A Process Calculus for Asynchronously Communication with Transmission Delay

Ichiro Satoh

satoh@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

1 Introduction

Distributed systems consist of multiple processors which corporate with one another by sending messages over communication networks. These communication networks often have transmission delay which results in a physical and logical function of geographical distance, communication bandwidth, and communication protocol overhead. The delay essentially characterizes distributed systems and one of the most difficulty in developing correct distributed systems. For example, the delay often affects the arrival time of messages and thus causes inexpectabel causalities among events on different processors. From the reason of efficency, in distributd systems communication among processors is often realized as an asynchronous form in order to reduce synchronization cost including communication delay between a sender processor and a receiver one. In order to construct and verify programs for distributed systems, we must take this delay into consideration. The objective of this paper is to propose a framework for describing quantitatively communication delay and analyzing the influence of the delay upon interactions among distributed processes. The framework is formulated based on a process algebra, for example ACP [1], CCS [10], π-calculus [11].

There have indeed been some process algebras for distributed systems. Most of them are extensions of existing process algebras with some natures of distributed computing: process locations [4, 9], port passing [7, 10, **?**], and local time, i.e., the inexistence of global time information [**?**], and not communication delay. In this paper, we dare to not introduce these natures, including process location in order to concentrating on investigating communication delay and its influence. [1] . The framework proposed in this paper is an extension of a timed process algebra [15] with the ability of expressing parallelism, communication delay, and asynchronous communication. Through its expressiveness power, we try to strictly analyze the influence of the delay upon the behavioral and temporal properties of interactions among distributed real-time processes.

[1] We leave some relationships between process location and communication delay to our other paper [**?**]

Moreover, in this paper we develop a verification technique for distributed real-time systems by means of a prebisimulation relation which can relate two distributed processes with respect to their relative speeds. If two processes are behaviorally equivalent and if the first process can execute its behaviors faster than the second one, the equality presents the first is *faster* than the second. We often have a chance to replace an old fashioned computer (i.e., a slower subsystem) of a (working) system with its latest fashioned computer (i.e., its faster one). We need to garantee the first computer can substitute for the second computer in the whole systems and can perform faster than the second one. Therefore, such a relation with respect to speed provides a powerful method construct and improve real distributed systems.

The organization of this paper: In the next section we present the basic idea of our formalism and then define the formalism. In Section 3 we define a timed prebisimulation which can relate processes according to their speeds. In Section 4 we present some examples to demonstrate the usefulness of the formalism. The final section briefly surveys some related work and contains some concluding remarks. Some proofs are given only in outline for lack of space. For details please consult [**?**].

2 Definition

This section first gives an informal exposition of our formalism and then defines its syntax and its transitional semantics. Our formalism is formulated by extending of CCS [10] with the ability of expressing communication delay, delaying processing, and asynchronous message. Its syntax and semantics are essentially the same as those of CCS, except for these extensions.

2.1 Basic Framework

Before formally defining the formalism, we summarize our basic idea of modeling communication delay and asynchronous comunicaion. Parallel processes communicate each other via asynchronous one-way message passing, Each message has its name and is received by a process which has an input port with the same name. The sender process can send a message and continue its execution without without waiting the reception of the message by another process. Communication delay for a message is assumed to be known before [2] and the order of arrival of messeages are indeterminate.

Communication Delay Communication delay is a minimum time which a message takes to be possible to be received after the message is sent out from its output port (or channel). In our formalims, delay time for each message sending is specified in the output port for the message. For example, a message whose name is a and delay is t_1 time units is described as: $a\uparrow^{t_1}$. Besides, it is often convenient to specify communication delay for a message at the input port. We introduce the description of communication delay in input ports (or channels) to be specified. For example, $a\downarrow^{t_2}$shows at least t_2 time units pass before a message named a is received at port a. At a result, the communicaton delay of a

[2] We here assume the delay can easily extend the formalism to deal with non-determinsitc delay.

message is the total amount of the delay time specifed in an output port for the message and that in an input one.

Time Passing and Delaying Processing: In our formalism time is discrete, instead of continuous. The passage of time is represented as a special transition which synchronously proceed in all processes. Also, many real distributed systems have several behaviors dependent on time. We introduce a delaying operation which explicitlry suspends execution for a specified number of units of time, similarly to the *delay* command in Ada.

Asynchronous Communication In synchronous communication, the sender need to wait an acknowledge message from the receiver and thus it must be idle at least for the round trip communication delay. Therefore, in distributed systems, communication is often realized as an asynchronous one. However, most of process algebras are essentially based on synchrnous communication and thus we need extensions of them with the expressiveness of asynchronous communication. We express a asynchronous message sending by creating a process which can engage only in an input action with the same name of the message. This is the way taken in [2, 7, 9].

