Timed π-Calculus

Neda Saeedloei(✉) and Gopal Gupta

University of Texas at Dallas, Dallas, TX, USA
{neda.saeedloei,gupta}@utdallas.edu

Abstract. We extend π-calculus with real-time by adding clocks and assigning time-stamps to actions. The resulting formalism, timed π-calculus, provides a simple and novel way to annotate transition rules of π-calculus with timing constraints. Timed π-calculus is an expressive way of describing mobile, concurrent, real-time systems in which the behavior of systems is modeled by finite or infinite sequences of timed events. We develop an operational semantics as well as a notion of timed bisimilarity for the proposed language. We present the properties of timed bisimilarity; in particular, expansion theorem for real-time, concurrent, mobile processes is investigated.

1 Introduction

The π-calculus was introduced by Milner et al. [12] with the aim of modeling concurrent/mobile processes. The π-calculus provides a conceptual framework for describing systems whose components interact with each other. It contains an algebraic language for describing processes in terms of the communication actions they can perform. Theoretically, the π-calculus can model mobility, concurrency and message exchange between processes as well as infinite computation (through the infinite replication operator '!').

In many cases, processes run on controllers that control physical devices; therefore, they have to deal with physical quantities such as time, distance, pressure, acceleration, etc. Examples include communicating controller systems in cars (Anti-lock Brake System, Cruise Controllers, Collision Avoidance, etc.), automated manufacturing, smart homes, etc. Properties of such systems, which are termed cyber-physical systems (CPS) [6,9], cannot be fully expressed within π-calculus. In a real-time/cyber-physical system the correctness of the system's behavior depends not only on the tasks that the system is designed to perform, but also on the time instants at which these tasks are performed. While π-calculus can handle mobility and concurrency, it is not equipped to model real-time systems or CPS and support reasoning about their behavior related to time and other physical quantities. We extend π-calculus with real time so that these systems can be modeled and reasoned about.

Several extensions of π-calculus with time have been proposed [2,4,5,10]; all these approaches discretize time rather than represent it faithfully as a continuous quantity. Discretizing means that time is represented through finite time intervals. As a result, *infinitesimally small time intervals cannot be represented*

M. Abadi and A. Lluch Lafuente (Eds.): TGC 2013, LNCS 8358, pp. 119–135, 2014.
DOI: 10.1007/978-3-319-05119-2_8,

or reasoned about in these approaches. In practical real-time systems, e.g., a nuclear reactor, two or more events *can* occur within an infinitesimally small interval. Discretizing time can miss the modeling of such behavior which may be wholly contained within this infinitesimally small interval. In our approach for extending π-calculus with time, time is faithfully modeled as a continuous quantity. The most notable work on extending π-calculus with *real time* is the work of Chen [3]; however, the replication operator of the original π-calculus is not considered in this work. Therefore, it is unable to model infinite processes. In our approach the infinite behavior of timed precesses is modeled through the infinite replication operator '!' as in original π-calculus.

We consider the extension of π-calculus with continuous time by adding finitely many real-valued clocks and assigning time-stamps to actions. The resulting formalism can be used for describing concurrent, mobile, real-time systems and CPS and reasoning about their behaviors.[1] In contrast to other extensions, in our work the notion of time and clocks is adopted directly from the well-understood formalism of timed automata [1]. For simplicity, the behavior of a real-time system is understood as a sequence (finite or infinite) of timed events, not states. The times of events are real numbers, which increase monotonically without bound.

2 Timed π-Calculus

2.1 Design Decisions

We define our timed π-calculus as an extension of the original π-calculus [12] with (local) clocks, clock operations and time-stamps. As in π-calculus, timed π-calculus processes use names (including clock names) to interact, and pass names to one another. These processes are identical to processes in π-calculus except that they have access to clocks which they can manipulate.

We assume an infinite set $\mathcal{N}$ of names (channel names and names passing through channels), an infinite set Γ of clock names (disjoint from $\mathcal{N}$) and an infinite set Θ of variables representing time-stamps (disjoint from $\mathcal{N}$ and Γ). When a process outputs a name through a channel, it also sends the time-stamp of the name and the clock that is used to generate the time-stamp. Inspired by the notion of name transmission in π-calculus, we can treat time-stamps and clocks just as other names and transmit them through/with channels. Just as channel transmission results in dynamic configuration of processes, clock and time-stamp transmission can result in dynamic temporal behavior of processes. Thus, messages are represented by triples of the form $\langle m, t_m, c\rangle$, where m is a name in $\mathcal{N}$, t_m is the time-stamp on m, and c is the clock that is used to generate t_m. It is important for the process to send its clock that is used to generate the time-stamp of the name, because the time-stamp of the incoming

[1] While we only focus on extending the π-calculus with continuous time, our method serves as a model for extending the π-calculus with other continuous quantities. An instance of this, though not in the context of π-calculus, can be found in [16].

name in conjunction with the clock received is used by the receiving process to reason about timing requirements of the system as well as *channel delays*.

In our timed π-calculus all the clocks are local clocks; however, their scope is changed as they are sent among processes. This will become clear when we explain how clock passing is performed in Sect. 2.5. Keeping the clocks local results in a considerably simpler design of the timed π-calculus without sacrificing its practical applicability. Note that all the clocks advance at the same rate. A clock can be set to zero simultaneously with any transition (transitions are defined formally in Sect. 2.4). At any instant, the *reading of a clock* is equal to the time that has elapsed since the last time the clock was reset. Following the semantics of timed automata [1], we only consider non-Zeno behaviors, that is, only a finite number of transitions can happen within a finite amount of time.

2.2 Clock Operations and Clock Interpretations

We consider two types of clock operations: resetting a clock and checking satisfaction of a clock constraint. Resetting a clock is used to remember the time at which a particular action in the system has taken place. Clock resets are represented by γ in the syntax of timed π-calculus. Clock constraints, denoted by δ, indicate timing constraints between actions that occur in the system. Note that if δ contains more than one constraint, then the conjunction of all constraints must be considered. δ and γ are defined by the following syntactic rules, in which c and $c_i, 1 \leq i \leq n$, are clock names, r is a constant in $\mathbb{R}_{\geq 0}$, t is a time-stamp and $\sim \in \{<, >, \leq, \geq, =\}$. ϵ represents an empty clock constraint or clock reset.

$$\delta ::= (c \sim r)\delta \mid (c - t \sim r)\delta \mid (t - c \sim r)\delta \mid \epsilon$$
$$\gamma ::= (c_1 := 0) \ldots (c_n := 0) \mid \epsilon$$

There are two ways to measure the passing of time while checking for a clock constraint. It can be measured and reasoned about against (i) the last time a clock was reset: e.g., a constraint $(c < 2)$ on sending m indicates that m must be sent out within two units of time since the clock c was reset, or (ii) the last time a clock c was reset in conjunction with a time-stamp t of a name. Note that in this case, the time-stamp t must be generated by clock c. For instance, suppose that a process P sends two consecutive names that are two units of time apart; if the time-stamp of the first name, generated by clock c is t_1, then the expression $c - t_1 = 2$ can be used to express this constraint.

