A Timed Calculus for Distributed Objects with Clocks

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

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

Department of Computer Science, Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama, 223, Japan
Tel: +81-45-560-1150 Fax: +81-45-560-1151

Abstract. This paper proposes a formalism for reasoning about distributed object-oriented computations. The formalism is an extension of Milner's CCS with the notion of local time. It allows to describe and analyze both locally temporal and behavioral properties of distributed objects and interactions among them. We introduce timed bisimulations with respect to local time. These bisimulations equate distributed objects if and only if their behaviors are completely matched and their timings are within a given bound. The bisimulations provide a method to verify distributed objects with temporal uncertainties and real-time objects with non-strict time constraints.

1 Introduction

Distributed systems consist of more than one processor loosely coupled by communication networks with inherent delay[1]. The notion of concurrent object-oriented computation [25] is considered as a powerful means to design and develop distributed systems. This is because objects are logically self-contained active entities interacting with one another through message passing. Recently, many programming languages for distributed systems were developed based on this notion [1, 22].

Communication delay is the most featuring characteristic of distributed systems. It prevents objects from sharing a global clock among all processors. Many distributed systems need real-time facilities to manage time critical responses and interactions with the real world. These facilities have to be materialized by clocks on local processors instead of the shared global clock. However, such clocks can never run at the same rate. Relative differences among clocks that cannot be ignored may lead cooperations among local processors to failure. In order to develop correct programs for distributed systems, we must take this problem into consideration. Hence, we need a

*Also with Sony Computer Science Laboratory Inc. 3-14-13 Higashi-Gotanda, Shinagawa-ku, Tokyo, 141, Japan.

[1]Geographical distance between processors manifests in communication delay.

theoretical model for describing and analyzing both locally temporal properties and functionally behavioral properties of distributed object-oriented computing.

A number of computing models for describing and verifying distributed systems have been proposed based on temporal logic, automata, and process calculi. However, most of them are intended to model only behavioral aspects in distributed computing and thus lack in the ability of representing temporal properties. Others can manage only temporal properties based on the global time which cannot be essentially realized in distributed systems. Consequently, they can not model the notion of local time in distributed systems. In the fields of artificial intelligence and real-time systems, some logical systems to deal with different time scales have been proposed in [4, 7, 16]. However they cannot sufficiently model time based on local clocks which may drift in distributed systems. Also, some researchers have explored methods for agreement on processes' time by using clock synchronization [10, 13] and for maintaining local time based on causality by using time-stamps [9, 12]. However the former methods cannot cope with systems where communication delay is unpredictable. The latter methods lose real-time duration between events.

The goal of this paper is to develop a formalism for reasoning about distributed object-oriented computations, in particular, local time properties in computation. The formalism is based on an existing process calculus, CCS [14]. This is because CCS allows us to easily model many features of concurrent objects by means of its powerful expressive capabilities [8, 18]. For example, objects can be viewed as processes; interactions among objects can be seen as communications; and encapsulation can be modeled by the restriction of visible communications. However, CCS essentially lacks in the notion of local time and thus we need to extend CCS with it. Indeed the authors introduced a timed extended process calculus, called RtCCS, for real-time concurrent (but non-distributed) object-oriented computing in an earlier paper [21]. RtCCS is an extension of CCS [14] with timeout operator and timed observation equivalences. The extensions make RtCCS unique among other timed process calculi [5, 6, 15, 17]. However, the extensions are based on the global time and thus RtCCS, like other timed calculi, cannot represent the notion of local time in distributed systems. In this paper, we develop a process calculus called DtCCS (*Distributed timed Calculus of Communication Systems*), by extending CCS with the notion of local time in order to explicitly describe and analyze both temporal properties based on local time and behavioral properties in distributed object-oriented computing[2]. Based on DtCCS, we furthermore develop theoretical proof techniques for distributed objects.

The organization of this paper is as follows: in the next section, we first informally introduce our approach to extend CCS with the notion of local time. Section 3 defines the syntax and the semantics of DtCCS and then presents how to describe distributed objects along with some examples in DtCCS. In Section 4 we present timed bisimulations based on local time and study their basic theoretical properties, and then we present some examples to demonstrate the usefulness of our formalism. The final section contains some concluding remarks and future works.

[2]There have indeed been some process calculi for distributed computing [2, 3, 11] but they are extensions of CCS with the concept of process locations and not local time.

2 Basic Framework

In this section, we present a brief introduction to our formalism.

Time in Distributed Computing

Before giving an exposition of our formalism, we first present the basic idea for modeling local time. If relative motion of all processors is negligible, from Einstein's Relativity, the passage of physical *time* in every processor elapses at the same rate. On the other hand, each local *clock* progresses at its own rate. Clock rates are different from each other and these rate may vary within a certain bound. Hence, we can envisage local clocks as follows: each actual local clock reads its own current time by translating the passage of the global time into its own time coordinate according to its own measurement rate. Therefore, local times which are measured by different clocks may be different from one another, although the clocks share the same global time. Summarizing this discussion, we give the following two basic assumptions on time in distributed computing: (1) *all processors in a distributed system share the conceptual global time*, and (2) *the local clock of each processor measures the global time according to its own time unit and precision*. We will hereafter call a clock running at a given constant rate as a *constant clock*, and call a clock running at a dynamically drifting rate within a given bound as a *variable clock*.

Extensions of CCS

According to the above assumptions, we develop a formalism for reasoning about behavioral and temporal properties of distributed objects by using a language based on CCS. In order to represent the temporal properties of distributed systems, we need to extend CCS with the ability of representing local time properties in distributed systems. To do this, we introduce three temporal primitives: *timed behavior*, *global time*, and *local clock*. We briefly summarize these extensions[3] as follows:

- *Timed Behavior*. The behavior of distributed systems is dependent on the passage of time, such as delaying process and timeout handling. Particularly, in distributed systems timeout handling is a crucial part for failure detection in communication network and other processors. Therefore, we introduce a special binary operator having the semantics of timeout handling, written as $\langle\ ,\ \rangle_t$, and called a *timeout* operator. For instance, $\langle P, Q\rangle_t$ behaves as P if P can execute an initial transition within t units of time, and behaves as Q if P does not perform any action within t units of time.
- *Global Time*. We assume that all processors share the conceptual global time. The passage of the global time is represented as a special action. The action is a synchronous broadcast message over all objects and corresponds to the passage of one unit of the global time. It is described as a $\surd$, called a *tick* action. Please note that, in our formalism, the existence of the global time does not implies the existence of an actual global clock: it only provides the time that each local clock may measure.

[3]The extensions, except for local clock, are essentially equivalent to ones in RtCCS in [21].

- *Local Clock.* Local clocks are given as special mappings from time instants on local time into corresponding time instants on the global time, according to its own time unit and precision. The mapping defines how many time instants on the global time corresponds to the length of one time unit in the local time. Conversely, local time on each processor is given as a collection of all the time instants which can be translated into the global time by the mapping. In order to represent a variable clock, the clock may be given as a non-deterministic mapping.

