On Modeling Real-time Mobile Processes

Jeremy Y. Lee

School of Computer Science and Engineering
University of New South Wales
NSW 2052 Australia
Email: yjlee@cse.unsw.edu.au

John Žic[†]

Principal Research Engineer
Motorola Australian Research Centre
Level 3, 12 Lord Street, Botany
NSW 2019 Australia
Email: jjzic@arc.corp.mot.com

Abstract

This paper introduces an algebra for modeling the real-time aspect of systems in a mobile environment. Our model makes use of many familiar concepts and properties from previous works on static, real-time process algebra, and incorporates these into Milner's dynamic π-calculus. The extended algebra allows both time values as well as names to be transmitted between processes, thereby allowing the modeling of, and reasoning about, *dynamic temporal behaviour* and *dynamic configurations* of systems. Further, conventional labeled transition system (LTS) semantics are found to be inadequately expressive for use with our algebra. We therefore propose a *timed labeled transition system* (*TLTS*) semantics. Finally, we illustrate the modeling power of our algebra with a comprehensive but simple example of a mobile streaming video player.

Keywords: real-time system, distributed system, process algebra, mobile processes, time transmission

1 Motivation

Process algebras (or calculi) have gained popularity in describing computing systems over the last two decades because their alegbraic notation and their compositional properties allow the description and analysis of non-timed concurrent systems. Real-time concurrent systems require considerable refinement of the process algebras, in both the algebraic notation as well as the underlying semantic models. Considerable effort in this latter area has resulted in a large collection of real-time versions of untimed process algebras, such as CCS (e.g. [M83, MT90, W91b, HR95, H91, FZ95]), CSP (e.g. [RR88, S91, DS95, S96]) and ACP (e.g. [BB91, NS94]).

Ever since the boom of the mobile communications industry in the 90's, various mobile devices have been manufactured. As a result, mobile computing has gained more and more attention over the past decade. Unfortunately, the conventional process algebras are not particularly suited for describing and reasoning about mobile processes since the structure and composition of processes and interactions among them are static while mobile computing imposes a dynamic nature into process interactions. This has led to the introduction of a new type of process algebra that takes mobility into account. In particular, the π-calculus [MPW92, M91, M99, P00] (which stems from CCS [M80, M89]) has drawn a great deal of interest from the computer science community because of its ability to describe dynamic process configuration or mobile processes.

Despite the attraction of using π-calculus to model mobile systems, real-time applications like mobile multimedia systems (e.g. streaming video player on mobile laptop) cannot be modeled satisfactorily with π-calculus due to its lack of time expressiveness. Yet, there are very few real-time extensions to π-calculus in the literature. The most notable of these is the stochastic π-calculus [P95, DLP96, P97] that allows probabilistic time delays to be associated with synchronising actions. The probabilities are drawn from some well-defined distribution, and can be used to describe the quantitative performance of systems. However, stochastic π-calculus has true concurrency semantics (rather than the simpler interleaving semantics) and hence is more complicated to reason about. We would like to develop a *simple* real-time extension to π-calculus with a simpler interleaving semantics.

This paper describes such an extension to the π-calculus. We call this algebra the *πRT-calculus*. Our interest is to make the modeling and reasoning about systems' *dynamic temporal behaviour* and *dynamic configurations* as simple and expressive as possible, and so there are minimal extensions to the π-calculus. We introduce only one new operator, the *timeout operator*. When introducing time notion to our model, we also make a distinction between when an action is possible (*enabled*) and when an action actually occurs (*fires*) since there might be delay between these two states that we want to capture. Many properties of our model are the same or similar to the previous works on static real-time algebras (a survey of these works is presented in [NS91]) and we still use the conventional interleaving semantics where we separate time events from normal actions. Inspired by the notion of name transmission, we also allow time transmission. Although many conventional real-time calculi can also make a trivial extension of transmitting time, it is the dynamic process configuration from π-calculus coupled with time transmission in our algebra that allows us to model real-time computing systems in a mobile environment.

This paper is organised as follows. Section 2 briefly introduces the syntax and notations of π-calculus. Section 3 presents a number of general design choices and properties of our model. Section 4 gives the syntax and operational semantics of our algebra. A number of technical issues are discussed regarding adding time notion to π-calculus. We also extend the conventional LTS since it neither represents our model faithfully nor preserves our model properties. We call this extended LTS, *timed LTS* or *TLTS*. We illustrate it with a simple vending machine example. Then in section 5 we give a comprehensive but simple example of a mobile streaming video player to illustrate the modeling power of our algebra. Section 6 concludes the work here.

[†] Previously with the School of Computer Science and Engineering, University of New South Wales.

2 Introduction to π-calculus

π-calculus [MPW92] stems from CCS [M80, M89]. CCS is a process algebra that can model concurrent processes and their interactions through handshaking. Each process term (or agent) in CCS has a static number of ports through which it can interact (communicate) with other agents (via handshaking or synchronisation). These ports are the agent's sort.

We assume an infinite set $\mathcal{N}$ of names and this set has a corresponding set $\overline{\mathcal{N}}$ of co-names with $\overline{\overline{\mathcal{N}}} = \mathcal{N}$. Ports are labeled with names taken from the set of labels $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$. Ports labeled with the corresponding names (input ports) and co-names (output ports) are linked together so the agents can communicate (synchronise) via these ports. The synchronisation will constitute an internal action, τ, within the two participating agents. Non-determinism is modeled when there are more than two ports linking together where only a pair can synchronise. Hence the whole set of actions $Act = \mathcal{L} \cup \{\tau\}$.

π-calculus extends CCS in that it allows agents to have dynamic sorts. Names (ports or channels) can be freely transmitted via names. As a result, it can model change of links and hence a dynamic structure of agents.

2.1 Operators and Notations

The operators and syntax used in π-calculus are briefly described below. Let P, Q be any agent. They can take on the following syntax. We use the metavariables w, x, y to range over the set of names $\mathcal{N}$.