For a process P, we define $c(P)$ to be the set of clock names in P. For every two processes P and Q we assume $c(P) \cap c(Q) = \emptyset$, initially. This property is also maintained all the time as transitions take place. A *clock interpretation* I for a set Γ of clocks is a mapping from Γ to $\mathbb{R}_{\geq 0}$. It assigns a real value to each clock in Γ. A clock interpretation I for Γ satisfies a clock constraint δ over Γ iff the expression obtained by applying I to δ evaluates to true. For $t \in \mathbb{R}_{\geq 0}$, $I + t$ denotes the clock interpretation which maps every clock c to the value $I(c) + t$. For $\gamma \subseteq \Gamma$, $[\gamma \mapsto t]I$ denotes the clock interpretation for Γ which assigns t to each $c \in \gamma$, and agrees with I over the rest of the clocks.

2.3 Syntax

The set of timed π-calculus processes is defined by the following syntactic rules in which, P, P', M and M' range over processes, x, y and z range over names in $\mathcal{N}$, c and d range over clock names in Γ, and t_y represents a time-stamp.

$$M ::= \delta\gamma\bar{x}\langle y, t_y, c\rangle.P \mid \delta\gamma x(\langle y, t_y, c\rangle).P \mid \delta\gamma\tau.P \mid 0 \mid M + M'$$
$$P ::= M \mid (P \mid P') \mid !P \mid (z)\ P \mid [x = y]\,P \mid [c = d]\,P$$

The expression $\delta\gamma\bar{x}\langle y, t_y, c\rangle.P$ represents a process that is capable of outputting name y on channel x. This process generates a time-stamp t_y using clock c and sends t_y and c along with y via the channel x, and evolves to P. The time-stamp t_y is the reading of clock c at the time of transition. *The assignment of a time-stamp to y and sending y is an atomic operation.* The clock constraint δ must be satisfied by the current value of clocks at the time of transition. γ specifies the clocks to be reset with this transition.

Example 1. The process $P = (c < 2)\bar{x}\langle y, t_y, c\rangle.P'$ is capable of sending name y on channel x within two units of time since clock c was last reset. Note that the time-stamp of y is the reading of clock c when the output takes place. Since the output can happen only within two units of time since c was last reset, then time-stamp t_y is a positive real number less than two ($t_y < 2$). t_y and c are both sent along with y through channel x.

The expression $\delta\gamma x(\langle y, t_y, c\rangle).P$ stands for a process which is waiting for a message on channel x. When a message arrives, the process will behave like $P\{z/y, t_z/t_y, d/c\}$ (substitution is formally defined in Definition 2) where z is the name received; t_z is the time-stamp of z; and d is the clock of the sending process that is used to generate t_z. The time-stamp t_z must satisfy the clock constraint expressed by δ; γ specifies the clocks to be reset with the transition.

Example 2. Assume process $Q = (e > 5)(d - t_z \leq 3)x(\langle z, t_z, d\rangle).Q'$ is the receiving process in Example 1. The received name, along with its time-stamp and the accompanying clock will be substituted for z, t_z and d, respectively. After substitution takes place, the constraint $c - t_y \leq 3$ specifies how long the received name was on transit. Any delay greater than three is not acceptable and cause the input action to not take place. Note that e is another local clock of Q. Both constraints $e > 5$ and $(d - t_z \leq 3)$ must be satisfied by the current value of clocks for the input action to take place.

Note that time-stamps are put on names only by the sending processes, that is no time-stamps are assigned to received names upon arrival. The value of a time-stamp generated by process P on sending name y, is the value of P's local clock c at the time of output. This value is generated such that it satisfies the clock constraint corresponding to the output action. Note that in Example 2, t_z gets bound to the time-stamp of the incoming name, as we do not assign time-stamps to the received names.

The expression $\delta\gamma\tau.P$ stands for a process that takes an internal action and evolves to P, and in doing so resets the clocks specified by γ, if the clock constraint δ is satisfied.

In each of three processes explained above, if the clock constraint δ is not satisfied by the value of clocks at the time of transition, then, the process becomes inactive. An inactive process, represented by 0, is a process that does nothing.

The operators + and | are used for nondeterministic *choice* and *composition* of processes, just as in π-calculus [12]. The *replication* $!P$, represents an infinite composition $P \mid P \mid \ldots$, just as in π-calculus. The *restriction* $(z)P, z \in \mathcal{N}$, behaves as P with z local to P. Therefore, z cannot be used as a channel over which to communicate with other processes or the environment. $[x = y]\, P, x, y \in \mathcal{N} \cup \Gamma$, evolves to P if x and y are the same name; otherwise, it becomes inactive.

Example 3. The timed π-calculus expression $x(\langle m, t_m, c\rangle).(c - t_m \leq 5)\bar{y}\langle n, t_n, c\rangle$ represents a process that is waiting for a message on channel x. The process upon receiving a name m with time-stamp t_m and its accompanying clock c on channel x, sends a name n with time-stamp t_n on channel y with the delay of at most 5 units of time since the time-stamp of m. The process will use the clock c to choose a time t_n on c such that $c - t_m \leq 5$.

In a process of the form $\delta\gamma x(\langle y, t, c\rangle).P$ the occurrences of y, t and c are binding occurrences, and the scope of the occurrences is P. In $(n)P, n \in \mathcal{N}$ the occurrence of n is a binding occurrence, and the scope of the occurrence is P.

Definition 1. *An occurrence of a (non-clock) name n in a process is* free *if it does not lie within the scope of a binding occurrence of n. An occurrence of a (non-clock) name in a process is* bound *if it is not free. All occurrences of a clock c in a process P are bound. The set of bound names of P, $bn(P)$, contains all names which occur bound in P. The set of names occurring free in P is denoted $fn(P)$. We write $n(P)$ for the set $fn(P) \cup bn(P)$ of names of P.*

Intuitively, the free (non-clock) names of a process, represent its (public) links to other processes. For instance, if processes P and Q share the same free name x, then, the channel x is shared between these two processes.

