プログラミング-言語・基礎・実践- 11－7
(1993. 3. 10)

A Timed Process Calculus Based on Distributed Time

Ichiro Satoh
satoh@mt.cs.keio.ac.jp

Tokoro Mario
mario@mt.cs.keio.ac.jp

Department of Computer Science, Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama, 223, Japan

This paper proposes a formalism for reasoning about the notion of local time in distributed systems. The formalism is an extension of CCS with local time and behavior depending the passage of time. It can explicitly represent local time measured by clocks having different time granularities and precisions. It can also model processes with different local times. Also, we define a timed bisimulation over processes with local clocks. It equates processes only if their behaviors are completely matched and their timing and clocks are slightly different. In this paper we present the basic construction of the formalism and some illustrative examples.

分散的時間性に基づくプロセス計算について

佐藤 一郎　　所 真理雄

慶應義塾大学 理工学研究科 計算機科学専攻

本論文では、分散計算における局所的 (非全域的) な時間性が表現可能な形式系をプロセス計算 (代数) に基づいて構築する。これは、プロセス計算の体系の一つである CCS に局所的な時間の表現性と時間に依存した動作の表現性を拡張した形式系である。これにより、測定精度や測定単位が異なる時計をもつプロセッサ上のプロセスの挙動や、それらの間の相互作用における動作や時間性が明示的に表現できるようになる。さらに、双模倣性 (Bisimulation) の概念を拡張することにより、局所的な時間性を考慮したプロセスの等価性を定式化する。本論文では、この形式系の構文と意味論を定義し、さらにその記述例を与える。

1 Introduction

In distributed systems, processors do not share a primary memory and are loosely coupled by communication networks with inherent delay. Processors cannot share any unique global view, such as a reference clock among them, because the delay forces processors to give only past information to other processors. However, in many real distributed systems, processors need real-time clocks to be used for detection of failures and for time-critical responses, and must cooperate with each other according to their own local clocks instead of a global clock. However, the precisions and granularities of different local clocks do not always coincide. In this case, if the differences of time precisions and granularities cannot be ignored, the cooperation may lead to failure. In order to develop correct distributed systems and correct program for the systems, we must take the difference between the processors' local clocks into consideration. It is very useful to support a computing model for describing and verifying such local time properties and functional logical behaviors in distributed computing.

However, most computing models for distributed systems are diverted from computing models for concurrent computing and cannot essentially represent local time properties in distributed systems. Some researchers have explored methods and algorithms for agreement on processes' time, such as virtual time [7] and clock synchronization [8, 5]. Unfortunately these are not intended to model the functional behavior as well as the time properties of distributed computing. Also, in [2] the authors present how to deal with different time granularities based on a temporal logic with object-oriented structure [11] but they cannot deal with different time precisions.

The goal of this paper is to investigate a formalism for reasoning about distributed computing, especially local time properties of distributed computing. The formalism is based on a timed extended process calculus, RtCCS [13] which is an extension of CCS [9] with the notion of global time.

However, RtCCS is essentially based on a global time so that it cannot represent the notion of local time in distributed systems. Also, other timed extended process calculi cannot represent local time properties [3, 10, 17]. There have been some models for distributed computing based on process calculi [1, 4, 6], but they are extended CCS with the concept of process locality and not local time. In this paper, we extend RtCCS with the notion of local time and then introduce a process calculus for distributed systems, called *DtCCS* (*Distributed timed Calculus of Communicating Systems*). Furthermore, we develop a theoretical proof technique for distributed systems based on DtCCS.

The organization of this paper is as follows: in the next section, we informally introduce our approach to represent processes with local clocks. In Section 3, we define the syntax and the operational semantics of DtCCS. Section 4 presents the concept of timed bisimilarity based on local time and studies its basic properties. The final section contains some concluding remarks. However, this paper is an abbreviated version of [15] and we leave the further details on DtCCS and all the proofs to [15].

2 Distributed Timed CCS

In this section, we describe our basic idea of time in distributed computing and a brief introduction to our formalism, and then we formally define the formalism.

First, we present our two assumptions for time in distributed computing.

- All processors in a distributed system share a global time.
- A clock in each processor may measure the global time according to its own time granularity [1] and precision.

This is because, if the relative motion of the processors is negligible, then all the proces-

[1] The interval of consecutive time instants.

sors must share the same physical global time. Actual clocks read their own current time by translations from the physical global time into their own time coordinate. Consequently, the instantaneous values of different processors' clocks are different from one another. The processors with clocks may not have the same time although they share the same global time.

We develop our formalism based on the above assumptions. We first introduce a conceptual global time and then give processes local clocks which can measure the global time in their own time granularities and precisions.