- 0: Inaction - an agent that represents a terminated or deadlocked process.
- $\overline{x}y.P$: Output action - an agent that outputs a name y on the output port $\overline{x}$ and then behaves as P.
- $x(y).P$ Input action - an agent that can input a name w from the input port x and binds to the name y and then behaves as $P\{w/y\}$ (substituting the name w for every occurrence of the name y in P).
- $\tau.P$: Internal action - an agent that performs a silent action that requires no interaction with the environment. τ can also arise from the communication of the corresponding input and output ports of two distinct agents.
- $(\nu x)P$: Scoping - an agent that makes the name (port) x invisible to the environment and hence denies interaction through port x or $\overline{x}$ with the environment.
- $[x = y]P$: Match - an agent that behave as P if the name x matches the name y.
- $P|Q$: Parallel composition - an agent that is composed of P and Q running concurrently. The composite agent can interact with the environment or the sub-agents can interact with each other within the composite agent.
- $P+Q$: Choice - an agent that can behave as P or Q depending on which can proceed. The choice is non-deterministic if both can proceed.
- $A(\tilde{x})$: Agent identifier - an agent that is defined by a process term containing the set of names $\tilde{x} = x_1, x_2, \ldots, x_n$ where the x_i are pairwise distinct.

In the agents $x(y).P$ and $(\nu y)P$ the occurrence of y is a *binding occurrence* and its scope in each case is P. An occurrence of a name y in an agent is said to be *free* if it does not lie within the scope of a binding occurrence of y. We denote the set of names occurring free within an agent P as $\mathrm{fn}(P)$. An occurrence of a name in an agent is said to be *bound* if it is not free and we denote the set of bound names within an agent P as $\mathrm{bn}(P)$. We denote the whole set of names occurring in an agent P as $\mathrm{n}(P) = \mathrm{fn}(P) \cup \mathrm{bn}(P)$.

3 Design Choices

3.1 Model Features

Global clock, Single observer: as is common in other (static) real-time algebras, this simplifies the formalisation.

Discrete time: We will use the set of natural numbers as the *time domain*, i.e. time is discrete and is strictly increasing. We can view each passing of a time unit as a clock tick and we model it as an explicit event as in [NS94]. This is somewhat contrary to the popular way of thinking that the passage of time is associated with observing a succession of distinct, observable events [L78]. By treating time passing as an explicit event, we can let time pass without any other actions occuring. (See also the *time visibility* property in sub-section 3.3.) We use the symbol $\varepsilon(t)$ to represent this time event as in [W90, W91a] and t represents the number of clock ticks passed in this time event.

Separation of actions and time events: As in almost all previous works on real-time process algebra, we adopt an interleaving model where we separate time events from normal actions.

Synchronous time events: Time progresses in a synchronous fashion, i.e. time events are observed by all components of a system and are allowed only if all components agree to participate.

Time transmission: Inspired by the notion of name transmission in π-calculus, we can treat time just as other names and transmit them on channels. Just as channel transmission results in dynamic configuration of processes, time transmission can result in dynamic temporal behaviour of processes.

3.2 Atomic actions and Idling

In conventional LTS semantics, $P \xrightarrow{\alpha} Q$ only states that the agent P *may* offer an action α and by doing so evolve to Q, but says nothing about *when* the transition occurs. When introducing time into our model, we wish to distinguish the situation where an agent *may* offer an action from the situation where an agent actually *commits* an action, i.e. the transition actually occurs. This will allow us to capture the possible time delay between these two states. We wish to make the following definitions explicit.

Definition 1 *An action α is enabled in an agent P if there exists an agent Q where the transition $P \xrightarrow{\alpha} Q$ is possible. In a state transition diagram, it means that there is a path out of state P, labeled α, to a state Q.*

Definition 2 *An action α is being fired in an agent P if the transition $P \xrightarrow{\alpha} Q$ occurs. This advances the transition history of P by one step. We denote the firing of an action α from P to Q as $P \overset{\alpha}{\hookrightarrow} Q$ to distinguish it from the usual enabling denotation.*

As soon as an agent P offers an action α, the action α is enabled. For example, figure 1 shows a transition diagram in which actions a, b are enabled in state P but not action c. It means that P has two transition choices, a or b. It does not show which path to follow. On the other hand, P fires action b if the path labeled with b is actually followed.

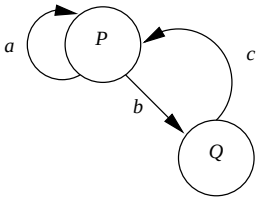

Figure 1: A sample transition diagram illustrating enabling.

An action is *atomic* or instantaneous in the sense that the firing of the action takes no time. To model a non-atomic action, we can explicitly put some delay before that action to model its duration. A note is in order. Since action is atomic, it is possible to have infinite actions performed within a finite time interval or even zero time (as opposed to finite variability in [NS91]). However, imposing restrictions regarding finite actions will make the theory more complicated. Hence, for realisable implementations, it is the designer's responsibility to impose proper and correct delays for each action.

As in [W91a], we wish to define controllable and non-controllable actions, but in terms of enabling and firing.

Definition 3 *A controllable action is an action whose enabling and firing may be separated in time.*

Definition 4 *A non-controllable action is an action whose enabling and firing coincide in time.*

Definition 5 *An agent is idle during the period of time between the enabling and firing of a controllable action.*

These two types of actions are distinguished from the environment's point of view. The environment has control over when controllable actions can fire and hence these actions are controllable. These are visible actions and since these actions need synchronisation with the environment, the enabling and firing of which *may* be separated in time. Hence, after an action is enabled but before it is fired, an agent is idle. On the other hand, the environment has no control over when non-controllable actions can fire and hence these actions are non-controllable. For example, internal actions are classified as non-controllable since they do not need synchronisation with the environment. Once they are enabled, they will fire immediately and hence the enabling and firing of which always coincide in time. That is, we do not allow time to pass between the enabling and firing of non-controllable actions. Restating these facts, we note that an action is non-controllable if we cannot distinguish between the occurrence (and time) at which they are enabled, and the occurrence (and time) at which they are fired, no matter what timing granularity we may impose. Figure 2 shows two time diagrams that illustrate the two types of actions.

3.3 Model Properties

Many of the following properties are also defined similarly in [W90, W91a, NS91].

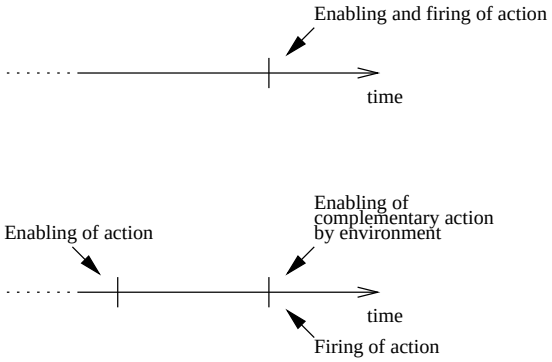

Figure 2: Non-controllable action (above) and controllable action (below).