Example 4. Let $P = x(\langle y, t, c\rangle).0$ and $Q = (d > 1)(d < 5)\bar{x}\langle z, t', d\rangle.0$. Then, $fn(P) = \{x\}, bn(P) = \{y, t, c\}$, $fn(Q) = \{x, z, t'\}$, and $bn(Q) = \{d\}$. x is a channel that is shared between P and Q. This behavior can be represented for example in the parallel composition of P and Q: $(P \mid Q)$.

Definition 2. *[12] A substitution is a function θ from a set of names $\mathcal{N}$ to $\mathcal{N}$. If $x_i\theta = y_i$ for all i with $1 \leq i \leq n$ (and $x\theta = x$ for all other names x), we write $\{y_1/x_1, \ldots, y_n/x_n\}$ for θ.*

The effect of applying a substitution θ to a process P is to replace each free occurrence of each name x in P by $x\theta$, with change of bound names to avoid name capture (to preserve the distinction of bound names from the free names). Substitution for time-stamps can be defined similarly.

Definition 3. *A clock substitution is a function θ_c from a set of clock names Γ to Γ. If $c_i\theta_c = d_i$ for all i with $1 \leq i \leq n$ (and $c\theta_c = c$ for all other clock names c), we write $\{d_1/c_1, \ldots, d_n/c_n\}$ for θ_c.*

The effect of applying a substitution θ_c to a process P, $P\theta_c$, is to replace all occurrences of each clock name c in P by $c\theta_c$.

Definition 4. *Given a clock c, the function θ_f creates a fresh copy, f, of c (f does not appear in any process) and updates the interpretation with $I(f) = I(c)$. The application of θ_f to c is represented by $c\theta_f$.*

Definition 5. *A clock renaming θ_r is a clock substitution $\{f_1/c_1, \ldots, f_n/c_n\}$ in which $f_i = c_i\theta_f, 1 \leq i \leq n$.*

The effect of applying a clock renaming $\theta_r = \{f_1/c_1, \ldots, f_n/c_n\}$ to process P, $P\theta_r$, is to replace all occurrences of each name c in P by $c\theta_r$.

2.4 Operational Semantics

First, we define actions by the following syntactic rule:

$$\alpha ::= \bar{x}\langle y, t, c\rangle \mid \bar{x}\langle (y), t, c\rangle \mid x(\langle y, t, c\rangle) \mid \tau$$

The first two actions are the *bound output actions*. Bound output actions are used to carry names out of their scope. $\bar{x}\langle y, t, c\rangle$ is used for sending a name y, time-stamp of y, t, and the (local) clock that is used to generate t, via channel x. The process that gives rise to this action can be of the form $\bar{x}\langle y, t, d\rangle.P, c = d\theta_f$. In this action x, y and t are free and c is bound; c is the fresh copy of d. The expression $\bar{x}\langle (y), t, c\rangle$ is used by a process for sending its private name y (y is bound in the process) and its (local) clock c. The process that gives rise to this action can be of the form: $(y)\bar{x}\langle y, t, d\rangle.P, c = d\theta_f$. In this action x and t are free, while y and c are bound[2]; c is a fresh copy of d.

As we mentioned before all the clocks in the calculus are local clocks: the clocks of a process P are accessible only by P. However, the scope of the clocks is extended as they are sent to other processes. If d is sent to process Q by P, then both P and Q will have access to d. As a result, they both can reset d as part of their future transitions. To prevent processes interfering with each other by resetting a shared clock, P must create a fresh copy of d, let us call it c, and send c to Q. This is the reason of creating fresh copies of clocks in both output actions.

The third action is the *input action* $x(\langle y, t, c\rangle)$. This action is used for receiving any name z with its time-stamp t_z, and a clock d via x. y, t and c are place holders in the receiving process for values that will be received as inputs. In this action x is free, while y, t and c are bound names.

[2] Since all the clocks are local clocks and all clock names are bound, we do not use parenthesis as we do for regular names to distinguish them from free names.

The last action is the *silent action* τ, which is used to express performing an internal action. Silent actions can naturally arise from processes of the form $\tau.P$, or from communications within a process (e.g., rule COM in Table 1).

We use $fn(\alpha)$ for set of free names of α, $bn(\alpha)$ for set of bound names of α, and $n(\alpha)$ for the union of $fn(\alpha)$ and $bn(\alpha)$. Note that $fn(\tau) = \emptyset$ and $bn(\tau) = \emptyset$.

A transition in timed π-calculus is of the form $P \xrightarrow{\langle \delta,\alpha,\gamma \rangle} P'$. This transition is understood as follows: if δ is satisfied by the current values of clocks, P evolves into P', and in doing so performs the action α and resets the clocks specified by

Table 1. Timed π-calculus transition rules

$$\text{IN}\ \frac{}{\delta\gamma x(\langle z,t,c\rangle).P \xrightarrow{\langle \delta\{t'/t,d/c\},x(\langle y,t',d\rangle),\gamma\{d/c\}\rangle} P\{y/z,t'/t,d/c\}}\ y \notin fn((z)P)$$

$$\text{OUT}\ \frac{}{\delta\gamma\bar{x}\langle y,t,c\rangle.P \xrightarrow{\langle \delta,\bar{x}\langle y,t,d\rangle,\gamma\rangle} P}\ d = c\theta_f \qquad \text{TAU}\ \frac{}{\delta\gamma\tau.P \xrightarrow{\langle \delta,\tau,\gamma\rangle} P}$$

$$\text{PAR}\ \frac{P \xrightarrow{\langle \delta,\alpha,\gamma\rangle} P'}{(P \mid Q) \xrightarrow{\langle \delta,\alpha,\gamma\rangle} (P' \mid Q)}\ bn(\alpha) \cap fn(Q) = \emptyset \qquad \text{SUM}\ \frac{P \xrightarrow{\langle \delta,\alpha,\gamma\rangle} P'}{P + Q \xrightarrow{\langle \delta,\alpha,\gamma\rangle} P'}$$

$$\text{COM}\ \frac{P \xrightarrow{\langle \delta,\bar{x}\langle z,t,c\rangle,\gamma\rangle} P' \quad Q \xrightarrow{\langle \delta',x(\langle z,t,c\rangle),\gamma'\rangle} Q'}{(P \mid Q) \xrightarrow{\langle \delta\delta',\tau,\gamma\gamma'\rangle} (P' \mid Q')}$$

$$\text{OPEN}\ \frac{P \xrightarrow{\langle \delta,\bar{x}\langle y,t,c\rangle,\gamma\rangle} P'}{(y)P \xrightarrow{\langle \delta,\bar{x}\langle (u),t,c\rangle,\gamma\rangle} P'\{u/y\}}\ y \neq x \wedge u \notin fn((y)P')$$