In our calculus, descriptions of objects with respect to local times are translated into ones in the global time. The descriptions are interpreted based only on the global time. Thus, we obtain a uniform way to easily analyze distributed objects with their own local times. As a result, we can use many pleasant properties of timed process calculi based on global time, including the proof techniques of RtCCS presented in [21]. The translation maps time values with respect to local time in the descriptions into values with respect to the global time. Our method of interpreting the descriptions on local time may seem to be similar to that of [4, 16]. However, the latter is intended to deal with different time scales in specification and cannot sufficiently model time based on variable clocks in distributed systems.

3 Definition

This section first presents the basic definitions for time related notations and then defines the syntax and the semantics of the calculus.

3.1 Time Domain

In our formalism, time is represented as a time domain composed of time instants. An instant represents a time interval from an initial time and we restrict the instants to be discrete, instead of continuous. Each time domain corresponds to a local time measured by one local clock. We assume a special time domain which is a finest and absolute reference time basis for all local times and is called the *global time domain.*

Definition 3.1 Let $\mathcal{T}$ denote the set of the positive integers including 0. We call $\mathcal{T}$ a *time domain.*

$$\mathcal{T} \equiv \mathcal{N} \cup \{0\} \quad \text{where } \mathcal{N} \text{ is a set of natural numbers.}$$

Especially, the global time domain is denoted as $\mathcal{T}_G$. □

All occurrences of any instant times on local time domains correspond to instant times in the global time domain. The linkage between a local time domain and the global time domain is given as the following mapping.

Definition 3.2 Let $\mathcal{T}_\ell$ be a local time domain and $\delta_{min},\ \delta_{max} \in \mathcal{T}_G : 0 < \delta_{min} \leq \delta_{max}$. A clock mapping $\theta : \mathcal{T}_\ell \rightarrow \mathcal{T}_G$ is defined as follows: for all $t \in \mathcal{T}_\ell$,

$$\theta(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta(t-1) + \delta & \text{if } t > 0 \ \text{ where } \ \delta \in \{\ d \in \mathcal{T}_G \mid \delta_{min} \leq d \leq \delta_{max}\} \end{cases}$$

where we call θ a *clock mapping*, or simply a *clock*. We let $\{\theta(t)\}$ denote $\{\ t_G \mid \forall t_G \in \mathcal{T}_G : t_G = \theta(t)\}$. Particularly, if $\delta_{min} = \delta_{max}$, we call θ a *constant clock* and hereafter will sometimes abbreviate the definition of θ as $\theta(t) \stackrel{\text{def}}{=} \delta t$. □

In above definition, δ corresponds to the interval of one time unit on the local time domain according to the global time domain. δ_{min} is the lower bound of that interval and δ_{max} the upper bound of that interval. Also $\{\theta(t)\}$ means the set of the whole time values can be evaluated in $\theta(t)$.

We present some examples of clock mappings.

Example 3.3 *Examples of clock mappings*

(1) The clock mapping of a constant clock whose time units is three units of the global time, is denoted as follows: $\theta(t) \stackrel{\text{def}}{=} 3t$.

(2) The clock mapping of a variable clock whose time unit varies from two to four units of the global time is given as follows:

$$\theta(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta(t-1) + \delta & \text{if } t > 0 \quad \text{where } \ \delta \in \{2,\ 3,\ 4\} \end{cases}$$

□

As previously mentioned, time is discrete in the calculus. Therefore, if an event is executed between two consecutive time instants, we say that the event occurs at the earlier time instant. Also, if two or more events occur between two consecutive time instants, we say that the events occur at the same time.

Here, we present the inverse of clock mapping. It maps a time instant on global time into the coordinates of local time.

Definition 3.4 The inverse mapping of θ: $\mathcal{T}_\ell \rightarrow \mathcal{T}_G$ is defined as follows:

$$\{\theta^{-1}(t_G)\} \stackrel{\text{def}}{=} \{\ t \mid \forall t \in \mathcal{T}_\ell : \mathbf{min}\ \{\theta(t)\} \leq t_G < \mathbf{max}\ \{\theta(t+1)\}\ \}$$

where $t_G \in \mathcal{T}_G$. □

To demonstrate how $\theta^{-1}(t)$ works, we show an inverse mappings.

Example 3.5 *Examples of clock inverse mappings*
Recall θ in (2) of Example 3.3. We present a few of the possible results of its inverse mapping $\theta^{-1}(t)$ as follows: $\{\theta^{-1}(1)\} = \{0\}, \{\theta^{-1}(2)\} = \{0, 1\}, \{\theta^{-1}(3)\} = \{0, 1\}, \{\theta^{-1}(4)\} = \{1, 2\}, \{\theta^{-1}(6)\} = \{1, 2, 3\}, \{\theta^{-1}(8)\} = \{2, 3, 4\}, \ldots$ □

We show an alternative definition of the inverse mapping below.

Proposition 3.6 Let θ: $\mathcal{T}_\ell \to \mathcal{T}_G$ and $\phi : \mathcal{T}_G \to \mathcal{T}_\ell$ be the following functions. We have that for all $t_G \in \mathcal{T}_G$: $\{\phi(t_G)\}$ is equivalent to $\{\theta^{-1}(t_G)\}$.

$$\begin{aligned} \theta(t) &\stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta(t-1) + \delta & \text{if } t > 0 \text{ where } \delta \in \{\ d \in \mathcal{T}_G \mid \delta_{min} \le d \le \delta_{max}\} \end{cases} \\ \{\phi(t_G)\} &\stackrel{\text{def}}{=} \{\ t \mid \forall t \in \mathcal{T}_\ell : \lfloor t_G/\delta_{max} \rfloor \le t \le \lfloor t_G/\delta_{min} \rfloor\ \} \end{aligned}$$

where $\delta_{min}, \delta_{max} \in \mathcal{T}_G : 0 < \delta_{min} \le \delta_{max}$. □

Note that the definitions of the clock mapping and its inverse mapping are different from that of usual functions in mathematics.

3.2 Notation and Syntax

Here we present the syntax of DtCCS. The syntax is an extension of Milner's CCS [14] by introducing a tick action, a timeout operator, and clock translation rules.

A few preliminary definitions are needed before giving the definition of the syntax. We first define notation conventions which we will follow hereafter.

Definition 3.7

- Let $\mathcal{A}$ be an infinite set of communication action names, ranged over by a,b,....
- Let $\overline{\mathcal{A}}$ be the set of co-names, ranged over by $\overline{a}$,$\overline{b}$,... where an action $\overline{a}$ is the complementary action of a, and $\overline{\overline{a}} \equiv a$.
- Let $\mathcal{L} \equiv \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of action labels, ranged over by $\ell, \ell', \ldots$.
- Let τ denote an internal action which is unobservable from the outside.
- Let $\surd$ denote a tick action which represents the passage of one time unit.
- Let $Act \equiv \mathcal{L} \cup \{\tau\}$ be the set of behavior actions, ranged over by α,β,....
- Let $Act_{\mathcal{T}} \equiv Act \cup \{\surd\}$ be the set of actions, ranged over by μ, ν,....

□

In our formalism, distributed objects are described by means of expressions as defined below. In order to clarify our exposition, we divide the expressions into two groups: sequential expressions for describing objects on a processor (or a node) with one local clock, and interacting expressions for describing interactions among distributed objects following different clocks.