Maximal progress: This property states that there is no unnecessary waiting when an action can be fired. It is certainly true for the non-controllable actions. It also states that if two complementary controllable actions are enabled, they will necessarily fire immediately and at the same time since their reaction constitute an internal action which is non-controllable. Formally, if $P \xrightarrow{\tau} P'$ for some P', then for $t > 0$, $P \xrightarrow{\varepsilon(t)} P''$ for no P''. This property suggests that non-controllable actions take precedence over controllable actions. However, unlike other works incorporating priority in the calculus, e.g. see [CLN99], there is neither explicit priority assigned to actions nor preemption mechanism introduced. This kind of priority is only a weak form in the sense that non-deterministic choice is still being made among enabled actions. It only states that an enabled action will take precedence over "not yet" enabled actions. This is similarly defined in [W91b, H91, S96, NS94].

Timelock freeness: There is no time lock in our model. All agents can allow time to pass if they cannot perform non-controllable actions, or as stated in [H91], there is arbitrary waiting in these agents. Formally, for all agent P and $P \xrightarrow{\alpha} P'$ for some P' and $\alpha \neq \tau$, then for $t > 0$, $P \xrightarrow{\varepsilon(t)} P''$ for some P''.

Time visibility: Since all components need to participate to let time pass, time is always visible and restriction has no effect on time progress. Formally, for $t > 0$, if $P \xrightarrow{\varepsilon(t)} P'$, then for any name x free in P, $(\nu x)P \xrightarrow{\varepsilon(t)} (\nu x)P'$.

Time determinacy: We impose that non-deterministic choice can only be made by ordinary actions but not time events, i.e. we do *not* allow non-deterministic choice to be made between an ordinary action and a time event. When an agent P is idle for some duration t, its resulting behaviour is completely determined by P and t. Formally, for $t > 0$, whenever $P \xrightarrow{\varepsilon(t)} P'$ and $P \xrightarrow{\varepsilon(t)} P''$, then $P' \equiv P''$ where $\equiv$ is a syntactically equivalent relation. Note that an agent P can still make a non-deterministic choice among a number of possible delays. It is deterministic only in the sense that *after* delaying for a certain period of time t, P will have deterministic behaviour. See also issue 4 in sub-section 4.2.

Time continuity: This is also known as time additivity in [NS91]. If an agent can idle for $t1+t2$ time units, then it can also idle for $t1$ and then $t2$ time units. Formally, for $t > 0$ and $u > 0$, $P \xrightarrow{\varepsilon(t+u)} P'$ iff there exists P'' such that $P \xrightarrow{\varepsilon(t)} P''$ and $P'' \xrightarrow{\varepsilon(u)} P'$.

Action persistency: This property states that if an action is enabled in an agent, the progress of time in that agent cannot suppress the (later) firing of that action. Formally, for $t > 0$, if $P \xrightarrow{\varepsilon(t)} P'$ and $P \xrightarrow{\alpha} Q$, then $P' \xrightarrow{\alpha} Q'$ for some Q'.

4 The πRT-calculus

4.1 Syntax and Operational Semantics

The πRT-calculus is based on Milner's π-calculus. Before presenting the syntax and semantics of our calculus, we give a brief account of the conventions and notations used in πRT-calculus.

- An infinite set $\mathcal{N}$ of names ranged over by x, y, etc. We assume $\varepsilon \notin \mathcal{N}$.
- The set $Act = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ of actions ranged over by α.
- The time domain D of natural numbers ranged over by t, u and the set $\Delta = \{\varepsilon(t) | t \in D\}$ of time events. We will treat t, u just as other names in $\mathcal{N}$.
- The set $Act \cup \Delta$ of all labels in our LTS ranged over by γ.
- The extended set $\mathcal{N} \cup D$ of names for the transmitted objects in the input and output actions and formal parameters of agent identifiers and is ranged over by m, n.

The syntax of πRT-calculus is given by:

$$\begin{array}{lcccccccc} P,Q & ::= & 0 & | & \overline{x}m.P & | & x(m).P & | & \tau.P & | \\ & & (\nu x)P & | & [x=y]P & | & P \overset{t}{\triangleright} Q & | & P|Q & | \\ & & P+Q & | & A(\tilde{m}) & & & & & \end{array}$$

- We take the syntax of the monadic version. In cases of multiple transmitted objects in the input and output actions, we will use the notations of $\overline{x}\langle\tilde{m}\rangle$and $x(\tilde{m})$ for the output and input actions respectively where $\tilde{m} = m_1, m_2, \ldots, m_n$ in the objects and in the formal parameters of the agent identifier $A(\tilde{m})$. These objects $\tilde{m}$ now take values from the set $\mathcal{N} \cup D$.
- The precedence of the operators is listed from the highest to the lowest in the order above.
- The timeout operator $P \overset{t}{\triangleright} Q$ is also similarly defined in [RR88, S91, NS94, HR95, H91]. It behaves as P before t time units and as Q *at* and *after* t time units. The first action of P is enabled for the duration of up to t time units exclusive and it can fire at any time during this period at which time Q will be dropped. If it does not fire within this period, at and after t time units, the first action of Q will be enabled and P will be dropped. We call P the *body* and Q the *exception* of the timeout operator. Note that the timeout operator is *not* associative and it associates to the right:

$$P \overset{t}{\triangleright} Q \overset{u}{\triangleright} R \equiv P \overset{t}{\triangleright} (Q \overset{u}{\triangleright} R)$$

- In an earlier version of the πRT-calculus, timeouts were specified using a prefix delay $\varepsilon(t)$ rather than using the timeout operator as suggested in [DS95]. However, the use of prefix delay for describing and reasoning about timeouts proved to be very cumbersome. The current version of the πRT-calculus uses the timeout operator to specify delays. For example, $0 \overset{t}{\triangleright} P$ means that the process P has been delayed by t time units.
- We treat the parameter t in the timeout operator $P \overset{t}{\triangleright} Q$ as a free name, i.e. $\mathrm{fn}(P \overset{t}{\triangleright} Q) = \mathrm{fn}(P) \cup \mathrm{fn}(Q) \cup \{t\}$. It can be substituted by any value from the transmitted object during execution. This contributes to the dynamic behaviour of the timeout operator.