$$\text{CLOSE}\ \frac{P \xrightarrow{\langle \delta,\bar{x}\langle (z),t,c\rangle,\gamma\rangle} P' \quad Q \xrightarrow{\langle \delta',x(\langle z,t,c\rangle),\gamma'\rangle} Q'}{(P \mid Q) \xrightarrow{\langle \delta\delta',\tau,\gamma\gamma'\rangle} (z)(P' \mid Q')}$$

$$\text{RES}\ \frac{P \xrightarrow{\langle \delta,\alpha,\gamma\rangle} P'}{(z)P \xrightarrow{\langle \delta,\alpha,\gamma\rangle} (z)P'}\ z \notin n(\alpha), z \in \mathcal{N}$$

$$\text{MATCH}\ \frac{P \xrightarrow{\langle \delta,\alpha,\gamma\rangle} P'}{[x = x]P \xrightarrow{\langle \delta,\alpha,\gamma\rangle} P'}\ x \in \mathcal{N} \text{ or } x \in \Gamma \qquad \text{REP}\ \frac{P \xrightarrow{\langle \delta,\alpha,\gamma\rangle} P'}{!P \xrightarrow{\langle \delta,\alpha,\gamma\rangle} (P'\theta_r \mid !P)}$$

$$\text{REP-COM}\ \frac{P \xrightarrow{\langle \delta,\bar{x}\langle z,t,c\rangle,\gamma\rangle} P' \quad P \xrightarrow{\langle \delta',x(\langle z,t,c\rangle),\gamma'\rangle} P''}{!P \xrightarrow{\langle \delta\delta',\tau,\gamma\gamma'\rangle} ((P'\theta'_r \mid P''\theta''_r) \mid !P)}$$

$$\text{REP-CLOSE}\ \frac{P \xrightarrow{\langle \delta,\bar{x}\langle (z),t,c\rangle,\gamma\rangle} P' \quad P \xrightarrow{\langle \delta',x(\langle z,t,c\rangle),\gamma'\rangle} P''}{!P \xrightarrow{\langle \delta\delta',\tau,\gamma\gamma'\rangle} ((z)(P'\theta'_r \mid P''\theta''_r) \mid !P)}$$

γ. With abuse of notation, we have used γ as a set of clocks to be reset. We call the triple $\langle \delta, \alpha, \gamma \rangle$ a timed action. The set of transition rules of timed π-calculus are represented in Table 1. These rules are labeled by timed actions.

In rule IN the incoming clock and time-stamp must satisfy the clock constraint in the receiving process, for transition to take place. The incoming clock might get reset upon arrival in the receiving process. These requirements are specified in the timed action for IN where t' and d (the received time-stamp and clock) are substituted for t and c, respectively. In rule OUT, d is a fresh copy of clock c which is created and sent along name y on outputting the name. The rule for COM is similar to that of original π-calculus; however, the clock c communicated between P and Q is a fresh clock name generated by rule OUT (the premise of COM).

The joint use of two rules OPEN and CLOSE is used for scope-extrusion of bound names (including clock names). A bound output combines with an input action, and once the bound name has been received, a restriction will be extended to the receiving process. This means the received name is still bound although its scope has grown. However, this restriction should not bind occurrences of free names in the receiving process. This is the reason for changing the name y to a fresh name u before sending y, as in the original π-calculus. Note that the OPEN rule does not changes the clock name c, as c is a fresh clock name generated by the rule OUT (the premise of OPEN).

Note that in the rule for REP the set of clock names in P' are *replaced* by fresh clock names by applying the renaming θ_r to P'. Similarly, in the rules for REP-COM and REP-CLOSE the clock names in the replicated processes are *replaced* by fresh clock names using ${\theta'}_r$ and ${\theta''}_r$. Note also that there are two more rules for SUM and PAR where the process Q takes an action. These rules are symmetric to SUM and PAR rules of Table 1 and are eliminated.

Example 5. Using OUT and OPEN we can derive:

$$(y)\bar{x}\langle y, t, c\rangle.P \xrightarrow{\langle \epsilon, \bar{x}\langle (u), t, d\rangle, \epsilon\rangle} P\{u/y\}$$

For all u such that u is y or $u \notin fn(P)$ and $d = c\theta_f$. Using IN we have that

$$x(\langle z, t_z, e\rangle).Q \xrightarrow{\langle \epsilon, x(\langle u, t, d\rangle), \epsilon\rangle} Q\{u/z, t/t_z, d/e\}$$

For all u such that u is z or $u \notin fn(Q)$. By applying CLOSE we derive

$$((y)\bar{x}\langle y, t, c\rangle.P \mid x(\langle z, t_z, e\rangle).Q) \xrightarrow{\langle \epsilon, \tau, \epsilon\rangle} (u)(P\{u/y\} \mid Q\{u/z, t/t_z, d/e\})$$

Next, we formally define the operational semantics of timed π-calculus and how the transitions change the interpretation.

Definition 6. *[1] A time sequence $w = w_1 w_2 \ldots$ is a finite or infinite sequence of time values $w_i \in \mathbb{R}$ with $w_i > 0$, satisfying the following constraints:*

– *Monotonicity: w increases strictly monotonically; that is, $w_i < w_{i+1}$ for all $i \geq 1$.*
– *Progress: For every $w \in \mathbb{R}$, there is some $i \geq 1$ such that $w_i \geq w$.*

A system specified by set of timed π-calculus processes starts with all the clocks initialized to 0. Moreover, for every two processes P and Q, $c(P) \cap c(Q) = \emptyset$, initially. As time advances the value of all clocks advances, reflecting the elapsed time. At time w_i, a process P_{i-1} takes a timed action $\langle \delta_i, \alpha_i, \gamma_i \rangle$ and evolves to P_i, if the current values of clocks satisfy δ_i. The clocks specified by γ_i are reset to 0, and thus start counting time with respect to it. This behavior is captured by defining *runs* of timed π-calculus processes. A run for a process P, records the state (process expression) and the values of all the clocks at the transition points. For a time sequence $w = w_1 w_2 \ldots$ we define $w_0 = 0$.