Definition

Here we present the syntax and the semantics of the formalism. The syntax is an extension of Milner's CCS [10] by introducing asynchronous output action, delaying. first define notation conventions which we will follow hereafter.

Definition 2.1

- Let $\mathcal{M}$ be an infinite multiple set of message names, ranged over by a,b,....
- Let $\mathcal{T}$ denote the set of the integers. □

In our formalism, distributed processes are described by means of expressions as defined below.

Definition 2.2 The set $\mathcal{P}_{\sum}$ of expressions, ranged over by $P, P_1, P_2, \ldots$ is the smallest set which contains the following expressions. When we assume $I \stackrel{\text{def}}{=} \{0,1\}$ in $\sum_{i\in I} a_i\downarrow^{t_i}.P_i$ (I is an index set), $\mathcal{P}_{\sum}$ is especially denoted as P.

$$
\begin{array}{rcll}
P & ::= & a\uparrow^t.P & (\textit{Message Sending}) \\
 & | & \sum_{i\in I} a_i\downarrow^{t_i}.P_i & (\textit{Alternatively Message Receiving}) \\
 & | & (t)P & (\textit{Delaying Process}) \\
 & | & P|P & (\textit{Parallel Execution}) \\
 & | & P \setminus L & (\textit{Message Restriction}) \\
 & | & A \stackrel{\text{def}}{=} P & (\textit{Recursive Definition})
\end{array}
$$

where $\sum_{i\in\emptyset} a_i\downarrow^{t_i}.P_i$ is $\mathbf{0}$, t is an element of $\mathcal{T}$. L is a subset of $\mathcal{M}$, and A is an element of the process constant set $\mathcal{C}$, ranged over $A, B, \ldots$. We shall often use the more simple notation $a\uparrow^t$instead of $a\uparrow^t_;$ and $a_0\downarrow^{t_0}.P_0 + \cdots + a_i\downarrow^{t_i}.P_i + \cdots$ instead of $\sum_{i\in\{0,\ldots i,\ldots} a_i\downarrow^{t_i}.P_i$. □

Remark 2.3 Intuitively, the meaning of constructors on $\mathcal{P}_{\sum}$ are as follows:

- $\mathbf{0}$ presents a terminated and deadlocked process.
- $a\uparrow^t.P$ sends a message with name a and can continue to execute P without blocking before the message is received by another process. t represents the minimum time which message a takes to be receivable, that is, t corresponds to the sending cost of message a on the sender.
- $\sum_{i\in I} a_i\downarrow^{t_i}.P_i$ can receive a message, named a_i and then behaves as P_i. t_i is the minimum time which the message takes to be received by the message input port $a_i\downarrow$
- $(t)P$ presents a process which is idle for t time units and then behaves as P.
- $P_1|P_2$ allows P_1 and P_2 to execute in parallel.
- $P \setminus L$ encapsulates all messages in a set L.
- $A \stackrel{\text{def}}{=} P$ represents that A is defined equation as P. We allow P to include A, that is, a recursive definition.

The operational semantics of our language is here strucutually given by two kinds of state transition relations: $\rightarrow \subseteq \mathcal{P}_{\sum} \times \mathcal{P}_{\sum}$ (called *behavioral transition*) and $\rightsquigarrow \subseteq \mathcal{P}_{\sum} \times \mathcal{P}_{\sum}$ (called *temporal transition*). Notice that the first transtions are not labeled by any action names, although most of CCS-like process calculi are formulated as labeled transition systems. This is because it was essentially convenient for CCS to use labeled transions to record the possible input or output actions of processes in CCS's handshake communication, but in our formalism the process communication is materialized by a message passing mechanism instead of any handshake one.

Moreover, in order to concentrate on investiaging an inequality between processes in the following section, we beforehand provide a syntactically equivalent formulation for some process expressions which can be treated as the same evidently. The formulation is givend by defining a structural congrucence $\equiv$ as below. This way is the way taken by Milner in [11] to deal with π-calculus.[3]

Definition 2.4 $\equiv$ is the smallest structural congruence which contains the following relations:

$$P_1|P_2 \equiv P_2|P_1 \quad P_1|(P_2|P_3) \equiv (P_1|P_2)|P_3 \quad P|\mathbf{0} \equiv P$$
$$P_1 + P_2 \equiv P_2 + P_1 \quad P_1 + (P_2 + P_3) \equiv (P_1 + P_2) + P_3 \quad P + P \equiv P \quad P + \mathbf{0} \equiv P$$
$$(a\uparrow^t.\mathbf{0}|P) \setminus L \equiv a\uparrow^t.\mathbf{0}|P \setminus L \text{ where } a \notin L \qquad P \setminus L_1 \setminus L_2 \equiv P \setminus L_2 \setminus L_1 \quad \mathbf{0} \setminus L \equiv \mathbf{0}$$
$$(t_1)(t_2)P \equiv (t_1 + t_2)P \quad (0)P \equiv P \quad (t)\mathbf{0} \equiv \mathbf{0}$$ □

Now we present the operational rules for our language.