In addition to the structural congruence laws for π-calculus, we add a few more laws to the set of structural congruence:

Zero Timeout: $P \overset{0}{\triangleright} Q \equiv Q$

Scope Extension over Timeout:

1. $(\nu x)(P \overset{t}{\triangleright} Q) \equiv (\nu x)P \overset{t}{\triangleright} Q$ if $x \notin \mathrm{fn}(Q)$
2. $(\nu x)(P \overset{t}{\triangleright} Q) \equiv P \overset{t}{\triangleright} (\nu x)Q$ if $x \notin \mathrm{fn}(P)$

Distribution over choice:

$$P \overset{t}{\triangleright} Q + R \overset{t}{\triangleright} S \equiv (P+R) \overset{t}{\triangleright} (Q+S)$$

We take the (late) operational semantics of π-calculus as our starting point. In our operational semantics, all the premises and conclusions in the rules are in terms of enabling (as in the conventional cases), i.e. these are enabling semantics. The added notion of firing is just to let us capture the time delay between enabling and firing (if there is any). We assume structural congruence can be applied at any point in inferring the rules. The transition rules of our algebra are listed below:

INACTION-DEL: $\dfrac{}{0 \xrightarrow{\varepsilon(t)} 0} \quad t > 0$

TAU-ACT: $\dfrac{}{\tau.P \xrightarrow{\tau} P}$

OUTPUT-ACT: $\dfrac{}{\overline{x}m.P \xrightarrow{\overline{x}m} P}$

OUTPUT-DEL: $\dfrac{}{\overline{x}m.P \xrightarrow{\varepsilon(t)} \overline{x}m.P} \quad t > 0$

INPUT-ACT: $\dfrac{}{x(m).P \xrightarrow{x(m)} P}$

INPUT-DEL: $\dfrac{}{x(m).P \xrightarrow{\varepsilon(t)} x(m).P} \quad t > 0$

TIMEOUT-ACT: $\dfrac{P \xrightarrow{\alpha} P'}{P \overset{t}{\triangleright} Q \xrightarrow{\alpha} P'} \quad t > 0$

TIMEOUT-DEL: $\dfrac{P \xrightarrow{\varepsilon(u)} P'}{P \overset{t}{\triangleright} Q \xrightarrow{\varepsilon(u)} P' \overset{t-u}{\triangleright} Q} \quad t \geq u > 0$

SUM: $\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$

SUM-DEL: $\dfrac{P \xrightarrow{\varepsilon(t)} P' \quad Q \xrightarrow{\varepsilon(t)} Q'}{P+Q \xrightarrow{\varepsilon(t)} P'+Q'} \quad t > 0$

PAR: $\dfrac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$

PAR-DEL: $\dfrac{P \xrightarrow{\varepsilon(t)} P' \quad Q \xrightarrow{\varepsilon(t)} Q' \quad \neg(P|Q \xrightarrow{\tau})}{P|Q \xrightarrow{\varepsilon(t)} P'|Q'} \quad t > 0$

$$\text{MATCH:}\quad \frac{P \xrightarrow{\gamma} P'}{[x=x]P \xrightarrow{\gamma} P'}$$

$$\text{RES:}\quad \frac{P \xrightarrow{\gamma} P'}{(\nu x)P \xrightarrow{\gamma} (\nu x)P'} \quad x \notin \mathrm{n}(\gamma)$$

$$\text{COM:}\quad \frac{P \xrightarrow{\overline{x}n} P' \quad Q \xrightarrow{x(m)} Q'}{P|Q \xrightarrow{\tau} P'|Q'\{n/m\}}$$

$$\text{OPEN:}\quad \frac{P \xrightarrow{\overline{x}y} P'}{(\nu y)P \xrightarrow{\overline{x}(y)} (\nu y)P'} \quad y \neq x$$

1. Rules INACTION-DEL, OUTPUT-DEL and INPUT-DEL allow time to progress in inaction and prefix agents. Note that there is the risk of unguarded recursion since the source and target of these transitions are the same. However, this highly undesirable behaviour may be avoided when we use it in conjunction with the PAR-DEL rule (see point 6 below). Other real-time calculi have taken a similar approach e.g. [S91, HR95, W91b, NS94].

2. The prefix τ has only one rule, TAU-ACT. It cannot allow time to pass due to the property of maximal progress.

3. The rule TIMEOUT-ACT states if an enabled action in the body is fired within the timeout period, then it continues thereafter and the exception is dropped. Note that the firing of an internal action will also drop the exception, which is different from the timeout operator in [S91] which only allows the firing of an external action to cancel the exception.

4. The rule TIMEOUT-DEL allows time to advance and the timeout period shortens accordingly. Note that the body advances the same amount of time. In this respect, it is different from the timeout operators of [NS94, HR95, H91], which use only unit delay and the body does not advance in time. However, it is similar to the start delay operator in ATP [NS94].

5. The rule SUM-DEL advances time only if both choices advance the same time. Note the passage of time does not resolve the choice which is the property of time determinacy and action persistency.

6. The rule PAR-DEL advances time only if both components advance the same time and no reaction occurs between them within this time period. The negative premise $\neg(P|Q \xrightarrow{\tau})$ makes sure they do not react and is the condition that guards the prefix action delays from unguarded recursion, i.e. it makes sure if the components can react, they will do so in no time and hence cannot delay (maximal progress). See also issues 3 and 5 in sub-section 4.2.

7. Rules MATCH and RES have the metavariable γ in the transition label, which means these rules also apply to time transitions, while rule OPEN only applies to normal action transitions.

8. The rest of the rules are the same as in π-calculus.

4.2 Technical Issues

The approach we have taken in integrating time notion into π-calculus is essentially the same as in conventional calculi. In particular, our model properties are similar to [W90, W91a] in many ways although we have used the timeout operator as in [S91, NS94]. There are however, a few issues which need to be clarified.

1. The introduction of transmission of time raises name compatibility issues, i.e. sorting. This is also a problem in π-calculus. Basically, we need to divide the name space into at least two partitions: one for normal port names, the other for time. We do not want meaningless agents such as $x(t).(P \overset{t}{\triangleright} Q)|\overline{x}y.R$ where the name y transmitted via x cannot be used as a timeout value. However, we will not enforce such sorting at this time.

