Protocol Synthesis from Timed and Structured Specifications

Akio Nakata Teruo Higashino Kenichi Taniguchi
Dept. of Information and Computer Sciences, Osaka University
Toyonaka, Osaka 560, Japan

Abstract

In this paper, we propose a method to synthesize protocol specifications automatically from service specifications written in a time-extended LOTOS called LOTOS/T+. In LOTOS/T+, structured descriptions, such as parallelism and interruption are allowed to describe service specifications, and time-constraints among non-adjacent actions can be described using Presburger formulas. Here we assume that there is a reliable communication channel between any two nodes and the maximum communication delay for each channel is bounded by a constant. Moreover we assume service specifications have no deadlocks. Under our simulation policy, a specification S' is derived from a given service specification S and a given maximum communication delay of each channel. In S', time-constraints necessary for exchanging synchronization messages are added. If S and S' can carry out the same behaviour, i.e., if S and S' are bisimulation equivalent when time is ignored, then a correct protocol specification for simulating S is derived from S' automatically.

1 Introduction

For designing reliable distributed systems, *protocol synthesis* methods are useful [1]. In the recent years, several methods for synthesizing correct protocol specifications from given service specifications mechanically have been proposed for FSM, EFSM, LOTOS and Petri Net models [2, 3, 4]. However those proposals do not consider quantitative time constraints for the systems. It is highly desirable to synthesize protocol specifications from time-constrained service specifications. Recently, in [5], a method to derive protocol specifications from timed service specifications written in a FSM model has been proposed, but in such a FSM model we can not specify complicated order of actions in a structural way.

In this paper, we propose a method for synthesizing correct protocol specifications automatically from given service specifications written in a sub-class of LOTOS/T+ (which is a modified version of [6]), one of timed extensions of LOTOS [7]. LOTOS/T+ has an ability to specify complicated action ordering such as parallel composition and interruption. Moreover, in LOTOS/T+, time constraints among actions can be specified as formulas using addition, subtraction and inequalities on integers. In addition, using variables to hold the time when preceding actions are executed, we can specify time constraints for succeeding actions.

In our method, we assume that (a) each communication channel is error-free and its maximum propagation delay is bounded by a constant, and that (b) all nodes with their clocks can start their executions simultaneously and the clocks always synchronize each other. Under this assumption, we give a simulation policy for each node to execute actions in exactly the same order as specified in a given service specification. Basically, the simulation policy is based on the method which we have proposed in [3, 8]. That is, after executing each action, say a, a synchronization message is sent to the node which executes a succeeding action, say b, to inform that a has been executed. If the execution time of a is needed, the time is also transmitted. The action b must be executed after the message is received. We derive protocol specifications under the above policy. However, if we consider time-constraints, many problems arise. For example, if a service specification states "the action a must be executed before time 3 at node 1, and then the action b must be executed before time 5 at node 2," and if the maximum communication delay from node 1 to node 2 is 3 units of time, the synchronization message sent from node 1 after a is executed may not reach node 2 before time 5. To cope with this kind of problem, we restrict, for example, the time constraint of the action a to "before time 2" so that we can guarantee the synchronization message reaches node 2 in time. As another example, suppose that a service specification states "the action a must be executed between time 1 and 3 at node 1, and after that the action b must be executed between time 4 and 5 at node 2". If the maximum communication delay from node 1 to node 2 is 3 units of time, the same observation as the previous example holds, i.e., the synchronization message from node 1 to node 2 may not reach in time. But as for the above case, a different solution is possible. Since each node has its own clock and all clocks synchronize each other, the ordering of actions a and b is guaranteed without any message exchange. That is, the temporal ordering as the total system is guaranteed if each node decides the execution time of its action a (or b) using its own clock.

In our derivation method, first, from a given service specification S and a given maximum delay of each channel, we derive a specification S' where additional time constraints are appended to S so that the message exchanges are carried out in time. We make only the weakest timing restrictions to S so that each node can simulate S under the above policy. If

0-8186-7216-1/95 $04.00

Table 1: Syntax of LOTOS/T+

E ::=	**stop** (untimed deadlock)
	exit (successful termination)
	$a;E$ (action prefix, untimed)
	$a[P(t,\bar{x})];E$ (action prefix, timed)
	$E[]E$ (choice)
	$E\|\|\|E$ (asynchronous parallel)
	$E\|\|E$ (synchronous parallel)
	$E\|[A]\|E$ (generic parallel composition)
	$E[> E$ (disabling)
	$E >> E$ (enabling)
	hide A **in** E (hiding)
	asap A **in** E ("as soon as possible" execution)
	$P[g_1,\ldots,g_k](\bar{e})$ (process invocation)

S and S' can execute the same behaviour (note that the transformation from S to S' does not necessarily preserve the equivalence), i.e., if they are observationally equivalent (bisimulation equivalent[6]) when we consider sending/receiving actions of synchronization messages and an action *tick* representing one unit time progress as unobservable, then a protocol specification satisfying S is derived automatically from S'.

The paper is organized as follows. Section 2 describes our specification language LOTOS/T+. In Section 3 we explain the protocol synthesis method. Section 4 concludes this paper.