Definition 3.8 The set $\mathcal{S}$ of sequential expressions, ranged over by $S, S_1, S_2, \ldots$ is the smallest set which contains the following expressions:

$$\begin{array}{rcll} S & ::= & \mathbf{0} & (\textit{Terminated Process}) \\ & \mid & X & (\textit{Process Variable}) \\ & \mid & \alpha.S & (\textit{Sequential Execution}) \\ & \mid & S_1 + S_2 & (\textit{Alternative Choice}) \\ & \mid & \mathbf{rec}\, X : S & (\textit{Recursive Definition}) \\ & \mid & \langle S_1, S_2 \rangle_t & (\textit{Timeout}) \end{array}$$

where t is an element of a time domain. We assume that X is always *guarded* [4]. We shall often use the more readable notation $X \stackrel{\text{def}}{=} S$ instead of $\mathbf{rec}\, X : S$. □

Intuitively, the meaning of constructors on $\mathcal{S}$ are as follows: $\mathbf{0}$ represents a terminated and deadlocked object; $\alpha.S$ performs an action α and then behaves like S; $S_1 + S_2$ represents an object which may behave as either S_1 or S_2; $\mathbf{rec}\, X : S$ binds the free occurrences of X in S; $\langle S_1, S_2 \rangle_t$ represents an object such that it behaves as S_1 if S_1 can execute an initial transition within t time units, whereas it behaves as S_2 if S_1 does not perform any action within t time units.

We now define the syntax of expressions for interactions among different processors with local clocks as shown below. We assume that $[\![\cdot]\!]_\theta$ is a clock translation mapping which will be defined later. It allows sequential expressions following a local clock θ to be translated into expressions on the global time domain $\mathcal{T}_G$.

Definition 3.9 The set $\mathcal{P}$ of interacting expressions on local time, ranged over by P, P_1, P_2, is defined by the following grammar:

$$\begin{array}{lcll} P & ::= & [\![S]\!]_\theta & (\textit{Local Object}) \\ & | & P_1 | P_2 & (\textit{Parallel Composition}) \\ & | & P[f] & (\textit{Relabeling}) \\ & | & P \setminus L & (\textit{Encapsulation}) \end{array}$$

where θ is a clock mapping. We assume that $f \in Act \to Act$, $f(\tau) = \tau$, and $L \subseteq Act$. □

The informal meaning of constructors of $\mathcal{P}$ is as follows: $[\![S]\!]_\theta$ represents a sequential expression S executed on a processor (or a node) with a local clock θ; $P_1|P_2$ allows P_1 and P_2 to execute in parallel; $P[f]$ behaves like P but with the actions in P relabeled by function f; $P \setminus L$ behaves like P but with actions in $L \cup \bar{L}$ prohibited.

3.3 Semantics

As we noted already, the definition of the semantics of DtCCS consists of two parts: clock translation rules, written as $[\![S]\!]_\theta$, which translate expressions on a local time into expressions on the global time, and structural transition rules which define the mean of all constructors in expressions on the global time.

Expressions on Local Time

We here define the clock translation rules $[\![S]\!]_\theta$. We first present the key idea of the rules. In order to translate expressions on local time into expressions on the global time, we translate all local time values in expressions on local time into time values on the global time. In our formalism only the deadline time of the timeout operator corresponds to such time values. We introduce special translation rules which map each deadline time on local clocks in expressions into deadline time based on the global time, by using the clock mapping.

[4] X is *guarded* in S if each occurrence of X is only within some subexpressions $\alpha.S'$ in S where α is not an empty element; c.f. *unguarded* expressions, e.g. $\mathbf{rec}\, X : X$ or $\mathbf{rec}\, X : X + S$.

Definition 3.10 Let $\mathcal{T}_\ell$ be a local time domain and θ be a clock mapping from $\mathcal{T}_\ell$ to $\mathcal{T}_G$. The clock translation rule $[\![\cdot]\!]_\theta$ is recursively defined by the following syntactic rewriting rules.

$$\begin{aligned}
[\![\mathbf{0}]\!]_\theta &\longrightarrow \mathbf{0} \\
[\![X]\!]_\theta &\longrightarrow X \\
[\![\alpha.S]\!]_\theta &\longrightarrow \alpha.[\![S]\!]_{\theta'} \\
[\![S_1 + S_2]\!]_\theta &\longrightarrow [\![S_1]\!]_{\theta'} + [\![S_2]\!]_{\theta'} \\
[\![\mathbf{rec}\, X : S]\!]_\theta &\longrightarrow \mathbf{rec}\, X : [\![S]\!]_{\theta'} \\
[\![\langle S_1, S_2\rangle t]\!]_\theta &\longrightarrow \langle [\![S_1]\!]_{\theta'}, [\![S_2]\!]_{\theta''}\rangle_{\theta'(t)}
\end{aligned}$$

where $\theta', \theta'' \stackrel{\text{def}}{=} \theta$ such that $\forall t_\ell \in \mathcal{T}_\ell : \theta'(t_\ell) = \theta(t_\ell)$, $\theta''(t_\ell) + \theta(t) = \theta(t_\ell + t)$. □

We briefly explain the intuitive meaning of the main rules. The third rule translates an unpredictable synchronization time for waiting for α into an unpredictable time on the global time domain. The fourth rule shows that all alternative subsequences in a processor share the same clock. The last rule means that the deadline time t on local clock θ is mapped into deadline time on the global time.

We show some notable points on the above clock translation rules.

- Hereafter we will often omit the $\longrightarrow$ translation if it is understood from the context.
- By the definition of $\mathcal{P}$, expressions applicable to $[\![\cdot]\!]_\theta$ are restricted to expressions in $\mathcal{S}$. This means that expressions to model internal interactions, such as concurrency and encapsulation, among objects sharing the same clock cannot be translated into the global time by $[\![\cdot]\!]_\theta$. However, this restriction never results in unavoidable difficulties in describing distributed systems. This is because any expression for interaction on the same clock can be reduced to an equivalent sequential expression in $\mathcal{S}$ by using the expansion rules shown in Corollary 1 and Proposition 10 of [21]. Therefore, we first translate expressions for interacting objects following the same clock into expressions in $\mathcal{S}$ using the expansion rules and then we can apply $[\![\cdot]\!]_\theta$ to the translated expressions.