2. The scoping of time, e.g. $(\nu 5)(P \overset{5}{\triangleright} Q)$, is also meaningless. Pure π-calculus also introduces this problem as it treats port names and data the same.

3. The transition rules for the prefix actions introduce unguarded recursion of time events when a single agent is defined outside the parallel composition. This is reasonable as we would expect such agent cannot proceed at all and can only wait (idle) forever since there is no other agents interacting with it. There is no timelock.

4. An agent can usually make a choice among a number of possible delays. For example, since time is continuous, an agent which can delay 5 time units can also delay 1, 2, 3 and 4 time units. When two parallel agents are prepared to delay for some period of time, they must agree on the actual length of the delay. So they are also making choice among a number of possible delays. But this will not affect their behaviour because for any delay they choose, the resulting behaviour is deterministic. Besides, they will normally agree on the maximum length of delay they can both tolerate before any of them is forced to perform any action due to maximal progress.

5. The negative premise in rule PAR-DEL may need to be justified since negative premises can introduce inconsistencies in our transition system. We can reason informally as follows. The rule PAR-DEL says that the component agents in a parallel composition can only delay if they cannot perform τ action. The only two rules that allow τ action are TAU-ACT and COM. We have two cases:

   (a) The rules PAR-DEL and TAU-ACT are consistent because in TAU-ACT the agent $\tau.P$ cannot delay and in PAR-DEL, if $P|Q$ can delay, they cannot perform τ action.

   (b) The rules PAR-DEL and TAU-ACT are also consistent. In COM, the premises dictate a pair of corresponding input and output actions in the component agents. Hence if $P|Q$ can perform τ action through COM, then it cannot delay through PAR-DEL. In PAR-DEL, if $P|Q$ can delay, then it cannot perform τ action. The choice of input/output actions and delay is well separated in both cases.

4.3 Labeled Transition System

We are aware that the conventional LTS neither represents our model faithfully nor preserves our model properties. In particular, the properties of timelock freeness, time determinacy and action persistency cannot be preserved, since an untimed LTS is not sufficiently expressive. Hence, we extend the conventional untimed LTS to include time and refer to it as *timed LTS* or *TLTS*.

Definition 6 *A timed labeled transition system, TLTS, over a set of labels $Act \cup \Delta$ is a tuple*

$$(\mathcal{Q}, \longrightarrow, \mathcal{T}, \Phi)$$

where

- $\mathcal{Q}$ *is the set of states*
- $\longrightarrow \subseteq \mathcal{Q} \times (Act \cup \Delta) \times \mathcal{Q}$ *is the transition relation*
- $\mathcal{T}$ *is a set of time variables that take values from the time domain* D
- Φ *is a set of timing constraints on* $\mathcal{T}$.

We will generally label time events with the time transition $\varepsilon(e)$ where e is some expression which may involve some time variables in $\mathcal{T}$. A transition labeled with $\varepsilon(e)$ is fired only if the values of the time variables involved in e satisfy their corresponding constraints specified in Φ. The concept is like the one in timed automata [AD94, Y92] but without start state and timer reset mechanism. The time variables together with the timing constraints specify the temporal behaviour of the system. Note that the time variables may take different values each time that transition is taken as long as they satisfy the timing constraints. For instance, reactive systems are not supposed to terminate and the time variables may take different values in each cycle.

TLTS will have properties according to our model properties:

Maximal progress: Whenever there is an edge coming out of a state labeled with τ, there will be no edge labeled with time event $\varepsilon(t)$ coming out of the same state, for any $t \in D$.

Timelock freeness: On the other hand, for each state where there is no edge coming out of it labeled with τ, there will be an edge labeled $\varepsilon(t)$ coming out of it, where $t \in D$.

Time determinacy: There is at most one edge labeled $\varepsilon(t)$ coming out of each state, where $t \in D$.

Time continuity: A period of time is split into disjoint sets and a state lasting for that period of time is split into an equivalent number of substates corresponding to each time set.

Action persistency: Whenever a time transition is taken in a state where some other actions are enabled, these enabled actions must be replicated in the subsequent states until they are fired.

Example: Vending Machine

Let's illustrate TLTS with the vending machine taken from [W90].

$$\begin{aligned} \mathrm{M}_0 &\stackrel{\text{def}}{=} money.\mathrm{M}_1 \\ \mathrm{M}_1 &\stackrel{\text{def}}{=} (0 \overset{2}{\triangleright} \overline{coffee} + 0 \overset{3}{\triangleright} \overline{tea}) \overset{30}{\triangleright} \mathrm{M}_0 \end{aligned}$$

The machine can sell either coffee or tea. It is first in state M_0 and waits for a buyer to input money. Then it evolves to state M_1. In state M_1, the machine will wait for 2 seconds and then offer coffee to the buyer. If the buyer does not take the coffee and wait one more second, the machine will also offer tea. So after 3 seconds, both coffee and tea will be offered. The buyer can choose either coffee or tea but not both. If the buyer does not make up his or her mind within 30 seconds, the machine will time out by not offering anything and go back to the initial state M_0 where the buyer would have lost a coin.

Due to our timeout semantics, the state M_1 has a slightly different behaviour than the one in [W90] and its possible transitions are stated formally as follows.

1. $\mathrm{M}_1 \overset{\varepsilon(t)\overline{coffee}}{\hookrightarrow\hookrightarrow} \mathrm{M}_0 \quad (2 \le t < 30)$
2. $\mathrm{M}_1 \overset{\varepsilon(t)\overline{tea}}{\hookrightarrow\hookrightarrow} \mathrm{M}_0 \quad (3 \le t < 30)$
3. $\mathrm{M}_1 \overset{\varepsilon(30)}{\hookrightarrow} \mathrm{M}_0$

The TLTS for the vending machine is shown in figure 3 where we split the total time span of 30 seconds after state M_1 into disjoint sets of shorter durations within each the system behaves differently. We have also named the intermediate states as M_2 and M_3.

1. When the machine starts up, it is in state M_0. The machine waits for the buyer to insert coin and is idle. Hence we have the edge looping back into M_0 labeled $\varepsilon(t1)$ with $t1 > 0$. When the buyer inserts a coin, the machine immediately goes to state M_1 through the edge labeled *money* in no time. Note that the action *money* is a controllable action (controlled by the buyer) and the time at which it is enabled and fired may be different due to the time event loop in this state. This is the reason why we have to separate the enabling and firing time.