2 LOTOS/T+

The specification language we use for describing both service specifications and protocol specifications is LOTOS/T+, which is slightly modified one from LOTOS/T[6]. The syntax and informal semantics of LOTOS/T+ are described below.

Definition 1 Behaviour expressions of LOTOS/T+ is defined as Table 1 (the preference of each operator is the same as LOTOS[7]) , where $a \in Act \cup \{i\}$(Act stands for a finite set of all observable actions, and i represents an internal (unobservable) action), $A \subseteq Act$, $k \in \mathbf{N}$($\mathbf{N}$ is a set of natural numbers), and $P(t,\bar{x})$ stands for a Presburger formula[9], that is, a first order logic formula whose atoms are integer linear inequalities, which has a free variable t and other free variables x_i. Here $\bar{x} \stackrel{\text{def}}{=} (x_1, x_2, \ldots, x_k)$ for some k. Intuitively, t represents the current time, e stands for an integer linear expression (ILE for short) and $\bar{e} \stackrel{\text{def}}{=} (e_1, e_2, \ldots, e_k)$ for some k, where each e_i is an ILE. □

In LOTOS/T+, time constraints of actions are described in a subclass of Presburger formulas, more specifically, logical combinations of the atoms each of which takes the form of either $e_l \leq t$, $t \leq e_u$ or $x = t$. Here, e_l (e_u) is an ILE representing the lower bound (upper bound , respectively) of the time an action is executable. The atomic formula $x = t$ means that the action's executed time is assigned to the variable x. For simplicity, we use an abbreviation $e_l \leq t \leq e_u$ for $e_l \leq t \wedge t \leq e_u$. Other symbols of inequality such as $<$,$\geq$,etc. may also be used. In our semantics, an upper bound e_u specified as a time constraint of an action means the action *must* be executed no later than e_u. In this case, we say that *urgency* of the action at time e_u is specified. Note that in our language, executability and urgency of each action at each given time t are decidable[6].

Example 1

$$B = a[2 \leq t \leq 3 \wedge x_0 = t]; b[t = x_0 + 3]; c[t = x_0 + 4]; \mathbf{stop}$$

The behaviour expression B represents the following behaviour. The action a must be executed between time 2 and 3, and the execution time of a is assigned to the variable x_0. Then b must be executed exactly 3 units of time after the execution of a. And then c must be executed exactly 4 units of time after a. □

Example 2

1. $E = a[x = t]; b; c[t \geq x + 2]; \mathbf{stop}$
2. $P = a[t = 5]; \mathbf{stop}[]b[t = 1]; P$

The first behaviour expression is an example that an action without time constraints is inserted between time constrained actions. b in the first example can be executed at any time after a is executed, that is, we consider that a formula "true" is omitted as a time constraint of b. Moreover, an unbounded interval "$t \geq x + 2$" is specified as a time constraint of c.

The second one is an example of recursive processes. A clock is reset to 0 at each moment P is invoked. Generally, each instance of processes has its own clock locally, which is reset to 0 at the beginning of the process's run. If a process P has a process parameter like $P(t)$, however, the clock is not reset to 0 but to the actual time t_0, that is, $P(t_0)$ is invoked. The corresponding LTS's are shown in Fig. 1. □

The difference between LOTOS/T+ and LOTOS/T is an interpretation of the behaviour of internal actions. In the method we propose, the delay of internal messages exchanged among nodes is assumed to be uncertain. On the other hand, in LOTOS/T internal actions are defined to be executed as soon as possible after it is enabled, so we cannot describe uncertain delay of internal actions in LOTOS/T. Thus, we define LOTOS/T+ so that the executable time of internal actions may be decided nondeterministically in the range of time constraints. To describe this property, we define a construct "**asap** A in B," representing the same behaviour B except the actions in A must be executed as soon as possible they are enabled.

The formal definition of the semantics is given as the inference rules in Fig. 2. From the rules, we can automatically decide whether an action $a[P(t,\bar{x})]$ is executable, if satisfiability of the corresponding predicates $P(0,\bar{x})$ and $\exists t' \exists \bar{x}[t' > 0 \wedge P(t',\bar{x})]$ is decidable. $P(0,\bar{x})$ denotes whether a is executable at the current time and $\exists t' \exists \bar{x}[t' > 0 \wedge P(t',\bar{x})]$ denotes whether a

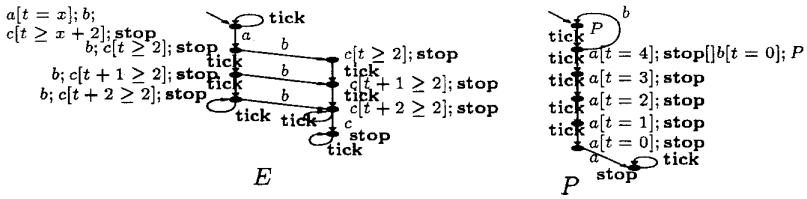

Figure 1: The semantics of E and P

$$\frac{}{\mathbf{stop} \xrightarrow{\mathbf{tick}} \mathbf{stop}} \text{(S1)} \qquad \frac{}{\mathbf{exit} \xrightarrow{\delta} \mathbf{stop}} \text{(E1)} \qquad \frac{}{\mathbf{exit} \xrightarrow{\mathbf{tick}} \mathbf{exit}} \text{(E2)}$$