Expressions on Global Time

The clock translation rules can completely eliminate all $[\![\cdot]\!]_\theta$ from expressions in $\mathcal{P}$. Therefore, in order to define the operational semantics of DtCCS, we need to give semantics to all syntactical constructors in $\mathcal{P}$ except $[\![\cdot]\!]_\theta$. The semantics is given in terms of a labeled transition system [20]. The operational semantics of the language which consists of the translated expressions on the global time, is given as a labeled transition system $\langle\, \mathcal{P},\ Act_\mathcal{T},\ \{\, \stackrel{\mu}{\longrightarrow} \mid \mu \in Act_\mathcal{T} \,\} \,\rangle$ where $\stackrel{\mu}{\longrightarrow}$ is a transition relation ($\stackrel{\mu}{\longrightarrow} \subseteq \mathcal{P} \times \mathcal{P}$ where $\mathcal{P}$ contains no constructor $[\![S]\!]_\theta$). The definition of the semantics is structurally given in two phases. The first phase defines the relations $\stackrel{\alpha}{\longrightarrow}$ for each $\alpha \in Act$. The inference rules determining $\stackrel{\alpha}{\longrightarrow}$ are given in Figure 1. This is based on the standard operational semantics for CCS except for the addition of the timeout

$$
\begin{array}{lll}
\alpha.P \xrightarrow{\alpha} P & & \\
P_1 \xrightarrow{\alpha} P_1' & \text{implies} & P_1 + P_2 \xrightarrow{\alpha} P_1',\ P_2 + P_1 \xrightarrow{\alpha} P_1' \\
P_1 \xrightarrow{\alpha} P_1' & \text{implies} & P_1|P_2 \xrightarrow{\alpha} P_1'|P_2,\ P_2|P_1 \xrightarrow{\alpha} P_2|P_1' \\
P_1 \xrightarrow{a} P_1',\ P_2 \xrightarrow{\bar{a}} P_2' & \text{implies} & P_1|P_2 \xrightarrow{\tau} P_1'|P_2' \\
P \xrightarrow{\alpha} P' & \text{implies} & P[f] \xrightarrow{f(\alpha)} P'[f] \\
P \xrightarrow{\alpha} P',\ \alpha \notin L \cup \overline{L} & \text{implies} & P \setminus L \xrightarrow{\alpha} P' \setminus L \\
P\{\mathbf{rec}\, X : P/X\} \xrightarrow{\alpha} P' & \text{implies} & \mathbf{rec}\, X : P \xrightarrow{\alpha} P' \\
P_1 \xrightarrow{\alpha} P_1',\ t > 0 & \text{implies} & \langle P_1, P_2 \rangle_t \xrightarrow{\alpha} P_1' \\
P_2 \xrightarrow{\alpha} P_2' & \text{implies} & \langle P_1, P_2 \rangle_0 \xrightarrow{\alpha} P_2'
\end{array}
$$

Figure 1: Operational Rules of $\mathcal{P}$ on Global Time

$$
\begin{array}{lll}
\mathbf{0} \xrightarrow{\surd} \mathbf{0} & & \\
\ell.P \xrightarrow{\surd} \ell.P & & \\
P_1 \xrightarrow{\surd} P_1',\ P_2 \xrightarrow{\surd} P_2' & \text{implies} & P_1 + P_2 \xrightarrow{\surd} P_1' + P_2' \\
P_1 \xrightarrow{\surd} P_1',\ P_2 \xrightarrow{\surd} P_2',\ P_1|P_2 \not\xrightarrow{\tau} & \text{implies} & P_1|P_2 \xrightarrow{\surd} P_1'|P_2' \\
P \xrightarrow{\surd} P' & \text{implies} & P[f] \xrightarrow{\surd} P'[f] \\
P \xrightarrow{\surd} P' & \text{implies} & P \setminus L \xrightarrow{\surd} P' \setminus L \\
P\{\mathbf{rec}\, X : P/X\} \xrightarrow{\surd} P' & \text{implies} & \mathbf{rec}\, X : P \xrightarrow{\surd} P' \\
P_1 \xrightarrow{\surd} P_1',\ t > 0 & \text{implies} & \langle P_1, P_2 \rangle_t \xrightarrow{\surd} \langle P_1', P_2 \rangle_{t-1} \\
P_2 \xrightarrow{\surd} P_2' & \text{implies} & \langle P_1, P_2 \rangle_0 \xrightarrow{\surd} P_2'
\end{array}
$$

Figure 2: Temporal Rules of $\mathcal{P}$ on Global Time

operator. The new action $\surd$ does not effect any rule. The second phase defines the relation $\xrightarrow{\surd}$ by inference rules given in Figure 2.

Let us give some remarks on the definition of the semantics.

- The syntax and the semantics of the translated expressions on the global time domain essentially coincide with these of RtCCS [21]. Therefore, the translated expressions enjoy proof techniques presented in [21] for RtCCS, such as timed strong equivalence and timed observation equivalence.
- As previously mentioned, there have been many time extended process calculi. The syntax and the semantics of the expressions translated into the global time domain (i.e. RtCCS) are somewhat similar to some of them. Detail comparisons can be found in [21].
- In DtCCS, external actions cannot be performed before their partner actions in other objects are ready to communicate it. Objects must perform $\surd$ actions,

while waiting for partner actions corresponding to their external actions. If an object can perform any executable communication (including τ), it must perform the communication immediately, instead of idling unnecessarily.

3.4 Examples on Description of Distributed Objects

In order to illustrate how to describe distributed object in DtCCS, we present some simple examples.

Example 3.11 *Interaction between distributed objects*
We suppose interaction between a client and a sever object on different processors by means of *remote procedure call*.

- The client object (*Client*) sends a request message ($\overline{req}$) and then waits for a return message (*ret*). If the return message is not received within 6 units of local time, then it sends the request message again.
- Upon reception of a request message (*req*), the server object (*Server*) sends a return message ($\overline{ret}$) after an internal execution of 5 units of local time.

These objects are denoted as follows:

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

We assume that the client and server objects are allocated to different processors. The time unit of variable clock θ_c for the client varies from 4 to 6 units of the global time. The time unit of variable clock θ_s for the server varies from 3 to 5 units of the global time. θ_c and θ_s are defined as follows:

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

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

By $[\![\cdot]\!]_\theta$ mapping rules, the client and the server are mapped on the global time domain as shown below. From the definition of θ_c and θ_s, there are multiple results for $\theta_c(6) \in \{24, 25, ..36\}$ and $\theta_s(5) \in \{15, 16, ..25\}$:

$$\begin{aligned} [\![Client]\!]_{\theta_c} &\longrightarrow \cdots \longrightarrow \overline{req}.\langle ret.\mathbf{0}, [\![Client]\!]_{\theta_c}\rangle_{\theta_c(6)} \\ [\![Server]\!]_{\theta_s} &\longrightarrow \cdots \longrightarrow req.\langle \mathbf{0}, \overline{ret}.[\![Server]\!]_{\theta_s}\rangle_{\theta_s(5)} \end{aligned}$$

The interaction between the objects is described as the following parallel composition:

$$([\![Client]\!]_{\theta_c} | [\![Server]\!]_{\theta_s}) \setminus \{req, ret\}$$

where $\setminus\{req, ret\}$ makes internal communications encapsulated from the environment. The result of the interaction is dependent on the evaluated values of $\theta_c(6)$ and $\theta_s(5)$. Here we show some of the possible results:

(1) In the case of $\theta_c(6) = 30$ and $\theta_s(5) = 20$:

$$\begin{aligned}
&(\llbracket Client \rrbracket_{\theta_c} | \llbracket Server \rrbracket_{\theta_s}) \backslash \{req, ret\} \\
&\qquad \xrightarrow{\tau} \quad (\langle ret.\mathbf{0}, \llbracket Client \rrbracket_{\theta_c} \rangle_{30} | \langle \mathbf{0}, \overline{ret}.\llbracket Server \rrbracket_{\theta_s} \rangle_{20}) \backslash \{req, ret\} \\
&\qquad (\xrightarrow{\surd})^{20} \quad (\langle ret.\mathbf{0}, \llbracket Client \rrbracket_{\theta_c} \rangle_{10} | \overline{ret}.\llbracket Server \rrbracket_{\theta_s}) \setminus \{req, ret\} \\
&\qquad \xrightarrow{\tau} \quad (\mathbf{0} | \llbracket Server \rrbracket_{\theta_s}) \setminus \{req, ret\} \\
&\qquad \quad (successful)
\end{aligned}$$

In the above case (i.e. $\theta_c(6) > \theta_s(5)$), the client can always receive the return message before it goes into timeout.