[3] Please note that the semantics can be easily reformulated in the style of Plotkin's SOS [**?**] without the use of the structural congruence.

Definition 2.5 *Behavioral Transition Rules* are given by the least relations $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$ as follows:

$$
\begin{array}{rll}
a\uparrow^t.P \ \rightarrow \ a\uparrow^t.\mathbf{0} \mid P & & (P \text{ is not } \mathbf{0}) \\
\sum_{i\in I} a_i\downarrow^{t_i}.P_i \mid a_i\uparrow^t.P \ \rightarrow \ P_i \mid P & & (t + t_i \leq 0) \\
P_1 \ \rightarrow \ P_1' & \text{implies} & P_1 \mid P_2 \ \rightarrow \ P_1' \mid P_2 \\
P \ \rightarrow \ P' & \text{implies} & P \setminus L \ \rightarrow \ P' \setminus L \\
P \ \rightarrow \ P' & \text{implies} & A \ \rightarrow \ P' \quad (A \stackrel{\text{def}}{=} P) \\
P_1 \equiv P_1', \quad P_1 \ \rightarrow \ P_2, \quad P_2 \equiv P_2' & \text{implies} & P_1' \ \rightarrow \ P_2'
\end{array}
$$

where we shall often simply written $(\equiv)^* \rightarrow (\equiv)^*$ as $\rightarrow$. □

Definition 2.6 *Temporal Transition Rules* are given by the least relations $\rightsquigarrow \subseteq \mathcal{P} \times \mathcal{P}$ as follows:

$$
\begin{array}{rll}
\mathbf{0} \dot{\rightsquigarrow} \mathbf{0} & & \\
(t)P \dot{\rightsquigarrow} (t-1)P & & (t \geq 0) \\
a\uparrow^t.\mathbf{0} \dot{\rightsquigarrow} a\uparrow^{t-1}.\mathbf{0} & & \\
\sum_{i\in I} a_i\downarrow^{t_i}.P_i \dot{\rightsquigarrow} \sum_{i\in I} a_i\downarrow^{t_i}.P_i & & \\
P_1 \dot{\rightsquigarrow} P_1', \quad P_2 \dot{\rightsquigarrow} P_2' & \text{implies} & P_1|P_2 \dot{\rightsquigarrow} P_1'|P_2' \\
P \dot{\rightsquigarrow} P' & \text{implies} & P \setminus L \dot{\rightsquigarrow} P' \setminus L \\
P \dot{\rightsquigarrow} P' & \text{implies} & A \dot{\rightsquigarrow} P' \quad (A \stackrel{\text{def}}{=} P) \\
P_1 \equiv P_1', \quad P_1 \dot{\rightsquigarrow} P_2, \quad P_2 \equiv P_2' & \text{implies} & P_1' \dot{\rightsquigarrow} P_2' \\
P \dot{\rightsquigarrow} P', \quad P \not\rightarrow & \text{implies} & P \rightsquigarrow P'
\end{array}
$$

where we shall often simply written $(\equiv)^* \rightsquigarrow (\equiv)^*$ as $\rightsquigarrow$. □

Notice that these rules directly respect the informal expositions of the constructs given proviously. We here give some technical remarks on the semantics.

Remark 2.7

(1) According to the rule: $a\uparrow^t.P \ \rightarrow \ a\uparrow^t.\mathbf{0} \mid P$, sending an asynchronous message with delay t time units is represented by creating a process which can engage only in an input action with the same name of the message after t time units. The rule: $a\uparrow^t.\mathbf{0} \ \rightsquigarrow \ a\uparrow^{t-1}.\mathbf{0}$ decreases t in $a\uparrow^t.\mathbf{0}$ by one, when one time unit passes. t in $a\uparrow^t.\mathbf{0}$ is more than 0, message a cannot be recevied.