$$\frac{P(0,\bar{c})}{a[P(t,\bar{x})];B \xrightarrow{a} [\bar{c}/\bar{x}]B} \text{(TAP1)} \qquad \frac{\exists t' \exists x[t' > 0 \wedge P(t',x)]}{a[P(t,\bar{x})];B \xrightarrow{\mathbf{tick}} a[P(t+1,\bar{x})];[t+1/t]B} \text{(TAP2)} \qquad \frac{}{a;B \xrightarrow{a} B} \text{(UAP1)}$$

$$\frac{}{a;B \xrightarrow{\mathbf{tick}} a;[t+1/t]B} \text{(UAP2)} \qquad \frac{B_1 \xrightarrow{\beta} B_1'}{B_1[]B_2 \xrightarrow{\beta} B_1'} \text{ iff } \beta \in Act \cup \{\delta, i\} \text{ (CH1)} \qquad \frac{B_2 \xrightarrow{\beta} B_2'}{B_1[]B_2 \xrightarrow{\beta} B_2'} \text{ iff } \beta \in Act \cup \{\delta, i\} \text{ (CH2)}$$

$$\frac{B_1 \xrightarrow{\mathbf{tick}} B_1' \quad B_2 \xrightarrow{\mathbf{tick}} B_2'}{B_1[]B_2 \xrightarrow{\mathbf{tick}} B_1'[]B_2'} \text{(CH3)} \qquad \frac{B_1 \xrightarrow{\mathbf{tick}} B_1' \quad B_2 \not\xrightarrow{\mathbf{tick}}}{B_1[]B_2 \xrightarrow{\mathbf{tick}} B_1'} \text{(CH4)} \qquad \frac{B_2 \xrightarrow{\mathbf{tick}} B_2' \quad B_1 \not\xrightarrow{\mathbf{tick}}}{B_1[]B_2 \xrightarrow{\mathbf{tick}} B_2'} \text{(CH5)}$$

$$\frac{B_1 \xrightarrow{\beta} B_1' \quad B_2 \xrightarrow{\beta} B_2'}{B_1|[A]|B_2 \xrightarrow{\beta} B_1'|[A]|B_2'} \text{ iff } \beta \in A \cup \{\delta\} \text{ (PA1)} \qquad \frac{B_1 \xrightarrow{\mathbf{tick}} B_1' \quad B_2 \xrightarrow{\mathbf{tick}} B_2'}{B_1|[A]|B_2 \xrightarrow{\mathbf{tick}} B_1'|[A]|B_2'} \text{(PA2)} \qquad \frac{B_1 \xrightarrow{a} B_1'}{B_1|[A]|B_2 \xrightarrow{a} B_1'|[A]|B_2} \text{ iff } a \notin A \vee a = i \text{ (PA3)}$$

$$\frac{B_2 \xrightarrow{a} B_2'}{B_1|[A]|B_2 \xrightarrow{a} B_1|[A]|B_2'} \text{ iff } a \notin A \vee a = i \text{ (PA4)} \qquad \frac{B_1|[\emptyset]|B_2 \xrightarrow{\alpha} B'}{B_1|||B_2 \xrightarrow{\alpha} B'} \text{(PA5)} \qquad \frac{B_1|[Act]|B_2 \xrightarrow{\alpha} B'}{B_1||B_2 \xrightarrow{\alpha} B'} \text{(PA6)}$$

$$\frac{B_1 \xrightarrow{a} B_1'}{B_1[> B_2 \xrightarrow{a} B_1'[> B_2} \text{(DI1)} \qquad \frac{B_2 \xrightarrow{\beta} B_2'}{B_1[> B_2 \xrightarrow{\beta} B_2'} \text{ iff } \beta \in Act \cup \{\delta, i\} \text{ (DI2)} \qquad \frac{B_1 \xrightarrow{\delta} B_1'}{B_1[> B_2 \xrightarrow{\delta} B_1'} \text{(DI3)}$$

$$\frac{B_1 \xrightarrow{\mathbf{tick}} B_1' \quad B_2 \xrightarrow{\mathbf{tick}} B_2'}{B_1[> B_2 \xrightarrow{\mathbf{tick}} B_1'[> B_2'} \text{(DI4)} \qquad \frac{B_1 \xrightarrow{a} B_1'}{B_1 >> B_2 \xrightarrow{a} B_1' >> B_2} \text{(EN1)} \qquad \frac{B_1 \xrightarrow{\delta} B_1'}{B_1 >> B_2 \xrightarrow{i} B_2} \text{(EN2)}$$

$$\frac{B_1 \xrightarrow{\mathbf{tick}} B_1' \quad B_2 \xrightarrow{\mathbf{tick}} B_2' \quad B_1 \not\xrightarrow{\delta}}{B_1 >> B_2 \xrightarrow{\mathbf{tick}} B_1' >> B_2'} \text{(EN3)} \qquad \frac{B \xrightarrow{\beta} B'}{\mathbf{hide}\ A\ \mathbf{in}\ B \xrightarrow{\beta} \mathbf{hide}\ A\ \mathbf{in}\ B'} \text{ iff } \beta \in (Act \setminus A) \cup \{\delta, i\} \text{ (HI1)}$$