In developing our formalism, called DtCCS, we first take up a timed extended process calculus RtCCS [13] as a semantic framework to represent the notion of global time. Second, in order to represent the above notion of local time, we extend it with the ability of representing clocks with different granularities and precisions. Therefore, we introduce a local evaluating function to the clock in RtCCS. The function is given in each local processor and translates local time which local processors' clocks measure into a corresponding global time. By using the function, processes on local times are translated into processes on a global time.

2.1 Time Domain

In our formalism, time is represented as a time domain. A time domain is a set of time instantaneous values and is formally defined as follows:

Definition 2.1 Let $\mathcal{T}$ denote the set of the positive integers including 0. We call $\mathcal{T}$ a *time domain.*

$$\mathcal{T} \equiv \mathcal{N} \cup \{0\}$$

∎

We assume a special time domain which can be measured by a conceptual perfect and finest clock. It is called *a global time domain* and is denoted as $\mathcal{T}_G$ and will sometimes be written as $\mathcal{T}$.

The time of any instant time in all local time domains can correspond to the time of an instant time in the global time domain. The linkages between the local time domains and the global time domain are related by a mapping, called a *clock mapping*, or simply a *clock*. The mapping shows how the length of the time interval of the local time domain is measured on the global time domain.

Definition 2.2 Let $\mathcal{T}_\ell$ be a local time domain. A clock mapping $\theta : \mathcal{T}_\ell \rightarrow \mathcal{T}_G$ is defined recursively by the following mapping. For all $t_\ell \in \mathcal{T}_\ell$,

$$\theta(t_\ell) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t_\ell = 0 \\ \theta(t_\ell - 1) + \delta_\ell & \text{if } t_\ell > 0 \end{cases}$$

where δ_{min}, δ_{max}, $\delta_\ell \in \mathcal{T}_G$, $\delta_{min} \leq \delta_\ell \leq \delta_{max}$. We will sometimes abbreviate the definition of θ as $\theta(t_\ell) \stackrel{\text{def}}{=} \delta_\ell t_\ell$ if $\delta_{min} = \delta_{max}$. ∎

δ_ℓ corresponds to an interval of the local time domain mapped into the global time domain. δ_{min} is the lower bound of that interval and δ_{max} the upper bound of that interval. As a result, in the local time domain of an accurate clock, the clock mapping is injective, whereas in the local time domain of an inaccurate clock the clock mapping is not.
In this formalism, we assume that any events which are executed between two consecutive time instants are treated as events which are executed at the faster time instant. Here, we present the inverse of clock mapping.

Definition 2.3 The inverse mapping of θ: $\mathcal{T}_\ell \rightarrow \mathcal{T}_G$ is defined as follows:

$$\theta^{-1}(t_G) \equiv \{\ t_\ell \mid \mathbf{min}\ \{\theta(t_\ell)\} \leq t_G < \mathbf{max}\ \{\theta(t_\ell + 1)\}\ \}$$

where $t_\ell \in \mathcal{T}_\ell$, $t_G \in \mathcal{T}_G$, and $\{\theta(t)\} \equiv \{\ t_G \in \mathcal{T}_G \mid \theta(t) = t_G\ \}$ ∎

$\{\theta(t)\}$ means the whole time values can be evaluated in $\theta(t)$. Note that the meaning of this definition is different from that of mathematical inverse function.

2.2 The DtCCS Language

In this subsection, we present the syntax of DtCCS by using that of RtCCS. First we informally introduce RtCCS before the definition of the syntax.

RtCCS is an extension of Milner's CCS [9] by introducing a tick action and a timeout operator. The tick action is described as $\surd$ and is a synchronous broadcast message over all processes. It corresponds to the passage of one time unit. The advance of time can be represented as a sequence of tick actions and is viewed as discrete time. The timeout operator described as $\langle E_1, E_2\rangle_t$. $\langle E_1, E_2\rangle_t$ denotes a process that after t time units becomes E_1, unless E_2 performs any actions prior to that. Intuitively $\langle E_1, E_2\rangle_t$ behaves as process E_1 if E_1 can execute an initial transition within t units of time, whereas $\langle E_1, E_2\rangle_t$ behaves as process E_2 if E_1 does not perform any action within t units of time.