(2) In the case of $\theta_c(6) = 24$ and $\theta_s(5) = 25$:

$$\begin{aligned}
&(\llbracket Client \rrbracket_{\theta_c} | \llbracket Server \rrbracket_{\theta_s}) \backslash \{req, ret\} \\
&\qquad \xrightarrow{\tau} \quad (\langle ret.\mathbf{0}, \llbracket Client \rrbracket_{\theta_c} \rangle_{24} | \langle \mathbf{0}, \overline{ret}.\llbracket Server \rrbracket_{\theta_s} \rangle_{25}) \backslash \{req, ret\} \\
&\qquad (\xrightarrow{\surd})^{24} \quad (\llbracket Client \rrbracket_{\theta_c} | \langle \mathbf{0}, \overline{ret}.\llbracket Server \rrbracket_{\theta_s} \rangle_1) \setminus \{req, ret\} \\
&\qquad (\xrightarrow{\surd})^{1} \quad (\llbracket Client \rrbracket_{\theta_c} | \overline{ret}.\llbracket Server \rrbracket_{\theta_s}) \setminus \{req, ret\} \\
&\qquad \quad (failure)
\end{aligned}$$

In the above case, the client goes into timeout before receiving a return message *ret* because of $\theta_c(6) \leq \theta_s(5)$. Thus, the objects goes into a deadlock.

DtCCS allows us to analyze explicitly how the differences among local clocks affect the result of interactions in distributed computing. □

Note that the deadline time of the timeout operator in *Server* means execution steps for handling the request in the server. θ_s in $\llbracket Server \rrbracket_{\theta_s}$ represents an index on the performance of a processor executing the server. For example, the larger δ_s in θ_s is, the faster the processor is. The result shows that the clock translation rules $\llbracket \cdot \rrbracket_\theta$ allows us to represent differences among processors' performances as well as differences among processors' clocks.

Example 3.12 *Embedding Asynchronous Communication with Delay*
We illustrate an embedding of an asynchronous communication mechanism in DtCCS. The key idea of the embedding is to express asynchronous communication in terms of synchronous communication and a messenger creation, i.e. asynchronous message sending is represented by creating a process which can engage only in an input action with the same name of the message. Note that this embedding is very similar to the ways to express asynchronous communication developed in [8, 11].

Let $\uparrow a.P$ and $\downarrow a.P$ denote asynchronous sending and receiving expressions, respectively. $\uparrow a.P$ sends a message to the target name a and continues to execute P without waiting the reception of the message. $\downarrow a.P$ behaves like P after receiving a message with a target name a. These expressions are encoded by expressions of DtCCS as follows:

(1) First we describe the case we can neglect the transmission delay of the communication.

$$\begin{array}{lcll} \uparrow a.P & \equiv & (\overline{c}.P | c.\overline{a}.\mathbf{0}) \setminus \{c\} & (\textit{Asynchronous Sending}) \\ \downarrow a.P & \equiv & a.P & (\textit{Receiving}) \end{array}$$

where we assume that newly introduced name c does not appear in P.

Note that $\uparrow a\,.P$ can continue to execute P without blocking before $\uparrow a$ is received by another object.

(2) Next we take communication delay into consideration. In the above expression $c.\overline{a}.\mathbf{0}$ corresponds to a communication channel for the message and thus we extend $c.\overline{a}.\mathbf{0}$.

$$\begin{array}{lcll} \uparrow a.P & \equiv & (\overline{c}.P | [\![c.\langle \mathbf{0}, \overline{a}.\mathbf{0} \rangle_1]\!]_{\theta_d}) \setminus \{c\} & (\textit{Asynchronous Sending}) \\ \theta_d(t) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{ll} 0 & \text{if } t = 0 \\ \theta_d(t-1) + \delta_d & \text{if } t > 0 \end{array} \right. & \text{where } \delta_d \in \{\ d \mid d_{min} \leq d \leq d_{max}\} \end{array}$$

where d_{min} and d_{max} correspond to the minimum and maximal delay of the transmission, respectively. The multiple results of $\theta_d(1)$ allows us to model uncertain delay time within a given bound ($d_{min} \leq d \leq d_{max}$).

(3) We extend the server object of Example 3.11 with asynchronous communication.

$$\textit{Server} \stackrel{\text{def}}{=} \downarrow \textit{req}\,.\langle \mathbf{0}, \uparrow \textit{ret}\,.\textit{Server} \rangle_5$$

We derive an expansion of *Server* as follows:

$$\begin{array}{rcl} [\![\textit{Server}]\!]_{\theta_s} & \equiv & \textit{req}.\langle \mathbf{0}, (\overline{c}.[\![\textit{Server}]\!]_{\theta_s} | c.\langle \mathbf{0}, \overline{\textit{ret}}.\mathbf{0} \rangle_{\theta_d(1)}) \setminus \{c\} \rangle_{\theta_s(5)} \\ & \xrightarrow{\textit{req}} & \langle \mathbf{0}, (\overline{c}.[\![\textit{Server}]\!]_{\theta_s} | c.\langle \mathbf{0}, \overline{\textit{ret}}.\mathbf{0} \rangle_{\theta_d(1)}) \setminus \{c\} \rangle_{\theta_s(5)} \\ & (\xrightarrow{\checkmark})^{\theta_s(5)} & (\overline{c}.[\![\textit{Server}]\!]_{\theta_s} | c.\langle \mathbf{0}, \overline{\textit{ret}}.\mathbf{0} \rangle_{\theta_d(1)}) \setminus \{c\} \\ & \xrightarrow{\tau} & ([\![\textit{Server}]\!]_{\theta_s} | \langle \mathbf{0}, \overline{\textit{ret}}.\mathbf{0} \rangle_{\theta_d(1)}) \setminus \{c\} \\ & \xrightarrow{\textit{req}} & \ldots \end{array}$$

where θ_s were already defined in Example 3.11.

The server can receive the next *req* message while $\overline{\textit{ret}}$ is being transmitted, and thus not yet received. □

4 Bisimulation Based on Local Time

This section defines two timed bisimulations. In the earlier paper [21], the author provided temporally strict equivalences in which timed equivalent objects must *completely* match their time properties as well as their functional behaviors. However, the temporal properties of any two distributed objects may not completely match one another because local processors may not compute at exactly the same speed and local clocks may not run at the same rate. It is natural and practical that two objects on

different processors can be treated as equivalent only if their behaviors are completely matched and differences in their timings are within a given bound. Therefore, we develop such equivalences by extending the notion of bisimulation [19, 14] and study some theoretical properties of them. Hereafter we will only deal with expressions with no occurrences of free[5] process variables.