$$\frac{B \xrightarrow{a} B'}{\mathbf{hide}\ A\ \mathbf{in}\ B \xrightarrow{i} \mathbf{hide}\ A\ \mathbf{in}\ B'} \text{ iff } a \in A \text{ (HI2)} \qquad \frac{B \xrightarrow{\mathbf{tick}} B'}{\mathbf{hide}\ A\ \mathbf{in}\ B \xrightarrow{\mathbf{tick}} \mathbf{hide}\ A\ \mathbf{in}\ B'} \text{(HI3)} \qquad \frac{B \xrightarrow{a} B'}{\mathbf{asap}\ A\ \mathbf{in}\ B \xrightarrow{a} \mathbf{asap}\ A\ \mathbf{in}\ B'} \text{(ASAP1)}$$

$$\frac{B \xrightarrow{\mathbf{tick}} B' \quad B \not\xrightarrow{a} \text{ for all } a \in A}{\mathbf{asap}\ A\ \mathbf{in}\ B \xrightarrow{\mathbf{tick}} \mathbf{asap}\ A\ \mathbf{in}\ B'} \text{(ASAP2)} \qquad \frac{[\bar{e}/\bar{x}]B\{g_1'/g_1,\ldots,g_k'/g_k\} \xrightarrow{\alpha} B'}{P[g_1',\ldots,g_k'](\bar{e}) \xrightarrow{\alpha} B'} \text{ iff } P[g_1,\ldots,g_k](\bar{x}) := B \text{ is a definition} \text{ (PR1)}$$

Figure 2: The operational semantics of LOTOS/T+

may be executable in the future. Since $P(t,\bar{x})$, a time constraint of the action, is a Presburger formula, all the predicates above are also Presburger formulas, so their satisfiability is decidable[9, 6]. Hence, we can construct mechanically the corresponding LTS's (possibly, of infinite state spaces) from given behaviour expressions. Here we give some rules to show how to construct LTS's. Firstly, for the process E in Example 2 (see Fig.1):

- $E = a[t = x];b;c[t \geq x+2];\mathbf{stop} \xrightarrow{a} b;c[t \geq 2];\mathbf{stop}$ [from the rule (TAP1)],
- $b;c[t \geq 2];\mathbf{stop} \xrightarrow{\mathbf{tick}} b;c[(t+1) \geq 2];\mathbf{stop}$ [from the rule (TAP2)],

Secondly, for the process P in Example 2 (see Fig.1):

- $P \xrightarrow{\mathbf{tick}} a[t = 4];\mathbf{stop}[]b[t = 0];P$ [from the rules (PR1), (CH3), (TAP2)],

- $a[t=4]$;**stop**[]$b[t=0]$;$P \xrightarrow{\textbf{tick}} a[t=3]$;**stop** [from the rules (CH4), (TAP2)],

3 Protocol Synthesis

3.1 Protocol Synthesis Problem

In this section, we define a protocol synthesis problem from timed service specifications. First we introduce some notations. Let place(a) denote a node assignment for the action a. In the rest of this paper, we assume that a^k stands for an action a with place(a) = k. Moreover, we use some notations $SP(B)$, $EP(B)$, $AP(B)$, whose intuitive meanings are the sets of the starting nodes of B, the ending nodes of B, all the participating nodes in B, respectively. For example, if $B = a^1; b^2;$ **exit**$|||e^3; d^2;$ **exit**, then $SP(B) = \{1,3\}$, $EP(B) = \{2\}$ and $AP(B) = \{1,2,3\}$. We can derive them from B and place() mechanically. The formal definitions of these notations appeared in [3].

[Protocol Synthesis Problem]

Assumptions: 1. there exists a reliable(error-free), asynchronous, full-duplex communication channel between every two nodes.
2. there's no limitations on contents of messages exchanged among nodes.
3. all nodes have their own clocks and they always synchronize each other.

Inputs:
- A service specification S.
- A node assignment place(a) for each action a.
- An upper bound of delay $d_{ij\max}$ for each channel from node i to j, such that $d_{ii\max} = 0$ and $\forall k\ d_{ij\max} \leq d_{ik\max} + d_{kj\max}$.

Here, we give the following restrictions for simplifying the derivation.

Restriction 1. S does not contain any deadlock states. And S does not contain the synchronous parallel composition (rendezvous).
Restriction 2. If S contains $B_1 [> B_2$ as a subexpression, B_1 must be a finite process, and there exists a constant t_0 such that B_1 can execute no action after time t_0 and B_2 can execute any action only after time t_0.
Restriction 3. If S contains $B_1 >> B_2$ as a subexpression, B_1 must be a finite process.
Restriction 4. Every process invocation in S must not have any process parameters, i.e. the behaviour of each invoked process does not depend on the previous behaviour.
Restriction 5. The context of each process invocation P must be either $a;P$ or $a[P(t,\bar{x})];P$, so that just one action precedes P.
Restriction 6. For every subexpression $B_1 [] B_2$ of S, there exists a node p such that $SP(B_1) = SP(B_2) = \{p\}$, and $EP(B_1) = EP(B_2)$[3].
Restriction 7. For every subexpression $B_1 [> B_2$ of S, $EP(B_1) = EP(B_2)$[3].