Notation and Syntax

We presuppose that $\mathcal{A}$ is a set of communication action names and $\overline{\mathcal{A}}$ the set of co-names. Let a,b,... range over $\mathcal{A}$ and $\overline{a}$,$\overline{b}$,... over $\overline{\mathcal{A}}$. An action $\overline{a}$ is the complementary action of a, and $\overline{\overline{a}} \equiv a$. Let $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be the set of action labels and ℓ,ℓ',... range over $\mathcal{A}$. Let τ denote a silent action which is considered to be unobservable outside a system. Let $\surd$ denote a *tick* action which represents the passage of one time unit. Finally let $Act \equiv \mathcal{A} \cup \overline{\mathcal{A}} \cup \{\tau\}$ ranged over by α,β,..., and $Act_T \equiv Act \cup \{\surd\}$ ranged over by μ, ν,....

Definition 2.4 The set $\mathcal{E}$ of RtCCS process expressions, ranged over by $E, E_1, E_2, \ldots$ is the smallest set which contains the following expressions.

E	::=	$\mathbf{0}$	(Deadlock Process)
	|	X	(Process Variable)
	|	$\alpha.E$	(Action Prefix)
	|	$E_1 + E_2$	(Summation)
	|	$E_1 \mid E_2$	(Composition)
	|	$E[f]$	(Relabeling)
	|	$E \setminus L$	(Restriction)
	|	$\mathbf{rec}\, X : E$	(Recursion)
	|	$\langle E_1, E_2\rangle_t$	(Timeout)

where $t \in \mathcal{T}_G$, $f \in Act \to Act$ and $L \subseteq Act$. We assume that $f(\tau) = \tau$, $f(\surd) = \surd$, and that X is always *guarded* [2]. We shall often use the more readable notation $X \stackrel{\text{def}}{=} E$ instead of $\mathbf{rec}\, X : E$. ∎

In the following, in order to clarity our exposition, we will divide process expressions $\mathcal{E}$ into two groups: expressions describing behavior of a processor and expressions describing interaction among processors with different clocks. We restrict the syntax of the former group to a subset of $\mathcal{E}$ which does not allow process expressions with parallelism. This is because concurrent executions on the same clocks can be reduced to nondeterministically sequential executions by using the expansion rules [3] for RtCCS. The syntax of the latter group consists of parallel, restriction, relabeling operator, and a special operator, $[\![S]\!]_\theta$, to represent the dependency of local time.

Definition 2.5 The set $\mathcal{S}$ of sequential process expression ranged over by S, S_1, S_2 is the subset of process expressions $\mathcal{E}$ is generated by the following grammar:

$$S ::= \mathbf{0} \mid \alpha.S \mid X \mid S_1 + S_2 \mid \mathbf{rec}\, X : S \mid \langle S_1, S_2\rangle_t$$

where t is an element of a local time domain. ∎

A process executed on a local clock θ is represented as a sequential process expression

[2] X is *guarded* in E if each occurrence of X is only within some subexpressions $\alpha.E'$ in E; c.f. *unguarded* expressions, e.g. $\mathbf{rec}\, X : X$ or $\mathbf{rec}\, X : X + E$.

[3] See Corollary 1 in [13]

supplied with a clock translation mapping $[\![\cdot]\!]_\theta$. The syntax of expressions for interaction among processors with different local clocks are defined by the following grammar:

Definition 2.6 The set $\mathcal{P}$ of process expressions on local clocks, ranged over by P, P_1, P_2, is defined by the following grammar:

$$P \ ::= \ [\![S]\!]_\theta \ | \ P_1|P_2 \ | \ P[f] \ | \ P \setminus L$$

where S is an element of the set of closed expressions in $\mathcal{S}$ and θ is a clock mapping ($\theta : \mathcal{T}_\ell \to \mathcal{T}_G$). Process expressions on $\mathcal{P}$ will sometimes be called DtCCS process expressions. ∎

$[\![S]\!]_\theta$ allows sequential process expressions on local clock θ to be translated into RtCCS expressions on global time domain $\mathcal{T}_G$. In the above definition, we could give $[\![\cdot]\!]_\theta$ as a high order function, i.e. $[\![P]\!]_\theta$ instead of $[\![S]\!]_\theta$. However, to simplify our formalism, we restrict $[\![\cdot]\!]_\theta$ to a first order function in this paper.

2.3 Semantics

As we noted already, the semantics of DtCCS is given in two steps. First, DtCCS processes are translated into RtCCS process expressions based on a global clock. Second, the semantics of the RtCCS expressions are given by structural transition rules in Figure 1.

The semantics of RtCCS

In order to define the semantics of DtCCS, we should first define the semantics of RtCCS. RtCCS is a labeled transition system $\langle \mathcal{E}, Act_\mathcal{T}, \{ \xrightarrow{\mu} \mid \mu \in Act_\mathcal{T} \} \rangle$ where $\xrightarrow{\mu}$ is a transition relation ($\xrightarrow{\mu} \subseteq \mathcal{E} \times \mathcal{E}$). The transition relation $\xrightarrow{\mu}$ is defined by structural induction and is the smallest relations satisfying the rules in Figure 1.