(2) A message, $a_i\uparrow^t.\mathbf{0}$, can be engaged by only a process with the same input port, $a_i\downarrow^{t_i}.P_i$ according to the rule: $\sum_{i\in I} a_i\downarrow^{t_i}.P_i \mid a_i\uparrow^t.P \ \rightarrow \ P_i \mid P$ (where $t + t_i \leq 0$). The condition $t_1 + t_2 \leq 0$ represents that, in order to be received by $a_i\downarrow^{t_i}.P_i$, the message needs to be idle for at least $t_1 + t_2$ time units after it is sent out. As a result, $t_1 + t_2$ corresponds to the minimum communication delay of the message transmission.

In order to illustrate how to describe distributed processes in our formalism we present some simple examples.

Example 2.8 We suppose interaction between two processes: $_{(1)}a\uparrow^2.c\downarrow^4.\mathbf{0}$ and $_{(2)}(a\downarrow^1.P_1 + b\downarrow^3.P_2)$. The former process is idle for 1 time unit and then sends message a with 3 time units communication delay and waits message b at an input port with 4 time units delay. The latter is idle for 2 time units and then waits message a an input port with 1 time units communication delay, or message c at one with 3 time units communication delay. The interaction between these processes is described as the following parallel composition:

$$\begin{aligned}
{}_{(1)}a\uparrow^2.b\downarrow^4.\mathbf{0} \mid {}_{(2)}(a\downarrow^1.P_1 + c\downarrow^3.P_2) &\rightsquigarrow a\uparrow^2.b\downarrow^4.\mathbf{0} \mid {}_{(1)}(a\downarrow^1.P_1 + c\downarrow^3.P_2) \\
&\to a\uparrow^2.\mathbf{0} \mid b\downarrow^4.\mathbf{0} \mid {}_{(1)}(a\downarrow^1.P_1 + c\downarrow^3.P_2) \\
&\rightsquigarrow a\uparrow^1.\mathbf{0} \mid b\downarrow^4.\mathbf{0} \mid a\downarrow^1.P_1 + c\downarrow^3.P_2 \\
&\rightsquigarrow a\uparrow^0.\mathbf{0} \mid b\downarrow^4.\mathbf{0} \mid a\downarrow^1.P_1 + c\downarrow^3.P_2 \\
&\rightsquigarrow a\uparrow^{-1}.\mathbf{0} \mid b\downarrow^4.\mathbf{0} \mid a\downarrow^1.P_1 + c\downarrow^3.P_2 \\
&\to b\downarrow^4.\mathbf{0} \mid P_1
\end{aligned}$$

Notice that message a can be received after four time units.

Definition 2.9 Let transitive translations denoted as follows:

$$\begin{aligned}
P \longrightarrow\!\!\!\!\to P' &\stackrel{\text{def}}{=} P \to \cdots \to P' \\
P \rightsquigarrow\!\!\!\!\to^n P' &\stackrel{\text{def}}{=} P \underbrace{\longrightarrow\!\!\!\!\to \rightsquigarrow \longrightarrow\!\!\!\!\to \cdots \longrightarrow\!\!\!\!\to \rightsquigarrow \longrightarrow\!\!\!\!\to}_{n \ times} P'
\end{aligned}$$

where we will sometimes abbreviate $P \rightsquigarrow\!\!\!\!\to^1 P'$ as $P \rightsquigarrow\!\!\!\!\to P'$.

We show a notable relation between the passage of time and the delay time of messages.

Lemma 2.10 We assume $\forall l, m$: $P \rightsquigarrow\!\!\!\!\to^m b\uparrow^{-l}.\mathbf{0}|P'$:

(i) If $\forall n$: $m - l \leq n \leq m$ then, for some $\dot{P}$ such that $P \rightsquigarrow\!\!\!\!\to^n b\uparrow^{-l+m-n}.\mathbf{0}|\dot{P}$ and $\dot{P} \rightsquigarrow\!\!\!\!\to^{m-n} P'$.

(ii) If $\forall n$: $m \leq n$ then, for some $\dot{P}$ such that $P \rightsquigarrow\!\!\!\!\to^n b\uparrow^{-l+m-n}.\mathbf{0}|\dot{P}$ and $P' \rightsquigarrow\!\!\!\!\to^{n-m} \dot{P}$.