Outputs: Protocol entity specifications $Node_1$, $Node_2$, ..., $Node_n$ for all nodes, which are *correct* in the following meaning:

Let I be the composite system which connects $Node_1$, $Node_2$, ..., $Node_n$ together with a communication medium which has channels from node i to j with maximum delay of $d_{ij\max}$. Intuitively, $\{Node_i\}_{i=1,2,...,n}$ are correct w.r.t. S when S can strictly simulate I including timing properties, whereas I can simulate S if time is ignored. In this case, a set of executable time of each action in I is a nonempty subset of that of the corresponding action in S.
Formally, the correctness is defined as follows. Let

$$I = \textbf{hide } G \text{ in } (\textbf{asap } G_s \text{ in } ((Node_1|||Node_2|||\ldots|||Node_n)|[G]|Medium)),$$

where G is a set of all sending/receiving actions of synchronization messages $\{s_{ij}(m), r_{ij}(m) \mid i,j \in \{1,2,\ldots,n\},\ m \in M\}$ and G_s is a set of all sending actions of synchronization messages $\{s_{ij}(m) | i,j \in \{1,2,\ldots,n\},\ m \in M\}$, and $Medium$ is a specification of the communication medium defined as follows:

$$Medium = |||_{i,j\in\{1,2,\ldots,n\}} Channel_{ij}$$
$$Channel_{ij} = |||_{m\in M}(s_{ij}(m)[x=t]; r_{ij}(m)[x \leq t \leq x + d_{ij\max}]; Channel_{ij})$$

Note that under the asynchronous communication medium, the sending actions are executed as soon as possible they are enabled, because they are spontanous. In contrast, the receiving actions are not spontanous, so they are not executed as soon as possible.
Before defining the correctness, we need some preliminary definitions.

Definition 2 Relations $\overset{\alpha}{\Longrightarrow}_t, \xrightarrow{\alpha}_u, \overset{\alpha}{\Longrightarrow}_u$ are defined as follows:

$$B \overset{\alpha}{\Longrightarrow}_t B' \overset{\text{def}}{=} \begin{cases} B(\xrightarrow{i})^* \xrightarrow{\alpha} (\xrightarrow{i})^* B', & \alpha \in Act \cup \{\delta, \textbf{tick}\} \\ B(\xrightarrow{i})^* B', & \alpha = \epsilon \end{cases}$$

$$B \xrightarrow{\alpha}_u B' \overset{\text{def}}{=} \begin{cases} B(\xrightarrow{\textbf{tick}})^* \xrightarrow{\alpha} (\xrightarrow{\textbf{tick}})^* B', & \alpha \in Act \cup \{\delta, i\} \\ B(\xrightarrow{\textbf{tick}})^* B', & \alpha = \epsilon \end{cases}$$

$$B \overset{\alpha}{\Longrightarrow}_u B' \overset{\text{def}}{=} \begin{cases} B(\xrightarrow{i}_u)^* \xrightarrow{\alpha}_u (\xrightarrow{i}_u)^* B', & \alpha \in Act \cup \{\delta\} \\ B(\xrightarrow{i}_u)^* B', & \alpha = \epsilon \end{cases}$$

□

Definition 3 A binary relation $\sqsubseteq_t$ on behaviour expressions is defined as a maximum one of relations $\mathcal{R}$ satisfying the following condition:

- If $I \mathcal{R} S$, then for all $\alpha \in Act \cup \{\delta, \epsilon\}$, all of the following conditions hold:
  1. If $I \overset{\alpha}{\Longrightarrow}_t I'$, then there exists some S' s.t. $S \overset{\alpha}{\Longrightarrow}_t S'$ and $I' \mathcal{R} S'$.
  2. If $I \overset{\textbf{tick}}{\Longrightarrow}_t I'$, then there exists some S' s.t. $S \overset{\textbf{tick}}{\Longrightarrow}_t S'$ and $I' \mathcal{R} S'$.
  3. If $S \overset{\alpha}{\Longrightarrow}_u S'$, then there exists some I' s.t. $I \overset{\alpha}{\Longrightarrow}_u I'$ and $I' \mathcal{R} S'$. □

Here we define the correctness.

Definition 4 We call a derived protocol specification $\{Node_i\}_{i=1,2,...,n}$ as $\sqsubseteq_t$-*correct w.r.t.* S if the following relation holds:

hide G **in** (**asap** G_s **in** (
$(Node_1|||Node_2|||\ldots|||Node_n)|[G]|Medium)) \sqsubseteq_t S$

□

3.2 Synthesis Method

Now we describe our method for synthesizing protocol specifications from timed service specifications.