The semantics of DtCCS

All the syntactical constructions of $\mathcal{P}$ except $[\![S]\!]_\theta$ are included in $\mathcal{E}$ and can be directly

$$\mathbf{ACT_0}: \ \frac{-}{\alpha.E \xrightarrow{\alpha} E} \qquad \mathbf{ACT_1}: \ \frac{-}{\ell.E \xrightarrow{\surd} \ell.E} \qquad \mathbf{ACT_2}: \ \frac{-}{\mathbf{0} \xrightarrow{\surd} \mathbf{0}}$$

$$\mathbf{SUM_0}: \ \frac{E \xrightarrow{\alpha} E'}{E+F \xrightarrow{\alpha} E'} \qquad \mathbf{SUM_1}: \ \frac{F \xrightarrow{\alpha} F'}{E+F \xrightarrow{\alpha} F'} \qquad \mathbf{SUM_2}: \ \frac{E \xrightarrow{\surd} E', \ F \xrightarrow{\surd} F'}{E+F \xrightarrow{\surd} E'+F'}$$

$$\mathbf{COM_0}: \ \frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F} \qquad \mathbf{COM_1}: \ \frac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'} \qquad \mathbf{COM_2}: \ \frac{E \xrightarrow{a} E', \ F \xrightarrow{\bar{a}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

$$\mathbf{COM_3}: \ \frac{E \xrightarrow{\surd} E', \ F \xrightarrow{\surd} F', \ E|F \not\xrightarrow{\tau}}{E|F \xrightarrow{\surd} E'|F'}$$

$$\mathbf{RES_0}: \ \frac{E \xrightarrow{\mu} E', \ \mu \notin L \cup \overline{L}}{E \setminus L \xrightarrow{\mu} E' \setminus L} \qquad \mathbf{RES_1}: \ \frac{E \xrightarrow{\surd} E'}{E \setminus L \xrightarrow{\surd} E' \setminus L}$$

$$\mathbf{REL}: \ \frac{E \xrightarrow{\mu} E'}{E[f] \xrightarrow{f(\mu)} E'[f]} \qquad \mathbf{REC}: \ \frac{E\{\mathbf{rec}\, X : E/X\} \xrightarrow{\mu} E'}{\mathbf{rec}\, X : E/X \xrightarrow{\mu} E'}$$

$$\mathbf{TIME_0}: \ \frac{E \xrightarrow{\alpha} E', \ t > 0}{\langle E, F \rangle_t \xrightarrow{\alpha} E'} \qquad \mathbf{TIME_1}: \ \frac{E \xrightarrow{\surd} E', \ t > 0}{\langle E, F \rangle_t \xrightarrow{\surd} \langle E', F \rangle_{t-1}} \qquad \mathbf{TIME_2}: \ \frac{F \xrightarrow{\mu} F'}{\langle E, F \rangle_0 \xrightarrow{\mu} F'}$$

Figure 1: Inference Rules of RtCCS

translated into RtCCS. We will give the rules to map $[\![S]\!]_\theta$ into RtCCS process expressions according to the syntactic structure of S. The rules allow all DtCCS process expressions in $\mathcal{P}$ to be translated into RtCCS process expressions. The rules are defined by the syntactic mapping shown below.

Before describing the definition of $[\![\cdot]\!]_\theta$, we present the key idea of the mapping. In our formalism, the only primitive which can measure elapse time, a clock, is only the deadline time of the timeout operator. The deadline represents the time before the timeout operator goes into a timeout state, and is reduced by one whenever a unit time passes. The mapping rules translate the deadline time on a local time domain into a deadline time on the global time domain by evaluating clock mapping θ.

Definition 2.7 Let $\mathcal{T}_\ell$ be a local time domain. The mapping $[\![\cdot]\!]_\theta$ from $\mathcal{S} \times (\mathcal{T}_\ell \to \mathcal{T}_G)$ to $\mathcal{E}$ is recursively defined by the following syntactic rewriting rules. We assume that θ is a clock mapping from $\mathcal{T}_\ell$ to $\mathcal{T}_G$.