Proof. From Definition 2.5 and 2.6. □

Now we are ready to define an inequality relation beteewn processes with respect to their relative speeds. For example, we first suppose two processes: $A_1 \stackrel{\text{def}}{=} {}_{(1)}a\uparrow^0.P$ and $A_2 \stackrel{\text{def}}{=} {}_{(3)}a\uparrow^0.P$. We would consider to the process A_1 to be fster than the process A_2, because A_1 can send message a faster than A_2. However, there is an underiable problem in further providing a precongruence with respect to speeds, in order to specify substitutability between these ordered processes. We suppose a process: $B \stackrel{\text{def}}{=} a\downarrow.b\uparrow.\mathbf{0} + c\downarrow.d\uparrow.\mathbf{0} \mid {}_{(2)}c\uparrow.\mathbf{0}$. Unfortunately, we cannot gurantee that corporations between the faster process (A_1) and B is always faster than one between the slower process (A_2) and B, because of $A_1|B \rightsquigarrow\!\!\!\!\to^2 b\uparrow.\mathbf{0}$ but $A_1|B \not\rightsquigarrow\!\!\!\!\to^n b\uparrow.\mathbf{0}$ ($n \geq 2$). This problem arises due to the existance of *timeout* context in B. Timeout handling is to restrict its desired computation at first and proceed an alternative computation, if the specified time passes without the desired computation are performed. That is to say, timeout handling is an operation to lose the capability of lose the ability to do other input port ($a\Downarrow$) due to the passing of time. In order to define a rational precongruence with respect to speeds, we here adopt an approach to subtlely weaken the expressive power of our language in order to restrict the expressiveness of the above undesirable timeout context.

3 Compositionability among Distributed Processes

In this section we propose a verification method to analyze corprationability among distributed processes with asynchronous message passing. There are a lot of timed equivalence relation over processes. The relations equate two processes only when their temporal and behavioral properties completely match. However, such equivalence relations are often too strict because distributed systems have a lot of nondeterministic properties. We here propose a order relation between two processes by extending the notion of bisimulation [**?**, 10]. The relation shows that they are behavioral equivalent and one of them can execute its behaviors *faster* than the another. Next, we show such a order relation is very useful for proving substitutability between two remote processes.

We first the notation of messages.

Definition 3.1

$$\prod_{i\in I} a_i\uparrow^{t_i} \stackrel{\text{def}}{=} a_1\uparrow^{t_1}.\mathbf{0} \mid \cdots \mid a_i\uparrow^{t_i}.\mathbf{0} \mid \cdots \qquad (i \in I)$$

Definition 3.2 A binary relation $\mathcal{R}_{[t_1,t_2]}$ ($\subseteq (\mathcal{P}\times\mathcal{T})\times(\mathcal{P}\times\mathcal{T})$) is a $[t_1,t_2]$-*timed prebisimulation* if $(P_1,P_2)\in\mathcal{R}_{[t_1,t_2]}$ (where $t_1\le t_2$) implies, for all $M\subset(\mathcal{M}\cup\{\epsilon\})$ and for all $k_a^1,k_a^2\in\mathcal{T}$ such that $0\le k_a^1+t_1\le k_a^2+t_2$,

(i) $\forall l_b^1\forall m_1\forall P_1'$: $\Pi_{a\in M}a\uparrow^{k_a^1} \mid P_1 \rightsquigarrow^{m_1} \Pi_{b\in N}b\uparrow^{-l_b^1} \mid P_1'$ then $\exists l_b^2\exists m_2\exists P_2'$: $\Pi_{a\in M}a\uparrow^{k_a^2} \mid P_2 \rightsquigarrow^{m_2} \Pi_{b\in N}b\uparrow^{-l_b^2} \mid P_2'$ and $l_b^2\le l_b^2$ and $(P_1',P_2')\in\mathcal{R}_{[t_1+m_1,t_2+m_2]}$

(ii) $\forall l_b^2\forall m_2\forall P_2'$: $\Pi_{a\in M}a\uparrow^{k_a^2} \mid P_2 \rightsquigarrow^{m_2} \Pi_{b\in N}b\uparrow^{-l_b^2} \mid P_2'$ then $\exists l_b^1\exists m_1\exists P_1'$: $\Pi_{a\in M}a\uparrow^{k_a^1} \mid P_1 \rightsquigarrow^{m_1} \Pi_{b\in N}b\uparrow^{-l_b^1} \mid P_1'$ and $l_b^1\le l_b^1$ and $(P_1',P_2')\in\mathcal{R}_{[t_1+m_1,t_2+m_2]}$

where we assume $m_1,m_2,l_b^1,l_b^2\in\mathcal{T}_{\ge 0}$.

We state the informal meaning of $\mathcal{R}_{[t_1,t_2]}$. We assume that t_1 time units passes in P_1 and t_2 time units passes in P_2.[4] In Definition 3.2 (i), a conceptual observer sends P_1 messages which can be sent since the begining of the experiment, that is, t_1 time units ago. It waits messages which P_1 returns. Also, the observer sends the same messages to P_2 after the timings of its sending them to P_1 and then waits messages which P_2 return. If the messages from P_1 and those from P_2 are coincide in their message contents and the timings of the messages from P_1 are earlier than those from P_2, P_2 can simulate P_1 even more slowly. In Definition 3.2 (ii), we analyze P_1 can simulate P_2 even faster. Note that the observer sends messages which can be sent to the processes even at their pasts.