Basically, we follow a similar idea to our previous work[3, 8]. Thus, after each node executed an action, it sends messages to the nodes which execute the succeeding actions, informing them that it has finished. We refer this kind of messages as *synchronization messages*. To handle time constraints between actions on different nodes, we naturally assume that synchronization messages may also contain, if needed, information about the time at which preceding actions were executed. One major problem is that the communication delay may make it impossible to execute an action in time. In general, all realistic communication media have propagation delay, and we cannot neglect uncertainty of such a delay in most cases. To overcome this problem, we propose the following method. First, for a given service specification S, we decide where to insert actions sending or receiving synchronization messages to simulate S, according to the policy similar to [3, 8]. Then we restrict time-constraints of some actions in S in order to guarantee the execution of succeeding actions are possible at the worst case of communication delay, keeping the restriction to a minimum. We represent the obtained specification as $Restr(S)$. Finally, from the restricted specification $S' = Restr(S)$, we derive protocol entity specifications for all nodes. If S and S' are equivalent[6], the derived protocol specifications are guaranteed correct w.r.t. S.

In the following subsections, we describe how the simulation of the service specification S is done, and how we can define the transformation $Restr()$, for each construct of LOTOS/T+.

3.2.1 Action Prefix

We can simulate Action Prefix $a^p[P(t,\bar{x})];B$ by sending a synchronization message from node p to all the nodes in $SP(B)$.

If time constraints are specified by assignment and referrence of the variables, nodes at which such variables are assigned to some values must propagate the values to the succeeding nodes.

Example 3

$$\begin{aligned} S &= a^1[x=t];b^2[t \le x+5 \wedge y=t]; \\ &\quad c^3[t \le x+7 \wedge t \le y+5];\textbf{exit} \\ &\quad d_{12}\max = d_{13}\max = d_{23}\max = 2 \\ Node_1 &= a[x=t];s_{12}(m,x);\textbf{exit} \\ Node_2 &= r_{12}(m,x);b[t \le x+5 \wedge y=t]; \\ &\quad s_{23}(m',x,y);\textbf{exit} \\ Node_3 &= r_{23}(m',x,y);c[t \le x+5 \wedge t \le y+5];\textbf{exit} \end{aligned}$$

□

Here we can remove some redundancies in inserting synchronization messages when time is considered. Specifically, if there's no executable time of a succeeding action that is earlier than or equal to some executable time of the preceding action, and there's no values to propagate to succeeding nodes, the synchronization message at this place is of no need to guarantee actions' order, i.e., time implicitly guarantees the order (recall Assumption 3 in Section 3.1). For example, let $S = a^1[P(t,\bar{x})];\ b^2[Q(t,\bar{y})];$ **exit** and suppose $\exists t,t',\bar{x},\bar{y}[P(t,\bar{x}) \wedge\ Q(t',\bar{y}) \wedge\ t' \le t]$ is unsatisfiable. Then from the time constraints, a is always executed before b, so even if we simply execute a and b at different places, the order is still preserved. Therefore, we can remove the synchronization message from node 1 to node 2 in this case.

Example 4 If the input is the following:

$$S = a^1[1 \le t \le 3];b^2[4 \le t \le 5];\textbf{exit}$$
$$d_{12}\max = 4,$$

we will simply derive:

$$\begin{aligned} Node_1 &= a[1 \le t \le 3];\textbf{exit} \\ Node_2 &= b[4 \le t \le 5];\textbf{exit} \end{aligned}$$

because $[(1 \le t \le 3) \wedge (4 \le t' \le 5) \wedge (t' \le t)]$ is unsatisfiable. □

From now, we consider the case where communication delay affects the simulation. For action prefix $a^p[P(t,\bar{x})];B$, we will derive a specification $Restr(S)$ whose time constraint of a^p is restricted so that there exists a time to execute the succeeding actions in B no matter how late the messages from the node p reach the nodes in $SP(B)$. Because we describe time constraints in Presburger formulas, we can easily restrict time constraints by logical conjunction.

Example 5 If the input is:

$$\begin{aligned} S &= a^1[1 \le t \le 3];b^2[4 \le t \le 7]; \\ &\quad c^3[5 \le t \le 10];d^2[6 \le t \le 12];\textbf{exit} \\ &\quad d_{12}\max = 4\ ,\ d_{23}\max = 4,\ d_{32}\max = 3, \end{aligned}$$

we restrict the time constraint of each action as follows:

d^2: $6 \le t \le 12$ (unmodified)
c^3: $5 \le t \le 10 \wedge\ \exists t'(t' \ge t + d_{32}\max \wedge 6 \le t' \le 12)$
$(\equiv 5 \le t \le 9)$
b^2: $4 \le t \le 7 \wedge\ \exists t'(t' \ge t + d_{23}\max \wedge\ 5 \le t' \le 9\ )$
$(\equiv 4 \le t \le 5)$
a^1: $2 \le t \le 4$ (unmodified (by Example 4))

So the derived protocol entity specification will be the followings:

$$\begin{aligned} Node_1 &= a^1[1 \le t \le 3];\textbf{exit} \\ Node_2 &= b^2[4 \le t \le \underline{5}];s_{23}(m1); \\ &\quad r_{32}(m2);d^2[6 \le t \le 12];\textbf{exit} \\ Node_3 &= r_{23}(m1);c^3[5 \le t \le \underline{9}];s_{32}(m2);\textbf{exit} \end{aligned}$$

□

Now we can define $Restr(S)$ formally as follows.