Definition 7. *A run r, denoted by $(\bar{P}, \bar{I})$, of a timed π-calculus process P, is a finite or an infinite sequence of the form*

$$\langle P_0, I_0 \rangle \xrightarrow[w_1]{\langle \delta_1, \alpha_1, \gamma_1 \rangle} \langle P_1, I_1 \rangle \xrightarrow[w_2]{\langle \delta_2, \alpha_2, \gamma_2 \rangle} \langle P_2, I_2 \rangle \xrightarrow[w_3]{\langle \delta_3, \alpha_3, \gamma_3 \rangle} \ldots$$

where P_i is a process and $I_i \in [\Gamma \to \mathbb{R}_{\geq 0}]$, for all $i \geq 0$, satisfying the following requirements:

– *Initiation: P_0 is the initial process expression, and $I_0(c) = 0$ for all $c \in \Gamma$.*
– *Consecution: for all $i \geq 1$, there is a transition of the form $P_{i-1} \xrightarrow{\langle \delta_i, \alpha_i, \gamma_i \rangle} P_i$ such that $(I_{i-1} + w_i - w_{i-1})$ satisfies δ_i and $I_i = [\lambda_i \mapsto 0](I_{i-1} + w_i - w_{i-1})$.*

Along a run $r = (\bar{P}, \bar{I})$, the values of the clocks at time $w_i \leq w \leq w_{i+1}$ are given by the interpretation $(I_i + w - w_i)$. When the transition from P_i to P_{i+1} occurs, the value $(I_i + w_{i+1} - w_i)$ is used to check the clock constraint. At time w_{i+1}, the value of a clock that gets reset is defined to be 0.

When the transition from $P_i = \delta\gamma x(\langle z, t, c \rangle).P$ to $P_{i+1} = P\{y/z, t'/t, d/c\}$ occurs ($\langle y, t', d \rangle$ is the received name), we check the satisfiability of the clock constraint $\delta\{d/c, t'/t\}$, similarly we reset the clocks specified by $\gamma\{d/c\}$. Intuitively, this means that the values of the received clock and time-stamp should satisfy the constraint δ for the transition to take place. Moreover, the incoming clock might get reset upon arrival. These requirements are specified in the timed action of rule IN in Table 1. When the transition from $P_i = \delta\gamma\bar{x}\langle y, t, c \rangle.P$ to $P_{i+1} = P$ occurs in which, the timed action $\langle \delta, \bar{x}\langle y, t, d \rangle, \gamma \rangle, d = c\theta_f$ takes place, the time-stamp t in $\bar{x}\langle y, t, d \rangle$ gets bound to $(I_i(c) + w_{i+1} - w_i)$. Note that at this point $I_{i+1}(d) = I_{i+1}(c)$.

2.5 Passing Clocks and Channels

Link (channel) passing in timed π-calculus is handled in exactly the same manner as in π calculus, in the sense that a process P can send a public channel x to a process Q. However, if Q already has access to a private channel x before the

Table 2. Axioms of structural congruence

α-conversion:	
$P \equiv Q$ if Q can be obtained from P by finite number of changes of bound names.	
Parallel Composition:	*Restriction:*
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	$(x)(y)\ P \equiv (y)(x)\ P$
$P \mid Q \equiv Q \mid P$	$(x)\ 0 \equiv 0$
$P \mid 0 \equiv P$	
	Replication:
Summation:	$!P \equiv P \mid !P$
$(P + Q) + R \equiv P + (Q + R)$	
$P + Q \equiv Q + P$	*Scope Extension:*
$P + 0 \equiv P$	$((y)P \mid Q) \equiv (y)(P \mid Q)$ if $y \notin fn(Q)$

transition, the latter must be renamed to avoid confusion: this is called scope intrusion [12]. If P has a private link x that it sends to Q, the scope of restriction will be extended, this is called scope extrusion [12]. In this case, if Q already has access to a public link x, then the name of the private link must be changed before the transition (these are reflected in rules OPEN and CLOSE).

All clocks in timed π-calculus are local clocks; moreover, processes access disjoint sets of clocks. When a process P sends its (local) clock c to another process Q, it creates a fresh copy of c and sends this copy to Q.

Assume that $P = \delta\gamma\bar{y}\langle x, t_x, c\rangle.P'$ and $Q = \delta'\gamma' y(\langle z, t_z, d\rangle).Q'$. Furthermore, assume that P sends $\langle x, t_x, c\rangle$ to process Q. This behavior can be captured by the following timed π-calculus transition.

$$(P \mid Q) \xrightarrow{\langle \delta\delta', \tau, \gamma\gamma' \rangle} (P' \mid Q'\{x/z, t_x/t_z, e/d\}), \qquad e = c\theta_f$$

Next, we define the structural congruence for proposed timed π-calculus.

2.6 Structural Congruence

The notion of structural congruence for timed π-calculus processes is identical to that of original π-calculus [12]. Two timed π-calculus processes are structurally congruent if they are identical up to structure. Structural congruence, $\equiv$, is the least equivalence relation preserved by the process constructs that satisfy the axioms in Table 2.

2.7 Timed Bisimulation

We would like to identify two processes which cannot be distinguished by an observer. We assume that the observer is able to observe all kinds of actions and moreover, it can observe the times at which the actions are taken place.

Definition 8. *A binary relation $\mathcal{S}$ on timed π-calculus processes is a (strong) timed simulation if $P\mathcal{S}Q$ implies that:*

1. *If* $\langle P, I\rangle \xrightarrow[w]{\langle \delta,\tau,\gamma\rangle} \langle P', I'\rangle$, *then for some* Q', $\langle Q, I\rangle \xrightarrow[w]{\langle \delta,\tau,\gamma\rangle} \langle Q', I'\rangle$ *and* $P' \mathcal{S} Q'$,
2. *If* $\langle P, I\rangle \xrightarrow[w]{\langle \delta,\bar{x}\langle y,t,c\rangle,\gamma\rangle} \langle P', I'\rangle$, *then for some* Q', $\langle Q, I\rangle \xrightarrow[w]{\langle \delta,\bar{x}\langle y,t,c\rangle,\gamma\rangle} \langle Q', I'\rangle$ *and* $P' \mathcal{S} Q'$,
3. *If* $\langle P, I\rangle \xrightarrow[w]{\langle \delta,\bar{x}\langle (y),t,c\rangle,\gamma\rangle} \langle P', I'\rangle$ *and* $y \notin n(P, Q)$, *then for some* Q', $\langle Q, I\rangle \xrightarrow[w]{\langle \delta,\bar{x}\langle (y),t,c\rangle,\gamma\rangle} \langle Q', I'\rangle$ *and* $P' \mathcal{S} Q'$,
4. *If* $\langle P, I\rangle \xrightarrow[w]{\langle \delta,x(\langle y,t,c\rangle),\gamma\rangle} \langle P', I'\rangle$ *and* $y \notin n(P, Q)$, *then for some* Q', $\langle Q, I\rangle \xrightarrow[w]{\langle \delta,x(\langle y,t,c\rangle),\gamma\rangle} \langle Q', I'\rangle$ *and for all* z, $P'\{z/y\} \mathcal{S} Q'\{z/y\}$.