$$
\begin{array}{rcl}
[\![\mathbf{0}]\!]_\theta & \rightharpoonup & \mathbf{0} \\
[\![X]\!]_\theta & \rightharpoonup & X \\
[\![\alpha.S]\!]_\theta & \rightharpoonup & \alpha.[\![S]\!]_{\theta'} \\
[\![S_1 + S_2]\!]_\theta & \rightharpoonup & [\![S_1]\!]_{\theta_1} + [\![S_2]\!]_{\theta_2} \\
[\![\mathbf{rec}\, X : S]\!]_\theta & \rightharpoonup & \mathbf{rec}\, X : [\![S]\!]_{\theta'} \\
[\![\langle S_1, S_2 \rangle_t]\!]_\theta & \rightharpoonup & n \in \{\theta_1(t)\} : \langle [\![S_1]\!]_{\theta_2}, [\![S_2]\!]_{\theta'} \rangle_n
\end{array}
$$

where $\theta', \theta_1, \theta_2$ are clock mappings from $\mathcal{T}_\ell$ to $\mathcal{T}_G$ and satisfy $\{\theta\} = \{\theta'\}$, and $\{\theta_1\}$, $\{\theta_2\}$ and $\forall t_\ell \in \mathcal{T}_\ell : \theta_1(t_\ell) = \theta_2(t_\ell)$. ∎

We explain the intuitive meaning of some mapping rules. In $[\![\alpha.E]\!]_\theta \rightharpoonup \alpha.[\![E]\!]_\theta$ an unpredictable synchronization time for α is directly translated into an unpredictable time on the global time domain. The mapping rule for the summation shows that all alternative processes in a processor share the same clock.

The above mapping rules and the transition relations in Figure 1 provide the basic mechanism of computation in our formalism. We will give some examples of translation by the mapping rules.

Example 2.8 *Examples of mapping* $[\![\cdot]\!]_\theta$
Consider process $\langle a.P, b.Q \rangle_2$. We show its mapping in two cases: when executed in a processor with an accurate clock and when executed in a processor with an inaccurate clock.

(i) *Mapping with an accurate clock*

Consider the execution of $\langle a.P, b.Q \rangle_2$ on a clock whose time granularity is always three time units. The clock is given as $\theta(t) = 3t$ and a computation of the process is as follows:

$$
\begin{array}{rcl}
[\![\langle a.P, b.Q \rangle_2]\!]_\theta & \longrightarrow & \langle [\![a.P]\!]_\theta, [\![b.Q]\!]_\theta \rangle_6 \\
& \longrightarrow & \langle a.[\![P]\!]_\theta, [\![b.Q]\!]_\theta \rangle_6 \\
& (\stackrel{\checkmark}{\longrightarrow})^6 & [\![b.Q]\!]_\theta \\
& \longrightarrow & b.[\![Q]\!]_\theta
\end{array}
$$

(ii) *Mapping with an inaccurate clock*

Consider the execution of $\langle a.P, b.Q \rangle_2$ on a clock whose time granularity may nondetermimisitically be evaluated as either one of 2 or 3, 4. The clock is given by $\theta(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta(t-1) + \delta & \text{if } t > 0 \end{cases}$ where $\delta \in \{2, 3, 4\}$ and the computation of the process is as follows:

$$
[\![\langle a.P, b.Q \rangle_2]\!]_\theta \quad (\longrightarrow)^2 \quad \langle a.[\![P]\!]_\theta, b.[\![Q]\!]_\theta \rangle_n
$$

where $n \in \{4, 5, 6, 7, 8\}$

The clock θ in (ii) corresponds to a clock having the granularity of 3 time units and the error of ± 1 time units. Thus, the nondermimism in the definition of clock mapping θ allows us to represent non perfect clocks, such as physical clocks which may essentially drift. ∎

3 Local Timed Bisimulation

In this section, in order to establish a proof technique for distributed systems, we define a local timed bisimulation. In the earlier paper [13], the author provided timed equivalences in which timed equivalent processes must completely match their time properties

as well as their functional behaviors. However, in distributed systems any two distributed processes may not temporally match one another because the clocks of the processors never run at exactly the same rate. It is natural that two processes on different processors can be treated as equivalent processes only if their behaviors are completely matched and their timing are slightly different. We develop such an appropriate experimenting technique for distributed processes by extending RtCCS's timed bisimulation with the notion of rough time.

In order to formulate the observational bisimulation for our calculus, we first define the two following transition relations due to the unobservability of τ.