From Definition 3.2, we can easily show the following properties.

Proposition 3.3 Let $P,P_1,P_2\in\mathcal{P}$ then,

(1) $(P,P)\in\mathcal{R}_{[t,t]}$ (2) $(P_1,P_2)\in\mathcal{R}_{[t_1,t_2]}$ and $(P_2,P_3)\in\mathcal{R}_{[t_2,t_3]}$ then $(P_1,P_3)\in\mathcal{R}_{[t_1,t_3]}$
(3) If $\mathcal{R}_{[t_1,t_2]}$ is a timed prebisimulation, $\bigcup_{i\in I}\mathcal{R}_{[t_1,t_2]}$ is a timed prebisimulation.

[4] Note that t_1 is always smaller than t_2.

Note that $\mathcal{P}_{\sum}$ cannot preserve (1) in the above propostion. We show a countexample: let $A \stackrel{\text{def}}{=} a_1\downarrow.b_1\downarrow.\mathbf{0} + a_2\downarrow.b_2\downarrow.P_2|(1)a_2\uparrow.\mathbf{0}$, then $a_1\uparrow.\mathbf{0}|A \rightarrowtail\!\!\!\!\!\rightarrow^2 b_2\downarrow.\mathbf{0}|a_2\uparrow^{-1}$ but $(2)a_1\uparrow.\mathbf{0}|A \rightarrowtail\!\!\!\!\!\rightarrow^2 a_1\uparrow.\mathbf{0}|b_2\downarrow.\mathbf{0}$. This is because $\sum a_i\downarrow^{t_i}.P_i$ $(i \geq 2)$ in $\mathcal{P}_{\sum}$ can often restrict the alternative input ports which has not recevied any messages yet. However, a process expresson including $\sum a_i\downarrow^{t_i}.P_i$ $(i \geq 2)$ can preserve (1) of the above propostion, only if input ports which has not recevied any messages are always possible, for example $\sum_{a_i \in M} a_i\downarrow^{t_i}.(P|\prod_{a_j \in M-\{a_i\}} a_j\downarrow^{t_j})$.

Definition 3.4 P_1 and P_2 are $[t_1, t_2]$ timed order, written $P_1 \preceq_{[t_1,t_2]} P_2$, if $(P_1, P_2) \in \mathcal{R}_{[t_1,t_2]}$ $(\subseteq \mathcal{P} \times \mathcal{T}) \times (\mathcal{P} \times \mathcal{T}))$ for some timed prebisimulation. That is,

$$\preceq_{[t_1,t_2]} \stackrel{\text{def}}{=} \bigcup\{\mathcal{R}_{[t_1,t_2]} \ : \ \mathcal{R}_{[t_1,t_2]} \quad \text{is } [t_1, t_2] \text{ timed prebisimulation. } \}$$

where we abbrave $\preceq_{[0,0]}$ as $\preceq$.

From Propostion 3.3, we can easily show the following basic properties.

Proposition 3.5

(1) $P \preceq_{[t,t]} P$ (2) If $P_1 \preceq_{[t_1,t_2]} P_2$ and $P_2 \preceq_{[t_2,t_3]} P_3$ then $P_1 \preceq_{[t_1,t_3]} P_3$

Let $P \preceq P$、$P_1 \preceq P_2$ and $P_2 \preceq P_3$ then $P_1 \preceq P_3$. Hence, $\preceq$ is a preorder relation.

Proposition 3.6 We assume $t_1' \leq t_1$, $t_2' \leq t_2$ and $t_1' \leq t_2'$:

$$P_1 \preceq_{[t_1,t_2]} P_2 \qquad \textit{then } P_1 \preceq_{[t_1',t_2']} P_2$$

This proposition shows that, if the first process with messages which can be sent t_1 time units before, is faster than the second one with messages which can be sent t_2 time units before, the first process with messages t_1' time units before is faster than the second process with messages after t_2' time units before.

We show a useful fact for proving substitutability between two behaviorally equivalent processes with different speeds.

Theorem 3.7

$$P_1 \preceq_{[t_1,t_2]} P_2 \quad \textit{and} \quad Q_1 \preceq_{[t_1,t_2]} Q_2 \qquad \textit{then} \qquad P_1|Q_1 \preceq_{[t_1,t_2]} P_2|Q_2$$

From Proposition 3.6 we have that if $P_1 \preceq P_2$ and $Q_1 \preceq Q_2$ then, $P_1|Q_1 \preceq P_2|Q_2$. We show some notable points on the above result.