2. In state M_1, the machine has to delay at least 2 seconds before it offers anything. Hence we have an edge coming out of M_1 labeled $\varepsilon(2)$ after which the machine goes into state M_2 where coffee is being offered. If the buyer takes the coffee, the machine goes back to the startup mode. Hence there is an edge labeled $\overline{coffee}$ out of state M_2 and goes back to state M_0.

3. If the buyer does not take coffee and waits longer, after 1 more second, tea will be offered as well. At this instant, it is 3 seconds after the machine enters state M_1. From the interval of 3 seconds to 29 seconds after state M_1, the machine offers both coffee and tea. (We are using discrete time. At 30 seconds, the machine will time out immediately and the buyer has no chance to take anything.) Hence there is an edge labeled $\varepsilon(t2)$ out of state M_2. The range of values that $t2$ can take will be $3-2 \le t2 \le 29-2$ or $1 \le t2 \le 27$ since $t2$ is measured relative to state M_2 which is already 2 seconds after state M_1. After a delay of $t2$ seconds, the machine goes to state M_3 where both coffee and tea are offered. Within this interval if the buyer takes either coffee or tea, the machine goes back to the startup mode. Hence, there is an edge labeled $\overline{coffee}$ and an edge labeled $\overline{tea}$ out of state M_3, both leading back to state M_0. Note that we have actually grouped a series of states between M_2 and M_3 (each separated by a time event of 1 time unit and all have the actions $\overline{coffee}$ and $\overline{tea}$ enabled) into just one state, M_3 because they all have the same behaviour, namely firing of the actions $\overline{coffee}$ and $\overline{tea}$. The time difference is absorbed in the time variable $t2$. This also illustrates why we separate the enabling and firing of controllable actions ($\overline{coffee}$ and $\overline{tea}$ in this case) where they allow time to pass between their enabling and firing.

4. At the instant of 30 seconds after state M_1, the machine will time out immediately and go back to the startup mode. Hence we have an edge labeled $\varepsilon(t3)$ out of state M_3 back to state M_0. The value that $t3$ can take is $t3 = 30 - 2 - t2$ or $t3 = 28 - t2$. Note how the value of $t3$ depends on the value of $t2$. This constraint is only used to make the total time span between state M_1 and timeout equal 30 seconds that is specified in

the specification of the machine. However, if the machine ever times out, then $t3 = 1$ because the waiting time when both coffee and tea are offered is absorbed in the time variable $t2$. The machine only times out when $t2 = 27$ which is the last instant the buyer can get anything. This will make $t3 = 1$. Note that if the buyer actually selects a coffee or tea in state M_3, then the machine will go back to state M_0 through the corresponding edge and it does not matter what value $t3$ will take in this case.

Hence, the TLTS for the vending machine has the following components:

- A set of states $\mathcal{Q} = \{\mathrm{M}_0, \mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3\}$
- The set of names $\mathcal{N} = \{money, coffee, tea\}$
- The set of labels $Act \cup \Delta = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\} \cup \Delta$
- A set of time variables $\mathcal{T} = \{t1, t2, t3\}$
- A set of timing constraints $\Phi = \{t1 > 0, 1 \leq t2 \leq 27, t3 = 28 - t2\}$

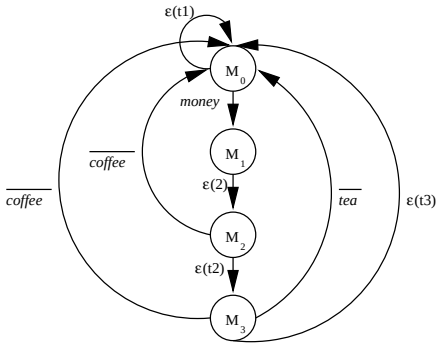

Figure 3: TLTS for the vending machine.

5 A simple example: a mobile streaming video player

In this section, we illustrate πRT-calculus with a simple example of a mobile streaming video player. The network topology is shown in figure 4.

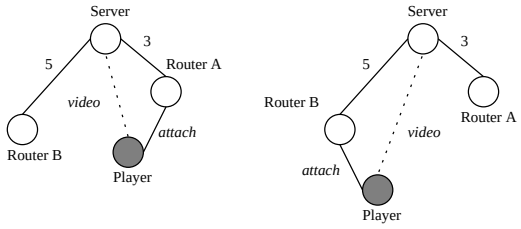

Figure 4: Network configuration for the mobile streaming video player.

This simplified network contains four nodes: the mobile streaming video player, two routers: router A and router B which serve as docking nodes attached by the player, and a video server that provides the video stream. The dotted line is the logical channel named "video" between the player and the server. The solid lines are the physical connections between nodes where the logical channel follows. The number 5 and 3 are the transmission delays in time unit of the link between router B and the server, and of the link between router A and the server, respectively. The channel "attach" is the physical as well as logical link between the mobile player and one of the attached routers. The player can get and play video frames from the server only when it attaches to one of the routers since the video frames have to follow the physical links. Assuming that the transmission delays between the player and the attached router is negligible, then the transmission delays between the mobile player and the server will effectively be the same as the transmission delays between the attached router and the server. Figure 4 shows the mobile player attached to router A on the left, and to router B on the right.

Figure 5 shows the ports of the player. Assuming the player is detached from any router in the beginning, the player has to attach to one of the routers through the "attach" port before receiving and playing video frames. The "attach" port receives from the router the transmission delay between the router and the server and another link "rel" that is used to release the attached router when the player moves away. After it is attached to one of the routers, the player idles for the specified amount of time received from the router since the first video frame ever to be received at this location is still in transit in the network. After the first delay, the player starts receiving the video frames through the "video" port. Note that the subsequent frames will follow immediately since the stream is continuous. After the player receives a frame, it decompresses and processes it. We model this processing as τ and its duration as a short delay of 1 time unit. Then the player delivers the frame to the user through the "play" port and repeats the cycle. At any time when the player is receiving frames, it may move away. We model this movement as a choice here. Note that the choice between receiving the frames and moving away is a non-deterministic one. When the player moves, it attaches to another router and receives the new transmission delay and release link for the new router. Then it releases the old router and repeats the cycle of receiving video frames.