Definition 4.1 *Optimistically Timed Bisimulation*
A binary relation $\mathcal{R}_\theta$ is a *optimistically timed bisimulation* on clock θ ($\theta : \mathcal{T}_\ell \to \mathcal{T}_G$) if $(P, Q) \in \mathcal{R}_\theta$ implies, for all $\alpha \in Act$,

(i) $\forall m \in \mathcal{T}_G,\ \forall P'\colon P(\xrightarrow{\checkmark})^m \xrightarrow{\alpha} P' \supset \exists n \in \mathcal{T}_G,\ \exists Q'\colon Q(\xrightarrow{\checkmark})^n \xrightarrow{\alpha} Q'$
$\wedge\ \{\theta^{-1}(m)\} \cap \{\theta^{-1}(n)\} \neq \emptyset \wedge (P', Q') \in \mathcal{R}_\theta.$

(ii) $\forall n \in \mathcal{T}_G,\ \forall Q'\colon Q(\xrightarrow{\checkmark})^n \xrightarrow{\alpha} Q' \supset \exists m \in \mathcal{T}_G,\ \exists P'\colon P(\xrightarrow{\checkmark})^m \xrightarrow{\alpha} P'$
$\wedge\ \{\theta^{-1}(m)\} \cap \{\theta^{-1}(n)\} \neq \emptyset \wedge (P', Q') \in \mathcal{R}_\theta.$

We let "$\sim_\theta$" denote the largest optimistically timed bisimulation, and call P and Q *optimistically timed bisimilar* if $P \sim_\theta Q$. Also if $\theta(t) \stackrel{\text{def}}{=} t$, we let $\sim$ denote $\sim_\theta$. □

Let us describe the informal meaning of $\sim_\theta$. If $P \sim_\theta Q$, an observer following clock θ cannot distinguish between the behavioral contents and the timings of P and Q. Especially, when the running rate of the clock drifts, there may be measurement errors due to the drift. The observer leaves these errors out of consideration. Therefore, $\sim_\theta$ optimistically equates the temporal properties of two objects.

Proposition 4.2 $\forall S_1, S_2 \in \mathcal{S}, \quad S_1 \sim S_2 \quad \textit{iff} \quad [\![S_1]\!]_\theta \sim_\theta [\![S_2]\!]_\theta$ □

This proposition shows a relationship between the optimistically timed bisimilarity and the clock translation rules.

Next we define the pessimistic counterpart of $\sim_\theta$.

Definition 4.3 *Pessimistically Timed Bisimulation*
A binary relation $\mathcal{R}_\theta$ is a *pessimistically timed bisimulation* on clock θ ($\theta : \mathcal{T}_\ell \to \mathcal{T}_G$ and $\forall t \in \mathcal{T}_\ell : \mathbf{max}\{\theta(t)\} \leq \mathbf{min}\{\theta(t+1)\}$) if $(P, Q) \in \mathcal{R}_\theta$ implies, for all $\alpha \in Act$,

(i) $\forall m \in \mathcal{T}_G, \forall i \in \{\theta^{-1}(m)\},\ \forall P'\colon P(\xrightarrow{\checkmark})^m \xrightarrow{\alpha} P' \supset \exists n \in \mathcal{T}_G, \exists Q'\colon Q$
$(\xrightarrow{\checkmark})^n \xrightarrow{\alpha} Q' \wedge \mathbf{max}\{\theta(i)\} \leq n < \mathbf{min}\{\theta(i+1)\} \wedge (P', Q') \in \mathcal{R}_\theta.$

(ii) $\forall n \in \mathcal{T}_G, \forall j \in \{\theta^{-1}(n)\},\ \forall Q'\colon Q(\xrightarrow{\checkmark})^n \xrightarrow{\alpha} Q' \supset \exists m \in \mathcal{T}_G, \exists P'\colon P$
$(\xrightarrow{\checkmark})^m \xrightarrow{\alpha} P' \wedge \mathbf{max}\{\theta(j)\} \leq m < \mathbf{min}\{\theta(j+1)\} \wedge (P', Q') \in \mathcal{R}_\theta.$

We let "$\simeq_\theta$" denote the largest pessimistically timed bisimulation, and call P and Q *pessimistically timed bisimilar* if $P \simeq_\theta Q$. Also if $\theta(t) \stackrel{\text{def}}{=} t$, we let $\simeq$ denote $\simeq_\theta$. □

[5] An occurrence of a variable X in an expression $P \in \mathcal{P}$ is called *bounded* if it occurs in a subexpression of the form $\mathbf{rec}\, X : P'$. Otherwise it is called *free*.

We show the informal meaning of $\simeq_\theta$. If P and Q are pessimistically timed bisimilar on θ, an observer according to clock θ cannot distinguish between their behavioral contents and between their timings. When θ is a variable clock, there may be measurement errors due to the drift of θ. The observer in $\simeq_\theta$ takes these errors into consideration and equates only objects whose temporal and behavioral properties cannot constantly be distinguished by the observer, regardless of whether the errors affect the measurement result of θ or not. Hence, $\simeq_\theta$ pessimistically equates the temporal properties of two objects.

Since $\simeq_\theta$ is a particular instance of $\sim_\theta$, we have the following relationship between $\sim_\theta$ and $\simeq_\theta$

Proposition 4.4 $\forall P_1, P_2 \in \mathcal{P}$, *If* $P_1 \simeq_\theta P_2$, *then* $P_1 \sim_\theta P_2$. □

The above proposition shows that $\simeq_\theta$ is more strict than $\sim_\theta$. Another interesting fact is that $\sim_\theta$ coincides with $\simeq_\theta$ if clock θ is a constant clock.

We show some useful relations for proving substitutability between two distributed objects.

Proposition 4.5 $\forall P_1, P_2 \in \mathcal{P} : P_1 \sim_\theta P_2$, $\forall Q \in \mathcal{P}$ such that $Q \in \mathcal{P}$ contains no timeout operator.

$$\begin{array}{llll} (1) & \alpha.P_1 & \sim_\theta & \alpha.P_2 \\ (3) & P_1 \setminus L & \sim_\theta & P_2 \setminus L \\ (5) & P_1|Q & \sim_\theta & P_2|Q \end{array} \qquad \begin{array}{llll} (2) & P_1 + Q & \sim_\theta & P_2 + Q \\ (4) & P_1[f] & \sim_\theta & P_2[f] \end{array}$$

where the same results holds for $\simeq_\theta$. □

Since distributed object-oriented computing is based on interaction among objects executing concurrently, the following substitutability for parallel composition provides a method to verify interactions among distributed objects.

Proposition 4.6 $\forall P_1, P_2 \in \mathcal{P}, \forall S \in \mathcal{S}$, *If* $P_1 \simeq_\theta P_2$, *then* $P_1|[\![S]\!]_\theta \sim_\theta P_2|[\![S]\!]_\theta$ □

This proposition shows that if an object according a clock θ interacts with one of two objects which cannot be distinguished by an observer following the same clock θ, the object itself cannot notice any difference in its interactions with either object.

Preorder on Clocks

It is well known that any actual clocks are different in their measurement unit and precision. Based on such unit and precision, we here formulate some order relations over local clocks and further study interesting relationships between the preorders and the timed bisimulations.