The relation $\mathcal{S}$ *is a (strong) timed bisimulation if both* $\mathcal{S}$ *and its inverse are timed simulation. The relation* $\dot{\sim}$, *(strong) bisimilarity, on timed processes is defined by* $P \dot{\sim} Q$ *if and only if there exists a timed bisimulation* $\mathcal{S}$ *such that* $P \mathcal{S} Q$.

Example 6. Assume P is a timed π-calculus process defined as:

$$(c < 2)(c := 0)\bar{x}\langle y, t_y, c\rangle \mid (d := 0)z(\langle w, t_w, d\rangle)$$

in which, $x \neq z$. Then,

$$P \dot{\sim} (c < 2)(c := 0)\bar{x}\langle y, t_y, c\rangle.(d := 0)z(\langle w, t_w, d\rangle) + (d := 0)z(\langle w, t_w, d\rangle).(c < 2)(c := 0)\bar{x}\langle y, t_y, c\rangle$$

Analogous to strong bisimilarity in π-calculus, $\dot{\sim}$ is not in general preserved by substitution of names. It follows that (strong) timed bisimilarity is not preserved by input prefix. As a result, (strong) timed bisimilarity is not a congruence.

Example 7. Assume P is defined as in Example 6, and $Q = u(\langle z, t_z, e\rangle).(P)$ Then,

$$Q \not\dot{\sim} u(\langle z, t_z, e\rangle).((c < 2)(c := 0)\bar{x}\langle y, t_y, c\rangle.(d := 0)z(\langle w, t_w, d\rangle) + (d := 0)z(\langle w, t_w, d\rangle).(c < 2)(c := 0)\bar{x}\langle y, t_y, c\rangle)$$

The reason is that, if z is instantiated to x (the channel name received in u is x), then $P\{x/z\}$ will have a τ transition which cannot be simulated by the right hand side of the equation.

Definition 9. *If* $x \neq y$, *then* $\delta\gamma\bar{x}\langle (y), t, c\rangle.P$ *means* $(y)\ \delta\gamma\bar{x}\langle y, t, c\rangle.P$, *and the prefix* $\bar{x}\langle (y), t, c\rangle$ *is called a derived prefix.*

A collection of algebraic laws for (strong) timed bisimilarity, which are extensions of algebraic laws for bisimilarity in π-calculus [12], is presented in Table 3. Note that ρ in proposition 5(d) denotes a prefix, including a derived prefix.

Table 3. Timed bisimilarity algebraic laws

Proposition 1 $\dot{\sim}$ is an equivalence relation.

Proposition 2.a If for all v in which $v \in fn(P) \cup fn(Q) \cup \{y\}$, $P\{v/y\} \dot{\sim} Q\{v/y\}$ then $\delta\gamma x(\langle y, t_y, c\rangle).P \dot{\sim} \delta\gamma x(\langle y, t_y, c\rangle).Q$

Proposition 2.b If $P \dot{\sim} Q$ then
(a) $P + R \dot{\sim} Q + R$
(b) $P \mid R \dot{\sim} Q \mid R$
(c) $[x = y]P \dot{\sim} [x = y]Q$
(d) $(u)P \dot{\sim} (u)Q$
(e) $\delta\gamma\tau.P \dot{\sim} \delta\gamma\tau.Q$
(f) $\delta\gamma\bar{x}\langle y, t_y, c\rangle.P \dot{\sim} \delta\gamma\bar{x}\langle y, t_y, c\rangle.Q$

Proposition 3 *Match*
(a) $[x = y]P \dot{\sim} 0$ if $x \neq y$
(b) $[x = x]P \dot{\sim} P$
(c) $[c = d]P \dot{\sim} 0$ if $c \neq d$
(d) $[c = c]P \dot{\sim} P$

Proposition 4 *Summation*
(a) $P + 0 \dot{\sim} P$
(b) $P + P \dot{\sim} P$
(c) $P + Q \dot{\sim} Q + P$
(d) $P + (Q + R) \dot{\sim} (P + Q) + R$

Proposition 5 *Restriction*
(a) $(y)\ P \dot{\sim} P$ $y \notin fn(P)$
(b) $(x)\ (y)\ P \dot{\sim} (y)\ (x)\ P$
(c) $(y)\ (P + Q) \dot{\sim} (y)\ P + (y)\ Q$
(d) $(y)\ \delta\gamma\rho.P \dot{\sim} \delta\gamma\rho.(y)\ P$ $y \notin n(\rho)$
(e) $(y)\ \delta\gamma\bar{y}\langle x, t, c\rangle.P \dot{\sim} 0$
(f) $(y)\ \delta\gamma y(\langle x, t, c\rangle).P \dot{\sim} 0$

Proposition 6 *Composition*
(a) $P \mid 0 \dot{\sim} P$
(b) $P_1 \mid P_2 \dot{\sim} P_2 \mid P_1$
(c) $(y)P_1 \mid P_2 \dot{\sim} (y)(P_1 \mid P_2)$ $y \notin fn(P_2)$
(d) $(P_1 \mid P_2) \mid P_3 \dot{\sim} P_1 \mid (P_2 \mid P_3)$
(e) $(y)(P_1 \mid P_2) \dot{\sim} (y)P_1 \mid (y)P_2$ $y \notin fn(P_1) \cap fn(P_2)$

Proposition 7 *Expansion*
Let $P \equiv \sum_i \delta_i\gamma_i\rho_i.P_i$ and $Q \equiv \sum_j \eta_j\lambda_j\phi_j.Q_j$ where δ_i and η_j are constraints, γ_i and λ_j specify the set of clocks to be reset and ρ_i and ϕ_j are prefixes; $bn(\rho_i) \cap fn(Q) = \emptyset$ for all i, and $bn(\phi_j) \cap fn(P) = \emptyset$ for all j; then

$$P \mid Q \dot{\sim} \sum_i \delta_i\gamma_i\rho_i.(P_i \mid Q) + \sum_j \eta_j\lambda_j\phi_j.(P \mid Q_j) + \sum_{\rho_i \text{comp} \phi_j} \delta_i\eta_j\gamma_i\lambda_j\tau.R_{ij}$$

The relation ρ_i comp ϕ_j (ρ_i complements ϕ_j) holds in the following four cases, which also defines R_{ij}:

1. ρ_i is $\bar{x}\langle u, t, c\rangle$ and ϕ_j is $x(\langle v, t_v, d\rangle)$; then R_{ij} is $(P_i|Q_j\{u/v, t/t_v, e/d\})$ where $e = c\theta_f$,
2. ρ_i is $\bar{x}\langle(u), t, c\rangle$ and ϕ_j is $x(\langle v, t_v, d\rangle)$; then R_{ij} is $(w)(P_i\{w/u\}|Q_j\{w/v, t/t_v, e/d\})$ where w is not free in $(u)P_i$ or in $(v)Q_j$ and $e = c\theta_f$,
3. ρ_i is $x(\langle v, t_v, d\rangle)$ and ϕ_j is $\bar{x}\langle u, t, c\rangle$; then R_{ij} is $(P_i\{u/v, t/t_v, e/d\}|Q_j)$ where $e = c\theta_f$,
4. ρ_i is $x(\langle v, t_v, d\rangle)$ and ϕ_j is $\bar{x}\langle(u), t, c\rangle$; then R_{ij} is $(w)(P_i\{w/v, t/t_v, e/d\}|Q_j\{w/u\})$ where w is not free in $(v)P_i$ or in $(u)Q_j$ and $e = c\theta_f$.

Proofs of above propositions are extensions of the proofs for untimed π-calculus processes [12] (these extensions take clocks into account) which are not presented here due to lack of space.

3 Example: The Railroad Crossing Problem

The generalized railroad crossing (GRC) problem [7] describes a railroad crossing system with several tracks and an unspecified number of trains traveling through the tracks. The gate at the railroad crossing should be operated in a way that guarantees the *safety* and *utility* properties. The *safety* property stipulates that the gate must be down while there is a train in the crossing. The *utility* property states that the gate must be up (or going up) when there is no train in the crossing. The system is composed of three components: *train*, *controller* and *gate*. The components of the system which are specified via three timed automata in Fig. 1, communicate by sending and receiving signals. We specify the components of the system in timed π-calculus.

The controller at the railroad crossing might receive various signals from trains in different tracks. In order to avoid signals from different trains being mixed, each train communicates through a private channel with the controller. A new channel is established for each approaching train to the crossing area through which the communication between the train and the controller takes place. For simplicity of presentation we consider only one track in this example.

In our modeling of railroad crossing problem in timed π-calculus each component of the system is considered as a timed π-calculus process. This model is presented in Table 4. Note that the design of the railroad crossing problem shown in Fig. 1 (originally from [1]) does not account for the delay between the sending of *approach* (*exit*) signal by a train and receiving it by the controller. Similarly the delay between sending *lower* (*raise*) by the controller and receiving it by the gate is not taken into account. Arguably, in a correct design, the delay before *approach* is received by the controller should be taken into account. The *lower* signal must be sent within one unit of time since the time-stamp of the *original approach* but not the time at which the controller receives the signal (note that the controller resets its clock to remember the time it receives *approach*). In contrast, in our specification of the railroad crossing problem in timed π-calculus,

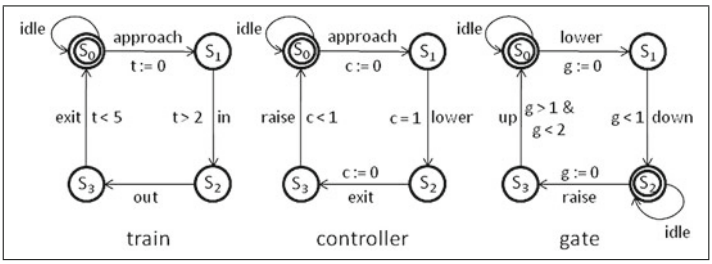

Fig. 1. Timed automata for train, controller, and gate in the railroad crossing problem

Table 4. The timed π-calculus expressions for components of the railroad crossing problem)

$$
\begin{array}{ll}
train \equiv & controller \equiv \\
\quad !(ch)\overline{ch1}\langle ch, t_c, t\rangle. & \quad !ch1(\langle y, t_y, d\rangle).y(\langle x, t_x, c\rangle). \\
\quad (t := 0)\overline{ch}\langle approach, t_a, t\rangle. & \quad ([x = approach](c = 1)(e := 0)\overline{ch2}\langle lower, t_l, e\rangle + \\
\quad (t > 2)\tau.\tau. & \quad\ [x = exit](c - t_x < 1)(e := 0)\overline{ch2}\langle raise, t_r, e\rangle) \\
\quad (t < 5)\overline{ch}\langle exit, t_e, t\rangle) & \\
gate \equiv & \\
\quad !ch2(\langle x, t_x, g\rangle). & \\
\quad ([x = lower](g < 1)\tau + [x = raise](g > 1)(g < 2)\tau) & \\
 & \\
 & main \equiv train \mid controller \mid gate
\end{array}
$$

we are considering the delays; therefore, all the time-related reasoning in the system is performed against *train*'s clock and the time-stamp of *approach* signal (sent by *train* to *controller*).

Note that in the π-calculus expression for *train* specified in Table 3, t is the local clock of *train* and the two consecutive τ actions correspond to *train*'s internal actions *in* and *out*. In the expression for *controller*, c is a place holder for the received clock t from *train*; while, e is the controller's clock that is reset before it is sent to *gate*. In the expression for *gate*, g is a place holder for the received clock e from *controller* and the two τ actions correspond to *gate*'s internal actions; the first τ represents *down*; while the second τ represents *up*.

Timed π-calculus allows the railroad crossing problem to be modeled more faithfully. Additionally, significantly more complex systems can be modeled. The timed π-calculus specification can be used for verification of the system as well as generating the implementation [15].

4 Discussions

Our proposed timed π-calculus extends the original π-calculus of Milner, while *preserving* the algebraic properties of the original π-calculus. The notion of timed bisimilarity and expansion in our calculus are also simple extensions of those found in the original π-calculus. Our calculus is a simple and powerful calculus which annotates the transitions of π-calculus with timing constraints. Our effort was driven by our desire to keep the design simple. The two most critical and fundamental assumptions/decisions made in this paper are discussed next.

First, on outputting a name we submit a clock and also the time-stamp of the name generated by the clock. Without time-stamps, precisely reasoning about channel delays becomes much more complex, if not impossible. As an example consider a scenario in which process P takes an action α and resets its clock c at the time of action in order to remember when the action took place. Let us assume that after t units of time (t is measured by c) have elapsed since α's occurrence, P sends a name n (along with clock c and time-stamp t_n generated

by c) to process Q. Note that P did not reset the clock before this output, as it continues to need to measure the time since the occurrence of α. At this point if we wanted to know for how long the name n was in transit, we could not calculate it without t_n. However, on receiving n on Q, the expression $c - t_n$ could be used to calculate the exact time for which the name n was in transit.