Definition 5 If $S = a[Q(t,x)];B$, then $Restr(S)$ is defined inductively as follows:

$$Restr(S) \stackrel{\text{def}}{=} Restr(S, \emptyset)$$

$$Restr(S,V) \stackrel{\text{def}}{=} \begin{cases} S & \text{if } B = \textbf{exit}, B = \textbf{stop} \\ & \text{or } B = P \text{ (Process invocation)}, \\ a[Q(t,\bar{x}) \wedge Q'(t)]; Restr(B, V \cup \bar{x}) & \text{otherwise.} \end{cases}$$

where, if $\{b_k[Q_k(t,y_k)] \mid k \in K\}$ is the set of starting actions of $Restr(B, V \cup \bar{x})$ with their time constraints,

$$Q'(t) \stackrel{\text{def}}{=} \begin{cases} \bigwedge_{k \in K}\{\exists t' \exists y_k[t' \geq t + d_{\text{place}(a),\text{place}(b_k)\text{max}} \\ \quad \wedge Q_k(t',y_k)]\} \\ \quad \text{if for some } k \in K \text{ s.t. } Q(t,\bar{x}) \wedge \\ \quad Q_k(t',y_k) \wedge t' \leq t \text{ is satisfiable,} \\ \quad \text{or any of the variables in } V \cup \bar{x} \text{ are} \\ \quad \text{referenced in } B, \\ true \qquad \text{otherwise.} \end{cases}$$

□

To summarize this section, our derivation takes 3 steps:

Step 1 determine at what position the synchronization messages are needed.
Step 2 according to the results of Step 1 and $d_{ij\text{max}}$, construct $Restr(S)$.
Step 3 decompose $Restr(S)$ into each node by the similar method to [3, 8], already described above.

3.2.2 Choice

To simulate choice expressions, we must solve the problem about distributed choice and empty alternatives[3]. A choice expression $B_1[]B_2$ is called *distributed choice* if the starting actions of B_1 and B_2 may be executed at different nodes. And we say that a node p has an *empty alternative* w.r.t. $B_1[]B_2$ if some actions in B_i may be executed at node p, whereas no actions in $B_{(i\text{mod}2)+1}$ are executed at node p. Distributed choice may cause simultaneous execution of the starting actions of both B_1 and B_2. Empty alternatives on node p may cause unconditional execution of B_i even if $B_{(i\text{mod}2)+1}$ is chosen. As for distributed choice, we avoid it by putting the same restriction (Restriction 6) as [3]. We have proposed a method for solving the empty alternative problem for the untimed case in [3]. But, here, we will use a slightly modified method. Unlike [3], the node where choice was made should immediately sends messages to the nodes that have empty alternatives in order not to violate time constraints of succeeding processes. Moreover, to make sure each B_i would not terminate before the messages sent to the nodes which has empty alternatives reach the destinations, the ending nodes of the chosen expression will receive acknowledgments from the nodes with empty alternatives before executing the ending actions (note that if the starting action of B_i coincides the ending action of it, i.e., the maximum length of B_i's action sequences is 1, this simulation method may not be applicable). Furthermore, to simulate a choice expression $B_1[]B_2$ in the above way successfully, we must not remove redundant synchronization messages in both B_1 and B_2, discussed in Section 3.2.1 (Example 4), otherwise the intermediate actions of each B_i may be executed independently, no matter which alternative is chosen.

To make it possible to simulate choice in the way above, we must guarantee that all the messages reach the destinations in time by restricting the time constraints of some actions. For a choice expression $B_1[]B_2$, if the messages sent from the node choice was made wouldn't have reached the destinations, or the acknowledgments wouldn't return, before the chosen behaviour B_i have been done, extra time would be spent waiting for the messages. So we will restrict the time constraints of the starting actions of B_1 and B_2 so that the messages can reach in time.

Example 6 Consider the following input:

$$S = a^1[2 \leq t \leq 5 \wedge x = t]; b^2[t \leq x+3]; c^3[t \leq x+6]; \textbf{exit} \\ []d^1[3 \leq t \leq 9]; e^3[t \leq 10]; \textbf{exit}$$

$$d_{12\text{max}} = 2,\ d_{23\text{max}} = 4,\ d_{13\text{max}} = 3$$

We must restrict the time constraint of d^1 in order to make the message from node 1 to 2 and the acknowledgment from node 2 to 3 reach by time 10.

$$Restr(S) = a^1[2 \leq t \leq 5 \wedge x = t]; b^2[t \leq x+2]; \\ c^3[t \leq x+6]; \textbf{exit} \\ []d^1[\underline{3 \leq t \leq 4}]; e^3[t \leq 10]; \textbf{exit}$$

Then, the specification of each node will be derived as follows:

$$Node_1 = a^1[2 \leq t \leq 5 \wedge x = t]; s_{12}(m1,x); s_{13}(m3,x); \\ \textbf{exit} \\ []d[3 \leq t \leq 4]; (s_{13}(m2); \textbf{exit} ||| s_{12}(m4); \textbf{exit})$$

$$Node_2 = r_{12}(m1,x); b^2[t \leq x+2]; s_{23}(m5); \textbf{exit} \\ []r_{12}(m4); s_{23}(m4); \textbf{exit}$$

$$Node_3 = r_{13}(m3,x); r_{23}(m4); c^3[t \leq x+6]; \textbf{exit} \\ [](r_{13}(m2) ||| r_{23}(m4)) >> e^3[t \leq 10]; \textbf{exit}$$ □

For defining $Restr(S)$, we need the auxiliary function $Restr'(S)$, which is the same as $Restr(S)$ except that no removal of redundant messages is considered. The formal definition of $Restr'(S)$ appears in [10].