We define order relations over clock functions.

Definition 4.7 Let $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_\ell$ be local time domains,

(i) *Granularity Preorder* Let $\theta_1 : \mathcal{T}_1 \to \mathcal{T}_G$ and $\theta_2 : \mathcal{T}_2 \to \mathcal{T}_G$.
If $\forall t_2 \in \mathcal{T}_2, \exists t_1 \in \mathcal{T}_1 : \{\theta_1(t_1)\} = \{\theta_2(t_2)\}$, *then* θ_1 *is finer than* θ_2, *written as* $\theta_1 \trianglelefteq \theta_2$.

(ii) *Precision Preorder* Let $\theta_1 : \mathcal{T}_\ell \to \mathcal{T}_G$, and $\theta_2 : \mathcal{T}_\ell \to \mathcal{T}_G$.
If $\forall t_\ell \in \mathcal{T}_\ell : \{\theta_1(t_\ell)\} \subseteq \{\theta_2(t_\ell)\}$, *then* θ_1 *is more precise than* θ_2, *written as* $\theta_1 \sqsubseteq \theta_2$.

□

Intuitively, $\theta_1 \trianglelefteq \theta_2$ means that one time unit of θ_2 corresponds to more than one time unit of θ_1. $\theta_1 \sqsubseteq \theta_2$ means that θ_2 is less accurate than θ_1.

Proposition 4.8 $\trianglelefteq$ and $\sqsubseteq$ are preorder relations. □

We here show relationships between $\trianglelefteq$ and timed bisimilarities.

Proposition 4.9 $\forall P_1, P_2 \in \mathcal{P}$, $\theta_1 \trianglelefteq \theta_2$, then

(1) *If* $P_1 \sim_{\theta_1} P_2$, *then* $P_1 \sim_{\theta_2} P_2$
(2) *If* $P_1 \simeq_{\theta_1} P_2$, *then* $P_1 \simeq_{\theta_2} P_2$ □

Intuitively, this property will be exemplified by the following example: if two objects cannot be distinguished by an observer following an accurate clock whose time unit is one second, then they cannot be distinguished by another observer having an accurate clock with the unit of one minute.

Below we present relationships between $\sqsubseteq$ and timed bisimilarities.

Proposition 4.10 $\forall P_1, P_2 \in \mathcal{P}$, $\theta_1 \sqsubseteq \theta_2$, then

(1) *If* $P_1 \sim_{\theta_1} P_2$, *then* $P_1 \sim_{\theta_2} P_2$
(2) *If* $P_1 \simeq_{\theta_2} P_2$, *then* $P_1 \simeq_{\theta_1} P_2$ □

From Proposition 4.9 and 4.10, we assume $\theta_1 \trianglelefteq \theta_2$ and $\theta_2 \sqsubseteq \theta_3$, then for any P and Q, if $P_1 \sim_{\theta_1} P_2$ then $P_1 \sim_{\theta_3} P_2$.

The strictness of $\sim_\theta$ (and $\simeq_\theta$) depends on clock θ. The orders of clocks are available in specifications with respect to different time scales and precisions.

Example 4.11 We suppose two clocks $\theta_{\pm 5\%}$ and $\theta_{\pm 10\%}$ whose worst measurement errors are $\pm 5\%$ and $\pm 10\%$ respectively.

$$\theta_{\pm 5\%}(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta_{\pm 5\%}(t-1) + \delta_{\pm 5\%} & \text{if } t > 0 \end{cases} \qquad \delta_{\pm 5\%} \in \{95, ..100, ..105\}$$

$$\theta_{\pm 10\%}(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta_{\pm 10\%}(t-1) + \delta_{\pm 10\%} & \text{if } t > 0 \end{cases} \qquad \delta_{\pm 10\%} \in \{90, ..100, ..110\}$$

Immediately we have $\theta_{\pm 5\%} \sqsubseteq \theta_{\pm 10\%}$. Also, from Proposition 4.10, $\sim_{\theta_{\pm 5\%}}$ is more strict than $\sim_{\theta_{\pm 10\%}}$. We have for any P_1 and P_2, if $P_1 \sim_{\theta_{\pm 5\%}} P_2$, then $P_1 \sim_{\theta_{\pm 10\%}} P_2$. Clearly the converse is not true. □

Examples on Verification of Distributed Objects

For the remainder of this section we will present some examples of verification of distributed objects by using the timed bisimilarities. We first illustrate how to equate two objects whose temporal properties are different.

Example 4.12 *Verification for Real-Time Objects*
We consider two server objects as given below. The informal exposition of the objects has already been described in Example 3.11.

$$\begin{aligned} Server_A &\stackrel{\text{def}}{=} req.\langle \mathbf{0}, \overline{ret}.Server_A \rangle_8 \\ Server_B &\stackrel{\text{def}}{=} req.\langle \mathbf{0}, \overline{ret}.Server_B \rangle_9 \end{aligned}$$

We assume a constant clock $\theta(t) \stackrel{\text{def}}{=} 6t$. An observer according to θ cannot distinguish between temporal properties of the objects, i.e. their execution times of 8 and 9 time units, because of $1 \in \{\theta^{-1}(8)\}$ and $1 \in \{\theta^{-1}(9)\}$. Hence, $\sim_\theta$ can equate them even if their temporal properties are different.

$$Server_A \sim_\theta Server_B \quad c.f. \quad Server_A \not\sim Server_B$$

This holds on $\simeq_\theta$ as well as $\sim_\theta$. □

- $\sim_\theta$ (and $\simeq_\theta$) provides a useful method to verify real-time objects with non-strict time constraints. For example, let $Server_A$ be a specification of the server object and $Server_B$ be an implementation of the object. The above result shows that the implementation completely satisfies the behavioral requirements in the specification, and that the temporal differences between them is within a permissible bound specified in terms of θ.
- Let us suppose a clock θ' such that $\theta \trianglelefteq \theta'$. By Proposition 4.9 we can easily prove $Server_A \sim_{\theta'} Server_B$. For example, let $\hat{\theta}(t) \stackrel{\text{def}}{=} 12t$ then $\theta \trianglelefteq \hat{\theta}$. Hence, we have $Server_A \sim_{\hat{\theta}} Server_B$.

Next we show an example to illustrate how to equate distributed objects following different clocks.

Example 4.13 *Verification for Distributed Objects with Inaccurate Clocks*
Let the client object be as follows:

$$Client \stackrel{\text{def}}{=} \overline{req}.\langle ret.\mathbf{0}, Client \rangle_2$$

We assume that the program is allocated on two processors with local clocks, θ_1 and θ_2, whose time unit may vary from 9 to 11 and from 10 to 12, respectively.

$$\theta_1(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta_1(t-1) + \delta_1 & \text{if } t > 0 \end{cases} \quad \text{where } \delta_1 \in \{9, 10, 11\}$$

$$\theta_2(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta_2(t-1) + \delta_2 & \text{if } t > 0 \end{cases} \quad \text{where } \delta_2 \in \{10, 11, 12\}$$

An observer according to the following variable clock $\theta'(t)$ equates these objects.

$$\theta'(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t = 0 \\ \theta'(t-1) + \delta' & \text{if } t > 0 \end{cases} \quad \text{where } \delta' \in \{14, 15, 16\}$$

$$[\![Client]\!]_{\theta_1} \sim_{\theta'} [\![Client]\!]_{\theta_2} \quad c.f. \quad [\![Client]\!]_{\theta_1} \not\sim [\![Client]\!]_{\theta_2}$$

This result shows that $\sim_\theta$ and $\simeq_\theta$ can equate two distributed objects even if the units and precisions of their clocks are different. □

We illustrate how to verify whether distributed objects can be substituted for each other.

