment. Interesting properties as bounded liveness and optimal reachability are presented in [2]. A verification tool called TIMO@PAT is presented in [9]; it was developed by using an extensible platform for model checkers called PAT.

In this paper rTIMO is used for comparing in a formal way the behaviours of critical systems. In particular, we have presented an example of applying rTIMO to the distributed railway bridge system, illustrating that rTIMO provides a natural framework for modelling and reasoning about critical systems with timed migration and concurrency given by interaction/communication. This leads to a compositional approach of verifying concurrent critical systems, in opposition to the noncompositional approach provided by inductive assertion method and Hoare logic.

Behavioural equivalences are useful to define some observational criteria that processes should fulfil. Several behavioural equivalences over $tD\pi$ and TIMO are studied in [4]. In practice, even though several processes can be valid solutions to a given problem, some processes may be preferable to others. For example, a faster or less resource consuming process is often preferred to one that is slower or demanding more resources, respectively. In fact, there are many ways to evaluate processes. An important goal of defining bisimulations is to obtain refinements

and equivalence relations that can reduce state space to their equivalence classes, in order to facilitate a more efficient (automated) verification.

We defined two bisimulations ($\simeq$ and $\simeq^o$) over real-time distributed processes, and illustrated them by using a distributed railway control system involving a mobile bridge and several trains. The behavioural equivalences are established in terms of relative time (timeouts) and locations, and are also used to distinguish between optimal and sub-optimal behaviours. We prove that $\simeq^o$ is a congruence, allowing a compositional reasoning of complex real-time systems in terms of their observable parallel behaviours. The first equivalence ($\simeq$) resembles the finite-horizon bisimulation defined over time-inhomogeneous Markov chains [10], in the sense that they also consider a threshold in time when comparing two systems.

The strong bisimulations studied in the paper are useful but their usage is somehow limited in the sense that at each moment either the time elapse or the multiset of actions should coincide. A weaker version of these bisimulations could be defined, having a more practical use in real problems: e.g., to distinguish between trains having the same route, but different moving time depending on the type of the train (e.g., InterRegio or InterCity). Such weak bisimulations in rTiMo and verification of realistic scenarios with a powerful model-checker like Uppaal [13] represent a future work. The capabilities of Uppaal allow verification of various properties: reachability of desired configurations (e.g., several mobile elements being close to each other at some time instance), the fact that the system does not block, and whether an error occurs (e.g., two trains collide).

Acknowledgements. The work was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0919.

References

1. Aman, B., Ciobanu, G.: Real-Time Migration Properties of rTiMo Verified in Uppaal. In: Hierons, R.M., Merayo, M.G., Bravetti, M. (eds.) SEFM 2013. LNCS, vol. 8137, pp. 31–45. Springer, Heidelberg (2013)
2. Aman, B., Ciobanu, G., Koutny, M.: Behavioural Equivalences over Migrating Processes with Timers. In: Giese, H., Rosu, G. (eds.) FORTE 2012 and FMOODS 2012. LNCS, vol. 7273, pp. 52–66. Springer, Heidelberg (2012)
3. Baeten, J.C.M., Middelburg, C.A.: Process Algebra with Timing. In: Monographs in Theoretical Computer Science, An EATCS Series. Springer, Berlin (2002)
4. Ciobanu, G.: Behaviour Equivalences in Timed Distributed π-Calculus. In: Wirsing, M., Banâtre, J.-P., Hölzl, M., Rauschmayer, A. (eds.) Software Intensive Systems. LNCS, vol. 5380, pp. 190–208. Springer, Heidelberg (2008)
5. Ciobanu, G., Juravle, C.: Flexible Software Architecture and Language for Mobile Agents. Concurrency and Computation: Practice and Experience 24, 559–571 (2012)
6. 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)

7. 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)
8. Ciobanu, G., Prisacariu, C.: Timers for Distributed Systems. Electronic Notes in Theoretic Computer Science 164(3), 81–99 (2006)
9. Ciobanu, G., Zheng, M.: Automatic Analysis of TiMo Systems in PAT. In: Proc. 18th International Conference on Engineering of Complex Computer Systems, pp. 121–124. IEEE Computer Society (2013)
10. Han, T., Katoen, J.-P., Mereacre, A.: Compositional Modeling and Minimization of Time-Inhomogeneous Markov Chains. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 244–258. Springer, Heidelberg (2008)
11. Heitmeyer, C., Lynch, N.: The Generalized Railroad Crossing: A Case Study in Formal Verification of Real-Time Systems. In: Proc. of IEEE Real-Time Systems Symposium, pp. 120–131 (1994)
12. Kamide, N.: Bounded Linear-Time Temporal Logic: A Proof-Theoretic Investigation. Annals of Pure and Applied Logic 163, 439–466 (2012)
13. Larsen, K.G., Petterson, P., Yi, W.: Uppaal in a Nutshell. International Journal on Software Tools for Technology Transfer 1(2), 134–152 (1997)
14. Milner, R.: Communicating and Mobile Systems: the π-calculus. Cambridge University Press (1999)
15. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press, New York (2011)
16. Yi, W., Pettersson, P., Daniels, M.: Automatic Verification of Real-time Communicating Systems by Constraint-Solving. In: International Conference on Formal Description Techniques, pp. 223–238 (1994)