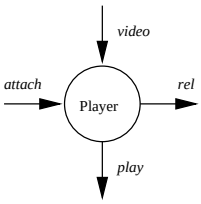

Figure 5: Ports of the video player.

$$
\begin{array}{lcl}
\text{Player} & \stackrel{\text{def}}{=} & attach(t, r).\text{Attached}(t, r) \\
\text{Attached}(t, r) & \stackrel{\text{def}}{=} & 0 \overset{t}{\triangleright} \text{Playing}(r) \\
\text{Playing}(r) & \stackrel{\text{def}}{=} & video.(0 \overset{1}{\triangleright} \tau.\overline{play}.\text{Playing}(r)) + \\
 & & attach(t, r).\overline{rel}.\text{Attached}(t, r)
\end{array}
$$

Figure 6 shows the ports of the routers. When the player attaches to the router, the router transmits to the player through the "attach" port the transmission delay between itself and the server together with the "rel" link used to release it. Then it does nothing except waiting to be released. We assume that the router can only serve one player at a time. When the player moves away, it sends a signal through the "rel" link to the router. The router, upon receiving the signal through the "rel" port, will release its resources and wait for the next attaching player.

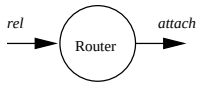

Figure 6: Ports of the routers.

$$\begin{aligned}
\text{Router}(t, rel) &\stackrel{\text{def}}{=} \overline{attach}\langle t, rel\rangle.rel.\text{Router}(t, rel)\\
\text{RouterA} &\stackrel{\text{def}}{=} \text{Router}(3, rel1)\\
\text{RouterB} &\stackrel{\text{def}}{=} \text{Router}(5, rel2)
\end{aligned}$$

There is only one port used by the server and is shown in figure 7. The server always just transmits video frames to the player through the "video" port. We assume the preparation of the video frames by the server is negligible compared to the transmission delay and so we do not model the time delay of the preparation of video frames. Also note that this is a synchronous model in the sense that the player does not send request for video frames and does not send acknowledgements to the server.

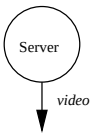

Figure 7: Port of the video server.

$$\text{Server} \stackrel{\text{def}}{=} \overline{video}.\text{Server}$$

The whole system is encoded from the user's point of view and is as follows:

$$\begin{aligned}
\text{System} \stackrel{\text{def}}{=}\ & (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
& (\text{Player}|\text{RouterA}|\text{RouterB}))
\end{aligned}$$

Suppose the player first attaches to router A. Router A transmits the transmission delay from the server which is 3 time units plus a release link to the player. The player then waits for the supplied delay of 3 time units:

$$\begin{aligned}
\text{System} &\stackrel{\tau}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (\text{Attached}(3, rel1)|\\
&\quad rel1.\text{Router}(3, rel1)|\text{RouterB}))\\
&\stackrel{\varepsilon(3)}{\hookrightarrow} \text{System}'
\end{aligned}$$

where

$$\begin{aligned}
\text{System}' \stackrel{\text{def}}{=}\ & (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
& (\text{Playing}(rel1)|\\
& rel1.\text{Router}(3, rel1)|\text{RouterB}))
\end{aligned}$$

After the transmission delay, the player gets the first video clip, decodes and processes it for 1 time unit and then plays the clip to the user. After which the player repeats the cycle of getting the subsequent video clips and playing:

$$\begin{aligned}
\text{System}' &\stackrel{\tau}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (0 \stackrel{1}{\triangleright} \tau.\overline{play}.\text{Playing}(rel1)|\\
&\quad rel1.\text{Router}(3, rel1)|\text{RouterB}))\\
&\stackrel{\varepsilon(1)}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (\tau.\overline{play}.\text{Playing}(rel1)|\\
&\quad rel1.\text{Router}(3, rel1)|\text{RouterB}))\\
&\stackrel{\tau}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (\overline{play}.\text{Playing}(rel1)|\\
&\quad rel1.\text{Router}(3, rel1)|\text{RouterB}))\\
&\stackrel{\overline{play}}{\hookrightarrow} \text{System}'
\end{aligned}$$

Note that the action $\overline{video}$ in the server is a controllable action. Even though it is enabled all the time, the firing of which depends on the player (when the correspondng *video* input action is enabled in the player). Once they are all enabled, they constitute a τ action in the system and will fire immediately as the first transition in the above. On the other hand, the processing of video frames in the player is modeled as a τ action and is non-controllable. Once it is enabled, it will fire without further time progress. This is illustrated in the second and third transitions above.

Suppose at some stage, the player moves away from router A and attaches to router B. Router B transmits the new transmission delay from the server which is now 5 time units plus a release link to the player. After attaching to router B, the player releases router A by sending a signal along the release link previously supplied by router A. The player then waits for the new supplied delay of 5 time units before receiving the stream again:

$$\begin{aligned}
\text{System}' &\stackrel{\tau}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (\overline{rel1}.\text{Attached}(5, rel2)|\\
&\quad rel1.\text{Router}(3, rel1)|\\
&\quad rel2.\text{Router}(5, rel2)))\\
&\stackrel{\tau}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (\text{Attached}(5, rel2)|\\
&\quad \text{RouterA}|rel2.\text{Router}(5, rel2)))\\
&\stackrel{\varepsilon(5)}{\hookrightarrow} \text{System}''
\end{aligned}$$

where

$$\begin{aligned}
\text{System}'' \stackrel{\text{def}}{=}\ & (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
& (\text{Playing}(rel2)|\\
& \text{RouterA}|rel2.\text{Router}(5, rel2)))
\end{aligned}$$

After the transmission delay, the player can start receiving and playing the stream:

$$\begin{aligned}
\text{System}'' &\stackrel{\tau}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (0 \stackrel{1}{\triangleright} \tau.\overline{play}.\text{Playing}(rel2)|\\
&\quad \text{RouterA}|rel2.\text{Router}(5, rel2)))\\
&\stackrel{\varepsilon(1)}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (\tau.\overline{play}.\text{Playing}(rel2)|\\
&\quad \text{RouterA}|rel2.\text{Router}(5, rel2)))\\
&\stackrel{\tau}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (\overline{play}.\text{Playing}(rel2)|\\
&\quad \text{RouterA}|rel2.\text{Router}(5, rel2)))\\
&\stackrel{\overline{play}}{\hookrightarrow} \text{System}''
\end{aligned}$$

When the player moves away from router B and back to router A again:

$$\begin{aligned}
\text{System}'' &\stackrel{\tau}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (\overline{rel2}.\text{Attached}(3, rel1)|\\
&\quad rel1.\text{Router}(3, rel1)|\\
&\quad rel2.\text{Router}(5, rel2)))\\
&\stackrel{\tau}{\hookrightarrow} (\nu\, video)(\text{Server}|(\nu\, attach, rel1, rel2)\\
&\quad (\text{Attached}(3, rel1)|\\
&\quad rel1.\text{Router}(3, rel1)|\text{RouterB}))\\
&\stackrel{\varepsilon(3)}{\hookrightarrow} \text{System}'
\end{aligned}$$

Note that after the first long transmission delay, the stream is flowing at a constant rate with a constant delay of 1 time unit between frames. It is only that each movement of the player incurs extra delay between frames. However, this is only a very rough approximation of the real situation. A more accurate analysis can be obtained using the stochastic approach [P97].