The second fundamental design decision in our calculus is our choice of local clocks and how clocks are treated. Initially, processes have access to distinct set of clocks. Later, after transitions take place, the scope of local clocks may grow; however, we keep the distinction between clocks of different processes. We achieve this by having processes create fresh copies of their own clocks (and updating the interpretation accordingly) and sending these copies instead of their original clocks. Adopting this convention prevents processes from resetting each others' clocks, as the sending process may keep using (possibly resetting) the clock that was sent for measuring other subsequent events. Our choice of clocks and our careful treatment of clocks enabled us to extend the transition rules of the original π-calculus naturally in order to obtain the transition rules of our timed π calculus. It also made our expansion theorem an straightforward extension of the original one.

5 Conclusions and Related Work

Since the π-calculus was proposed by Milner et al. [12], many researchers have extended it for modeling distributed real-time systems. Berger has introduced timed π-calculus (π_t-calculus) [2], asynchronous π-calculus with timers and a notion of *discrete* time, locations, and message failure, and explored some of its basic properties. Olarte has studied temporal CCP (tcc) as a model of concurrency for mobile, timed reactive systems in his Ph.D thesis [13]. He has developed a process calculus called universal temporal CCP (utcc). His work can be seen as adding mobile operation to the tcc. In utcc, like tcc, time is conceptually divided into time intervals (or time units); therefore it is discretized. Lee et al. [10] introduced another timed extension of π-calculus called real-time π-calculus (πRT-calculus). They have introduced the time-out operator and considered a global clock, single observer as part of their design, as is common in other (static) real-time process algebras. They have used the set of natural numbers as the time domain, i.e., time is *discrete* and is strictly increasing. Ciobanu et al. [4] have introduced a model called timed distributed π-calculus in which they have considered timers for channels, by which they restrict access to channels. They use *decreasing* timers, and time is discretized in their approach also. Many other timed calculi have similar constructs which also discretize time [5,8,11]. In summary, all these approaches share some common features; they use a *discrete* time-stepping function or timers to increase/decrease the time-stamps after every action (they assume that every action takes exactly one unit of time). *In contrast, our approach for extending π-calculus with time faithfully treats time as continuous.*

Posse et al. [14] have proposed π_{klt}-calculus as a real-time extension of π-calculus and study a notion of time-bounded equivalence. They have developed

an abstract machine for the calculus and developed an implementation based on this abstract machine for the π_{klt}-calculus in a language called `kiltera`. The replication operator of the original π-calculus is missing in this work.

The work of Yi [17] shows how to introduce time into Milner's CCS to model real-time systems. An extra variable t is introduced which records the time delay before a message on some channel α is available, and also a timer for calculating delays. The idea is to use delay operators to suspend activities. In our opinion, it is much harder to specify real-time systems using delays. Our approach provides a more direct way of modeling time in π-calculus via clocks, and also can be used to elegantly reason about delays.

The proposed timed π-calculus is an expressive, natural model for describing real-time, mobile, concurrent processes. It preserves the algebraic rules of the original π-calculus, while keeps the expansion theorem simple.

With respect to future work, we would like to extend our timed π-calculus with other continuous quantities; so that more complex systems as well as CPS can be expressed. While we have used our calculus for modeling and verifying the railroad crossing problem [15], we would like to model the generalized railroad crossing (GRC) problem in our calculus and use it for verifying properties of the system.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
2. Berger, M.: Towards abstractions for distributed systems. Technical report, Imperial College London (2004)
3. Chen, J.: Timed extensions of π calculus. Theor. Comput. Sci. **11**(1), 23–58 (2006)
4. Ciobanu, G., Prisacariu, C.: Timers for distributed systems. Electr. Notes Theor. Comput. Sci. **164**(3), 81–99 (2006)
5. Degano, P., Loddo, J.V., Priami, C.: Mobile processes with local clocks. In: Dam, M. (ed.) LOMAPS-WS 1996. LNCS, vol. 1192, pp. 296–319. Springer, Heidelberg (1997)
6. Gupta, R.: Programming models and methods for spatiotemporal actions and reasoning in cyber-physical systems. In: NSF Workshop on CPS (2006)
7. Heitmeyer, C., Lynch, N.: The generalized railroad crossing: a case study in formal verification of real-time systems. In: IEEE Real-Time Systems Symposium, pp. 120–131. IEEE Computer Society Press, Los Alamitos (1994)
8. Laneve, C., Zavattaro, G.: Foundations of web transactions. In: Sassone, V. (ed.) FoSSaCS 2005. LNCS, vol. 3441, pp. 282–298. Springer, Heidelberg (2005)
9. Lee, E.A.: Cyber physical systems: design challenges. In: IEEE Symposium on Object Oriented Real-Time Distributed Computing, ISORC '08. IEEE Computer Society, Washington (2008)
10. Lee, J.Y., Zic, J.: On modeling real-time mobile processes. Aust. Comput. Sci. Commun. **24**(1), 139–147 (2002)
11. Mazzara, M.: Timing issues in web services composition. In: Bravetti, M., Kloul, L., Zavattaro, G. (eds.) EPEW/WS-EM 2005. LNCS, vol. 3670, pp. 287–302. Springer, Heidelberg (2005)

12. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, parts i and ii. Inf. Comput. **100**(1), 1–77 (1992)
13. Olate, C.: Universal temporal concurrent constraint programming. Ph.D thesis, LIX, Ecole Polytechnique (2009)
14. Posse, E., Dingel, J.: Theory and implementation of a real-time extension to the π-calculus. In: Hatcliff, J., Zucca, E. (eds.) FMOODS/FORTE 2010, Part II. LNCS, vol. 6117, pp. 125–139. Springer, Heidelberg (2010)
15. Saeedloei, N.: Modeling and verification of real-time and cyber-physical systems. Ph.D. thesis, University of Texas at Dallas, Richardson, Texas (2011)
16. Saeedloei, N., Gupta, G.: A logic-based modeling and verification of CPS. SIGBED Rev. **8**, 31–34 (2011). http://doi.acm.org/10.1145/2000367.2000374
17. Yi, W.: CCS + time = an interleaving model for real time systems. In: Albert, J.L., Monien, B., Artalejo, M.R. (eds.) ICALP 1991. LNCS, vol. 510, pp. 217–228. Springer, Heidelberg (1991)