Definition 3.1

(i) $P \stackrel{\mu}{\Longrightarrow} Q \stackrel{\text{def}}{=} P(\stackrel{\tau}{\longrightarrow})^* \stackrel{\mu}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^* Q$

(ii) $P \stackrel{\widehat{\mu}}{\Longrightarrow} Q \stackrel{\text{def}}{=} P(\stackrel{\tau}{\longrightarrow})^* \stackrel{\mu}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^* Q$ if $\mu \neq \tau$ and otherwise $P(\stackrel{\tau}{\longrightarrow})^* Q$. ∎

Definition 3.2 A binary relation $\mathcal{R}$ is a *local timed bisimulation* if $(P,Q) \in \mathcal{R}_\theta$ implies, for all $\alpha \in Act$,

(1) $\forall m \in \mathcal{T},\ \forall P'\colon P(\stackrel{\surd}{\Longrightarrow})^m \stackrel{\alpha}{\Longrightarrow} P' \supset \exists n \in \mathcal{T},\ \exists Q'\colon Q(\stackrel{\surd}{\Longrightarrow})^n \stackrel{\widehat{\alpha}}{\Longrightarrow} Q' \wedge \{\theta^{-1}(m)\} \cap \{\theta^{-1}(n)\} \neq \emptyset \wedge (P',Q') \in \mathcal{R}_\theta$.

(2) $\forall n \in \mathcal{T},\ \forall Q'\colon Q(\stackrel{\surd}{\Longrightarrow})^n \stackrel{\alpha}{\Longrightarrow} Q' \supset \exists m \in \mathcal{T},\ \exists P'\colon P(\stackrel{\surd}{\Longrightarrow})^m \stackrel{\widehat{\alpha}}{\Longrightarrow} P' \wedge \{\theta^{-1}(m)\} \cap \{\theta^{-1}(n)\} \neq \emptyset \wedge (P',Q') \in \mathcal{R}_\theta$.

We let "$\approx_\theta$" denote the largest local timed bisimulation, and call P and Q *local timed bisimilar* if $P \approx_\theta Q$. Also if $\theta(t) \stackrel{\text{def}}{=} t$, we let $\approx_{\mathbf{id}}$ (or simply $\approx_{\mathcal{T}}$) denote $\approx_\theta$. ∎

Intuitively, if two processes P and Q are local timed bisimilar, they cannot be distinguished from one another in their visible behaviors and in the timing of their behaviors, by an external observer having clock θ.

Proposition 3.3 Let $S_1, S_2 \in \mathcal{S}$, then $[\![S_1]\!]_\theta \approx_\theta [\![S_2]\!]_\theta$ if $S_1 \approx_{\mathbf{id}} S_2$

This proposition shows a relationship between that the local time bisimilarity on a clock and the clock translation mapping of the clock.

Proposition 3.4 Let $P_1, P_2 \in \mathcal{P}$: $P_1 \approx_\theta P_2$, then

(1) $\alpha.P_1 \approx_\theta \alpha.P_2$
(2) $P_1 \setminus L \approx_\theta P_2 \setminus L$
(3) $P_1[f] \approx_\theta P_2[f]$
(4) $P_1|Q \approx_\theta P_2|Q$

where $Q \in \mathcal{P}$ contains no timeout operator.

Thus, the bisimilarity is not preserved by the parallel operator because Q is restricted to an untimed process. However, only if we restrict the clock of the bisimilarity to a clock with no drift, the parallel operator preserves the bisimilarity.

Proposition 3.5 Let clock θ is an accurate clock (without drift), If $\forall P_1, P_2 \in \mathcal{P}$, $\forall S \in \mathcal{S}$: $P_1 \approx_\theta P_2$, then

$$P_1|[\![S]\!]_\theta \approx_\theta P_2|[\![S]\!]_\theta$$

This proposition states that if a process with an accurate clock cannot distinguish between two processes, the process never notices the difference of the cooperation with either processes.

4 Examples

In order to illustrate how to describe and verify distributed systems in our formalism, we present a simple example

Example 4.1 *Interaction between distributed processes*

We describe the cooperation between a client process and a server process on different processors.

- The client sends a request message ($\overline{req}$) and then waits for a return message (ret). If the return message cannot be received within 6 time units, then it sends the request message again.

- Upon reception of a request message (req), the server process sends a return message ($\overline{ret}$) after an internal execution for 5 time units.

These processes are denoted as follows:

$$Client \stackrel{\text{def}}{=} \overline{req}.\langle ret.\mathbf{0}, Client\rangle_6$$
$$Server \stackrel{\text{def}}{=} req.\langle \mathbf{0}, \overline{ret}.Server\rangle_5$$

The cooperation between the processes is described as the parallel composition of $Client$ and $Server$.