Example 4.14 *Substitutability between Distributed Objects*
Recall the two clients $[\![Client]\!]_{\theta_1}$ and $[\![Client]\!]_{\theta_2}$, and clock θ' already presented in Example 4.13. We have:

$$[\![Client]\!]_{\theta_1} \simeq_{\theta'} [\![Client]\!]_{\theta_2}$$

We suppose a server object being executed by a processor having the clock θ'. From Proposition 4.6, we have:

$$Server \stackrel{\text{def}}{=} req.\langle \mathbf{0}, \overline{ret}.Server \rangle_1$$

$$[\![Client]\!]_{\theta_1} | [\![Server]\!]_{\theta'} \sim_{\theta'} [\![Client]\!]_{\theta_2} | [\![Server]\!]_{\theta'}$$

Two pessimistically timed bisimilar objects on θ can be substitutable for each other as long as they interact with any objects on the local clock θ. Therefore $\sim_\theta$ and $\simeq_\theta$ provide a method to verify reusability and substitutability for distributed objects. □

5 Conclusion

In this paper we have seen a way to formulate local time properties in distributed computing, along with developing a formalism based on a minor temporal extension of CCS [14]. The extension is based on the notion of local time and thus allows us to model various local time aspects in distributed computing, such as inaccurate clocks on local processors and timeout handling. The formalism provides a theoretical framework to describe and analyze both temporal and behavioral properties of distributed objects with temporal uncertainties and interactions among them.

Based on the notion of bisimulation [14, 19], we defined bisimulations with respect to local time. These bisimulations can equate two objects whose functional behaviors completely match and whose timings are different within a given bound. The bisimulations are appropriate and useful to verify distributed objects and real-time objects with non-strict time constraints.

Finally, we would like to point out some further issues. The formalism is based on synchronous communication but in many distributed systems asynchronous communication may seem more appropriate. We plan to develop a calculus based on

asynchronous communication. Particularly, we believe that the study of time properties for asynchronous communications will provide us with concepts for programming languages for distributed systems[6]. We are interested in investigating timed bisimulation based on observation concept which can ignore internal behavior, i.e. τ-transition. Besides, the extension for representing local time presented in this paper is essentially independent of DtCCS. We are interested in whether the extension can be applied to other timed formalisms based on global time, such as other timed calculi, real-time temporal logic, and timed Petri nets.

Acknowledgements

We would like to thank an anonymous referee for providing many constructive and valuable suggestions. We also thank Professor S. Matsuoka, and R. Pareschi for significant suggestions. We are grateful to K. Takashio for stimulating comments and discussions. We heartily thank V. Vasconcelos for very insightful comments on an earlier version of this paper.

References

[1] Black, A., Hutchinson, N., July, E., and Levy, H., *Object Structure in the Emerald System*, Proceedings of ACM OOPSLA'86, November, p78-86, 1986.

[2] Boudol, G., Castellani, I., Hennessy, M., and Kiehn, A., *A Theory of Processes with Localities*, Proceedings of CONCUR'92, LNCS 630, p108-122, August, 1992.

[3] Castellani, H., and Hennessy, M., *Distributed Bisimulation*, Journal of ACM, Vol.36, No.4, p887-911, 1989.

[4] Corsetti, E., Montanari, A., and Ratto, E., *Dealing with Different Granularities in Formal Specifications of Real-Time Systems*, Real-Time Systems, Vol.3, No.2, May, 1991.

[5] Hansson, H., and Jonsson, B., *A Calculus of Communicating Systems with Time and Probabilities*, Proceedings of 11th IEEE Real-Time Systems Symposium, p278-287, December, 1990.

[6] Hennessy, M., *On Timed Process Algebra: a Tutorial*, Technical Report 2/93, University of Sussex, 1993

[7] Hobbs, J. R., *Granularity*, Proceedings of 9th International Joint Conference Artificial Intelligence, August, 1985.

[8] Honda, K., and Tokoro, M., *An Object Calculus for Asynchronous Communication*, Proceedings of ECOOP'91, LNCS 512, p133-147, June, 1991.

[9] Jefferson, D. R., *Virtual Time*, ACM TOPLAS, Vol.7, No.3, 1985.

[10] Kopetz, H., *Clock Synchronization in Distributed Real-Time Systems*, IEEE Transactions on Computers, Vol.36, No.8, p933-940, August, 1987.

[11] Krishnan, P., *Distributed CCS*, Proceedings of CONCUR'91, LNCS 527, p393-407, August, 1991.

[12] Lamport, L., *Time, Clocks, and the Ordering of Events in a Distributed System* Communication of the ACM, Vol.21, No.7, p558-565, July, 1978,

[6] The reader may refer to the authors' preliminary work in this context, e.g. [23].

[13] Lundelius, J., and Lynch, N., *An Upper and Lower Bound for Clock Synchronization*, Information and Control, Vol.62, p190-204, 1984.

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

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

[16] Montanari, A., Ratto, E., Corsetti, E., and Morzeniti, A., *Embedding Time Granularity in Logical Specification of Real-Time Systems*, Proceedings of EUROMICOR'91, Workshop on Real-Time Systems, p88-97, June, 1991.

[17] Nicollin, X., and Sifakis, J., *The Algebra of Timed Process ATP: Theory and Applications*, IMAG Technical Report, RT-C26, 1990.

[18] Nierstrasz, O. M., and Papathomas, M., *Viewing Objects as Patterns of Communicating Agents*, Proceedings of ECOOP/OOPSLA'90, October, p38-43, 1990.

[19] Park, D., *Concurrency and Automata on Infinite Sequences*, Proceedings of Theoretical Computer Science, LNCS 104, p167-187, 1981.

[20] Plotkin, G. D, *A Structural Approach to Operational Semantics*, Technical Report, Department of Computer Science, Arhus University, 1981.

[21] Satoh, I., and Tokoro, M., *A Formalism for Real-Time Concurrent Object-Oriented Computing*, Proceedings of ACM OOPSLA'92, p315-326, October, 1992.

[22] Takashio, K., and Tokoro, M., *DROL: An Object-Oriented Programming Language for Distributed Real-time Systems*, Proceedings of ACM OOPSLA'92, October, 1992.

[23] Tokoro, M., and Satoh, I., *Asynchrony and Real-Time in Distributed Systems*, US/Japan Seminar on Parallel Symbolic Computing, October, 1992.

[24] Yi, W., *CCS + Time = an Interleaving Model for Real Time Systems*, Proceedings of Automata, Languages and Programming'91, LNCS 510, p217-228, 1991.

[25] Yonezawa, A., and Tokoro, M., editors, *Object-Oriented Concurrent Programming*, MIT Press, 1987.