(1) From the above result, if two processes (or two group of paralle processes) are behaviorally equivalent and the first process can perform its behaviors faster than the second process, we can gurantee that a system embadding the faster process the same (or slower) system embadding the slower process are behaviorally equivalent each other and the system embadding the faster process can peform faster than the system embadding the slower one. Since distributed computing is based on interactions among processes executing in parallel, the following substitutability for parallel composition provides a poweful method to analyze and verify the temporal and behaviroal prorperties of interactions among distributed processes.

(2) We do not always hold that if $t_1 < t_1'$, A$t_2 < t_2'$, if $P_1 \preceq_{[t_1,t_2]} P_2$, and $Q_1 \preceq_{[t_1,t_2]} Q_2$, then $P_1|Q_1 \preceq_{[t_1',t_2']} P_2|Q_2$. To show that a faster process can substitute for its slower one used as a component of a system, the faster process need to perform faster than the slower one, allowing for all messages can be sent after the beginning time of the system.

Proposition 3.8

$$P_1 \preceq_{[t_1+k_1,t_2+k_2]} P_2 \quad \textit{iff} \quad (k_1)P_1 \preceq_{[t_1,t_2]} (k_2)P_2$$

The delay operation means that supends the following execution for its specifed time units. $(k_1)P_1 \preceq_{[t_1,t_2]} (k_2)P_2$ corresponds that P_1 can perform faster than P_2, allowing for messages which can be sent $k_1 + t_1$ (or $k_2 + t_2$) time units before.

Proposition 3.9 Let $P_1 \preceq_{[t_1,t_2]} P_2$, then

(1) $k_1 + t_1 \leq k_2 + t_2$ then $a\uparrow^{k_1}.P_1 \preceq_{[t_1,t_2]} a\uparrow^{k_2}.P_2$
(2) $P_1 \setminus L \preceq_{[t_1,t_2]} P_2 \setminus L$

This shows that the order relation is preserved in message sending and restriction. However, $a\downarrow^{k_1}.P_1 \preceq_{[t_1,t_2]} a\downarrow^{k_2}.P_2)$ is not deduced from $P_1 \preceq_{[t_1,t_2]} P_2$, and thus the relation is a precongruence over $\mathcal{P}$ expressions. This is because $a\downarrow^{k_1}.P_1$ (or $a\downarrow^{k_2}.P_2$) can received messages t_1 (or t_2) before.

Remarks

In this paper, we dealed with only communication delay between an observer and ports of processers. However we can explicitly add communication delay between an observer and processes themselves by slightly changing Definition 3.2. For example, we first assume d_1 time units is communication delay between the observer and d_2 time units is communication delay between the observer and process P_2. (i) of Definition 3.2 is refomulated below: $\Pi_{a\in M} a\uparrow^{k_a^1+d_1} | P_1 \rightarrowtail^{m_1} \Pi_{b\in N} b\uparrow^{-l_b^1-d_1} | P_1'$ then $\Pi_{a\in M} a\uparrow^{k_a^2+d_2} | P_2 \rightarrowtail^{m_2} \Pi_{b\in N} b\uparrow^{-l_b^2-d_2} | P_2'$ and $l_b^2 \leq l_b^2$ and $(P_1', P_2') \in \mathcal{R}_{[t_1+m_1,t_2+m_2]}$. For (ii) we can define in the same way. Moreover, we have a notable fact that when the above relatin is written as $P_1 \preceq^{(d_1,d_2)}_{[t_1,t_2]}$, we have that if $P_1 \preceq^{(d_1,d_2)}_{[t_1,t_2]}$ then, $P_1 \preceq^{(d_1',d_2')}_{[t_1,t_2]}$ where $d_1' \geq d_1$ and $d_2' \geq d_2$.

4 Discussion

Related Work

This section presents an overview of related work. There have been a number of computing models for reasoning about distributed (real-time) systems, in particular communication delay and asynchronous communication. Most of existing works assume communication delay is inherently arbitary, i.e., from zero to forever. Thus, they can analyze causality between message sending and receiving by using time-stamps (for example see [**?**]) and partial order semantics (for example see [6]), but they lose real-time duration between actions.

Recently, a lot of researchers have explored real-time extensions of process algebras for synchournous communication, for example, see [12, 13, 15]. There is a notable work by Moller and Tofts in [12]. The authors studied a preorder relating timed processes with respect speed based on the bisimulation concept, like ours. Moreover, they pointed out the existance of anormaly in defining such a relation which is a precongruence. However, the order relation is seriously dependent on synchronous communication and it is not preserved in the parallel operator only when the number of parallel processes never change and thus it cannot apply to dynamic systems with process creation.