(i) First, we assume that both the processes are executed on the same processor with a clock. In this case, the cooperation is described as $[\![Client]\!]_\theta | [\![Server]\!]_\theta$ (where $\theta(t) \stackrel{\text{def}}{=} 3t$) and its computation is described as below:

$$[\![Client]\!]_\theta \ (\longrightarrow)^* \ \overline{req}.\langle ret.\mathbf{0}, [\![Client]\!]_\theta\rangle_{18}$$
$$[\![Server]\!]_\theta \ (\longrightarrow)^* \ req.\langle \mathbf{0}, \overline{ret}.[\![Server]\!]_\theta\rangle_{15}$$

$$([\![Client]\!]_\theta | [\![Server]\!]_\theta) \setminus \{req, ret\}$$
$$\stackrel{\tau(req)}{\longrightarrow} (\stackrel{\surd}{\longrightarrow})^{15} \stackrel{\tau(ret)}{\longrightarrow} (\mathbf{0}|[\![Server]\!]_\theta) \setminus \{req, ret\}$$

In the above interaction, the client can receive the return message before it goes to timeout.

(ii) Next we assume that the two processes are allocated to different processors. The client process is executed by a processor with clock θ_c, whose time granularity is from 4 to 6 time units and the server process by one with clock θ_s, whose time granularity is from 3 to 5 time units. θ_c and θ_s are defined as follows:

$$\theta_c(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta_c(t-1) + \delta_c & \text{if } t > 0 \end{cases}$$
where $\delta_c \in \{4, 5, 6\}$

$$\theta_s(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta_s(t-1) + \delta_s & \text{if } t > 0 \end{cases}$$
where $\delta_s \in \{3, 4, 5\}$

The cooperation between the server and client process is constructed by the following parallel composition:

$$([\![Client]\!]_{\theta_c} | [\![Server]\!]_{\theta_s}) \setminus L$$
where $L \stackrel{\text{def}}{=} \{req, ret\}$

This means that the processes are located on different processors because the clock mapping θ_c and θ_s are independent from one another. By $[\![\cdot]\!]_\theta$ mapping rules, the client and the server are mapped into the global time domain as follows:

$$[\![Client]\!]_{\theta_c} \ (\longrightarrow)^* \ \overline{req}.\langle ret.\mathbf{0}, [\![Client]\!]_{\theta_c}\rangle_{t_c}$$
where $t_c = \{\theta_c(6)\}$ $(= \{24, 25, \ldots 36\})$
$$[\![Server]\!]_{\theta_s} \ (\longrightarrow)^* \ req.\langle \mathbf{0}, \overline{ret}.[\![Server]\!]_{\theta_s}\rangle_{t_s}$$
where $t_s = \{\theta_s(5)\}$ $(= \{15, 16, \ldots 25\})$

From the definition of θ_c and θ_s, there are mutliple results for $\theta_c(6)$ and $\theta_s(5)$. For example, we present the computation in the case of $t_c = 24$ and $t_s = 25$.

$$([\![Client]\!]_{\theta_c} | [\![Server]\!]_{\theta_s}) \setminus L \stackrel{\tau(req)}{\longrightarrow} (\stackrel{\surd}{\longrightarrow})^{24}$$
$$([\![Client]\!]_{\theta_c} | \langle \mathbf{0}, \overline{ret}.[\![Server]\!]_{\theta_s}\rangle_1) \setminus L$$
$$\cdots \ (failure)$$

In the above case, the client goes to timeout before receiving a return message ret because of $t_c \leq t_s$.

Example 4.2 *Example of Local Timed Bisimulation*

We consider two server processes, $Server_A$ and $Server_B$, and a client process, $Client$. The behavior of the server and client processes are already described in Example 4.1 but we assume that the execution time of $Server_A$ is 4 time units and that of $Server_B$ 5 time units. These processes are described as follows:

$$Server_A \stackrel{\text{def}}{=} req.\langle \mathbf{0}, \overline{ret}.Server_B\rangle_4$$
$$Server_B \stackrel{\text{def}}{=} req.\langle \mathbf{0}, \overline{ret}.Server_B\rangle_5$$
$$Client \stackrel{\text{def}}{=} \overline{req}.\langle ret.\mathbf{0}, Client\rangle_3$$

(i) By using the local timed bisimilarity which depends on an accurate clock $\theta(t) \stackrel{\text{def}}{=} 6t$, we verify the behaviors and time properties of the two servers.

$$Server_A \approx_\theta Server_B$$
$$(\text{ cf. } Server_A \not\approx_{\text{id}} Server_B \text{ })$$

The above relation means that an observer dependent on a clock whose time granularity is six time units, cannot detect any time difference within six time units. As a result, the observer cannot distinguish the two server processes although they are different.

(ii) Let the client be performed on a processor with clock θ. Then, from Proposition 3.5 the cooperation between the client and the servers is given as follows:

$$Server_A|[\![Client]\!]_\theta \approx_\theta Server_B|[\![Client]\!]_\theta$$

This example means that if observer on a processor having clock θ cannot detect the difference between two server, the client on the processor cannot detect the difference.