6 Conclusion

In this paper, we have introduced a timed extension to π-calculus, the πRT-calculus. We extend π-calculus in a natural way that allows transmission of time as ordinary data between processes. Most of the model properties are retained from the previous works on real-time algebra and we make a minimal extension to π-calculus by introducing only a timeout operator. We have presented the syntax and the operational semantics of πRT-calculus, as well as a set of structural congruence laws for the timeout operator. Model properties can be preserved by extending the conventional LTS to include time (and this resulted in the TLTS). Finally, we have presented two examples to illustrate the modeling power of πRT-calculus.

We will continue with the development of the algebra, and in particular, concentrate on developing appropriate bisimulation equivalence relations for the algebra.

References

[AD94] Rajeev Alur, David L. Dill, "*A Theory of Timed Automata*", Theoretical Computer Science 126(2):183-235, 1994.

[BB91] J. C. M. Baeten, J. A. Bergstra, "*Real Time Process Algebra*", Formal Aspects of Computing 3(2):142-188, 1991.

[CLN99] R. Cleaveland, G. Luttgen, V. Natarajan, "*Priority in Process Algebras*", In *Handbook of Process Algebra*, Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, (Ed.), Elsevier, 2001.

[DLP96] Pierpaolo Degano, Jean-Vincent Loddo, Corrado Priami, "*Mobile Processes with Local Clocks*", In *Proceedings of Workshop on Analysis and Verification of Multiple-Agent Languages*, Stockholm 1996, LNCS 1192, (M. Dam Ed.), Springer-Verlag.

[DS95] Jim Davies, Steve Schneider, "*A brief history of Timed CSP*", Theoretical Computer Science 138(2):243-271, 1995.

[FZ95] Colin Fidge, John Zic, "*An Expressive Real-Time CCS*", In *Proceedings of the Australasian Conference on Parallel and Real-Time Systems*, p365-372, 1995.

[H91] Hans A. Hansson, "*Time and Probability in Formal Design of Distributed Systems*", PhD thesis, Dept. of Computer Systems, Uppsala University, Sweden, 1991.

[HR95] Matthew Hennessy, Tim Regan, "*A Process Algebra for Timed Systems*", Information and Computation 117(2):221-239, 1995.

[L78] Leslie Lamport, "*Time, Clocks, and the Ordering of Events in a Distributed System*", Communications of ACM, 21(7):558-565, July 1978.

[M80] Robin Milner, "*A Calculus of Communicating Systems*", Springer-Verlag, 1980.

[M83] Robin Milner, "*Calculi for Synchrony and Asynchrony*", Theoretical Computer Science 25(3):267-310, 1983.

[M89] Robin Milner, "*Communication and Concurrency*", Prentice-Hall, 1989.

[M91] Robin Milner, "*The Polyadic π-calculus: a Tutorial*", In *Logic and Algebra of Specification* (F. L. Bauer, W. Brauer, H. Schwichtenberg, Ed.), p203-246, Springer-Verlag, 1993.

[M99] Robin Milner, "*Communicating and Mobile Systems: the π-calculus*", Cambridge University Press, 1999.

[MPW92] Robin Milner, Joachim Parrow, David Walker, "*A Calculus of Mobile Processes*" Part I&II, Information and Computation 100(1):1-40,41-77, 1992.

[MT90] F. Moller and C. Tofts, "*A Temporal Calculus of Communicating Systems*", Proceedings of CONCUR'90, LNCS 458, p401-415, 1990.

[NS91] Xavier Nicollin, Joseph Sifakis, "*An Overview and Synthesis on Timed Process Algebras*", Proceedings of CAV 91, LNCS 575, p376-398, 1991.

[NS94] Xavier Nicollin, Joseph Sifakis, "*The Algebra of Timed Processes ATP: Theory and Application*", Information and Computation 114(1):131-178, 1994.

[P00] Joachim Parrow, "*An Introduction to the π-calculus*", `http://www.it.kth.se/~joachim/intro.ps`, 2000.

[P95] Corrado Priami, "*Stochastic π-calculus*", The Computer Journal 38(7):578-589, 1995.

[P97] Corrado Priami, "*Qualitative and Quantitative Analysis of Mobile Systems*", `http://citeseer.nj.nec.com/priami97qualitative.html`, 1997.

[RR88] G. M. Reed, A. W. Roscoe, "*A Timed Model for Communicating Sequential Processes*", Theoretical Computer Science 58(1-3):249-261, 1988.

[S91] Steve Schneider, "*An Operational Semantics for Timed CSP*", Programming Research Group, Oxford University, UK, Feb 1991.

[S96] Steve Schneider, "*Specification and Verification in Timed CSP*", In *Real-time Systems Specification, Verification and Analysis* (M. Joseph Ed.), chapter 6, p147-181, Springer-Verlag, 1996.

[W90] Y. Wang, "*Real-Time Behaviour of Asynchronous Agents*", Proceedings of CONCUR'90, LNCS 458, p502-520, 1990.

[W91a] Y. Wang, "*CCS + Time = an Interleaving Model for Real Time Systems*", Proceedings of ICALP'91, p217-228, 1991.

[W91b] Y. Wang, "*A Calculus of Real Time Systems*", PhD thesis, Dept. of Computer Science, Chalmers University of Technology, 1991.

[Y92] Sergio Yovine, "*Compiling Timed Algebras into Timed Automata*", Proceedings of XVIII Conferencia Latinoamericana de Informatica, PANEL'92, 1992.