On the otherhand, there are some process algebras with the ability of expressing asynchronous communication [7, 9, 5]. However, there is a few framework to deal with both delay and asynchrony in communication. Among [3, 8], asynchronous communication in process algebras inheretly based on sysnchronous one can be modeled by introducing auxiliary meachnisms; buffering meachisms and auxiliary operators, often called *state operators*. Their extensions are always not suitable with computational aspects of process algebras. Also, in [2] the authors proposed a process algebra with the ability of expressing asynchronous communication with delay. It is based on an timed extended calculus of ACP [1]. Like ours, it represents asynchronous message transmission as a creation of a process which corresponds to the message in the way taken in [7, 9] and it represents communication delay by suspending the created process for the amount of the delay. Therefore, it is very similar to ours. However, the process algebra in [2] just provides just a language to describe systems with asynchronous communication with delay and failure, and it does not provide any verification for these systems.

Concluding Remarks

In this paper we proposed a formalism for the specification and verification of distributed real-time processes which interact with one another through asynchronous message passing with transmission delay. It is an extension of a process algebra with the ability of expressing communicaiton delay, asynchrnous communication.

Based on the formalism, we also developed a substitutionality for distributed processes with respect to their speed. It is formulated as a prebisimulation relation which can distinguish between behaviourally equivalent processes which peform the behaviours at different speed. In defining this relation, we needed to restrict the expressive power of the language. This fact shows that a faster process with timeout handling cannot always substituted for its slower one and reveals an impotant problem in constructing soft real-time systems.

Finally, we would like to point out some further issues. The study of relating asynchronously communicating processes has just started. We have many problems to be solved. As mentioned previously, there is an anomaly in defining a substitutive notion of such a speed relation. This anomaly is not essentially dependent on our formalism. We believe it is a general problem in defining relating processes with respect to speed. We plan to investigate this problem in various asynchronous interactions. Also, we are interested in a contrast to existing order relations by Moller and Tofts in [12]. Besides, one purpose of the formalism was to investigate a process algebra for reasioning asynchronous communication among remote processes without introducing the concept of process location. There are some important relationships between communicaton delay and process location. We are interested in developing a process algebra with the ability of experssing

communiation delay depending on distance between the sender and the receiver.

References

[1] Baeten, J. C. M, and Bergstra, J. A, *Process Algebra*, Cambridge University Press 1990.

[2] Baeten, J. C. M, and Bergstra, J. A, *Asynchronous Communication in Real Space Process Algebra*, Formal Techniques in Real-Time and Fault-Tolerant System, LNCS 591, p473-491, 1991.

[3] Bergstra, J. A, and Klop, J. W., *Process Algebra with Asynchronous Communication Machnisms*, Seminar on Concurrency, LNCS 197, p76-95, 1985.

[4] 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.

[5] de Bore, F.S., Klop, J.W., and Palamidessi, *Asynchronous Communication in Process Algebra*, Proceedings of LICS'92, p137-147, 1992.

[6] Degano, P, deNicola, R. D., and Montanari, U., *A Distributed Operational Semantics for CCS Based on Condition / Event Systems*, Acta Infomatica, Vol.26, p59-91, 1988.

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

[8] Jifeng, M. B, and Hoare, C. A. R., *A Theory of Synchrony and Asynchrony*, Proceedings of IFIP WG2.2/2.3 Programming Concepts and Methods, p459-478, 1990

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

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

[11] Milner, R., Parrow. J., Walker, D., *A Calculus of Mobile Processes*, Information and Computation, Vol.100, p1-77, 1992.

[12] Moller, F., and Tofts, C., *Relating Processes with Respect to Speed*, Proceedings of CONCUR'91, LNCS 527, Springer-Verlag, 1991.

[13] Nicollin. X., and Sifakis, J., *An Overview and Synthesis on Timed Process Algebras*, Proceedings of Computer Aided Verification, LNCS 575, p376-398, Springer-Verlag, 1991.

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

[15] Satoh, I., and Tokoro, M., *A Formalism for Real-Time Concurrent Object-Oriented Computing*, Proceedings of 7th ACM Object Oriented Programming Systems and Languages, and Applications, p315-326, 1992.

[16] Satoh, I., and Tokoro, M., *A Timed Calculus for Distributed Objects with Clocks*, Proceedings of 8th European Conference on Object Oriented Programming, LNCS 707, p326-345, Springer-Verlag, 1993.

[17] Satoh, I., and Tokoro, M., *A Process Calculus with Communication Delay and Locality*, to appear as Keio CS Technical Report.