The above example shows that the bisimilarity can ignore slight temporal differences between two functionally equivalent processes so that it can equate two distributed processes whose time properties are not matched by the existence of the difference in the local clocks of the processes and it can equate real-time processes whose temporal requirement are not strict.

5 Concluding Remarks

We introduced a formalism to describe and verify distributed systems. In our earlier paper [13] we introduced RtCCS. However, since RtCCS was developed based on the global time, it cannot represent computations based on local time, such as distributed computing. In this paper we extended RtCCS with the notion of local time and developed a new formalism, called DtCCS. It can represent local time measured by clocks having different time granularities and precisions. It can also model the functional and temporal behaviors of processes on the same local time and interactions between processes on different local time. Therefore, it provide a formal framework for describing and analyzing distributed computing which consisted of distributed processes with different local times. We also presented a local timed bisimulation which defines an equivalent relation between processes whose functional behaviors are completely matched and whose timing are slightly different. Therefore, it can ignore the slight difference of local clocks and can equate processes on different local clocks. For further details the readers are referred to the full version of this paper [15].

Finally, we would like to point out some further issues. Although, in distributed computing asynchronous communication may seem more appropriate, the current formalism is based on synchronous communication. We are interested in extending DtCCS with asynchronous communication. Particularly, we believe that the study of the time properties for asynchronous communications will provide us with the concepts of programming languages for distributed systems. We are also interested in investigating whether our calculus and local timed bisimulation can be mathematically formulated in topological metric space.

The concept of our mapping local time into global time is independent on our formalism and RtCCS and The concept of our mapping local time into global time can be naturally adopted into other formalism based on the concept of global time, such as real-time temporal logic and timed Petri net.

References

[1] Castellani, H., Hennessy, M., *Distributed Bisimulation*, Journal of ACM, Vol.36, No.4, 1989.

[2] Corsetti, E., Montanari, A., and Ratto, E., *Dealing with Different Granularities in Formal Specifications of Real-Time Systems*,

Journal of Real-Time Systems, Vol.3, No.2, May, 1991.

[3] Hansson, H. and Jonsson, B., *A Calculus of Communicating Systems with Time and Probabilities*, Proc. IEEE 11th Real-Time Systems Symposium, 1990.

[4] Kiehn, A., Castellani, I., Hennessy, M., and Boudol, G., *Observing Localities*, Proc. Symposium on Mathematical Foundations of Computer Science, LNCS 520, 1991.

[5] Kopetz, H., *Interval Measurements in Distributed Real-Time Systems*, Proc. IEEE 9th Conference on Distributed Computing Systems, June, 1987.

[6] Krishnan, P., *Distributed CCS*, Proc. CONCUR'91, LNCS 527, August, 1991.

[7] Lamport, L., *Time, Clocks, and the Ordering of Events in a Distributed System* Communication of the ACM, Vol.21, No.7, July, 1978, pp.558-565.

[8] Lamport, L., and Smith,M., *Synchronizing Clocks in the Presence of Faults*, Journal of ACM, Vol.32, No.1, 1985.

[9] Milner, R., *Communication and Concurrency*, Prentice Hall, 1989.

[10] Moller, F., and Tofts, C., *A Temporal Calculus of Communicating Systems*, Proc. CONCUR'90, LNCS 458, 1990.

[11] Morzenti, A., and Pietro, S. P., *An Object-Oriented Logic Language for Modular System Specification*, Proc. ECOOP'91, LNCS 512, June, 1991.

[12] Park, D., *Concurrency and Automata on Infinite Sequences*, Proc. ICALP, LNCS 104, 1981.

[13] Satoh, I., and Tokoro, M., *A Formalism for Real-Time Concurrent Object-Oriented Computing*, Proc. ACM Object-Oriented Programming Systems, Language, and Applications (OOPSLA'92), October, 1992.

[14] Satoh, I., and Tokoro, M., *A Formal Description for Parallel Processes with Time Properties*, To appear in Transactions on Information Processing Society of Japan, Vol.34, No.4, 1993. (in Japanese)

[15] Satoh, I., *A Timed Calculus for Distributed Objects with Clocks*, To appear in Proc. European Conference on Object-Oriented Programming (ECOOP'93), LNCS, July, 1993.

[16] Tokoro, M., and Satoh, I., *Asynchrony and real-time in distributed systems*, US/Japan Seminar on Parallel Symbolic Computing, Cambridge, MA., October, 1992.

[17] Yi, W., *CCS + Time = an Interleaving Model for Real Time Systems*, Proc. Automata, Languages and Programming'91, LNCS 510, 1991.