Now we can define $Restr(S)$ as follows:

Definition 6 If $S = B_1[]B_2$, then $Restr(S)$ is defined inductively as follows:

$$Restr(S) \stackrel{\text{def}}{=} Restr'(f(B_1))[]Restr'(f(B_2))$$

where, we assume that $\{b_k[Q_k(t,y_k)] \mid k \in K\}$ is the set of the starting actions of B_i with their time constraints, and that $f(B_i)$ is an expression B_i whose time constraint of each starting action $Q_k(t,y_k)$ is replaced

with $Q_k(t, y_k) \wedge R'_k(t)$. Here $R'_k(t)$ is a Presburger formula defined as follows.

$$R'_k(t) \stackrel{\text{def}}{=} \bigwedge_{\substack{q \in AP(B_i) \setminus AP(B_{(i \bmod 2)+1}) \\ l \in L, r \in EP(B_i)}} \{\exists t' \exists z_l [t' \geq t + d_{pq}\max + d_{qr}\max \wedge R_l(t', z_l)]\}$$

where $\{R_l(t, z_l) | l \in L\}$ denotes the time constraints of $EP(B_i)$ and $SP(B_i) = \{p\}$. □

3.2.3 Asynchronous Parallel

For any asynchronous parallel expression $B_1 ||| B_2$, B_1 and B_2 are executed independently. So any synchronization messages are necessary between B_1 and B_2. Thus, $Restr(S)$ is defined as follows:

Definition 7 If $S = B_1 ||| B_2$, then $Restr(S) \stackrel{\text{def}}{=} Restr(B_1) ||| Restr(B_2)$ □

3.2.4 Enabling

For enabling expression $B_1 >> B_2$, we can apply essentially the same idea as action prefix. Due to the lack of space, we omit the details about $Restr()$ transformation for the enabling expressions. The details can be found in [10].

3.2.5 Disabling

For each disabling expression $B_1 [> B_2$, we make a strong restriction, Restriction 2, for simplicity. That is, for some t_0, all actions in B_1 are not executable after time t_0, and all actions in B_2 are executable only after time t_0. From Restriction 2, there is no cases that actions in B_1 and B_2 are simultaneously enabled at different nodes. So $B_1 [> B_2$ can be simulated by inserting messages to notify successful termination of B_1 to all nodes.

To make this simulation method work,the messages notifying B_1's termination have to reach before t_0.

Example 7 The input described below satisfies Restriction 2 ($t_0 = 11$) and Restriction 7:

$S = a^1[1 \leq t \leq 4]; b^2[3 \leq t \leq 8]; c^3[7 \leq t \leq 10]; \mathbf{exit}$
$[> d^3[12 \leq t]; \mathbf{exit}$
$d_{12}\max = 3, d_{23}\max = 4, d_{31}\max = 3, d_{32}\max = 4,$
$d_{ij}\max = 2$ for other i,j.

In order to guarantee that the notification of successful termination sent from node 3 to nodes 1 and 2 can reach before time $t_0 = 11$, the time constraint of c^3 must be restricted to $7 \leq t \leq 7$, because the notification from node 3 to node 2 may take $d_{32}\max = 4$ units of time. Then, the restriction discussed in the previous section is applied for b^2 and a^1.

$Restr(S) = a^1[\underline{1 \leq t \leq 1}]; b^2[\underline{3 \leq t \leq 4}];$
$c^3[\underline{7 \leq t \leq 7}]; \mathbf{exit}[> d^3[12 \leq t]; \mathbf{exit}$

From this, the protocol entity specification of each node will be derived as below:

$Node_1 = a^1[1 \leq t \leq 1]; r_{31}(m1); \mathbf{exit}[> i[t = 12]; \mathbf{exit}$
$Node_2 = b^2[3 \leq t \leq 4]; r_{32}(m1); \mathbf{exit}[> i[t = 12]; \mathbf{exit}$
$Node_3 = c^3[7 \leq t \leq 7]; (s_{31}(m1) ||| s_{32}(m1)) >> \mathbf{exit}$
$[> d^3[12 \leq t]; \mathbf{exit}$ □

The definition of $Restr(S)$ is as follows:

Definition 8 If $S = B_1 [> B_2$,

$$Restr(S) \stackrel{\text{def}}{=} Restr(g'(B_1)) [> Restr(B_2),$$

where $g'(B_1)$ represents a transformation replacing the time constraint $P(t, \bar{x})$ of each ending action of B_1 with $P(t, \bar{x}) \wedge P'(t)$. Here

$$P'(t) \stackrel{\text{def}}{=} \bigwedge_{p \in EP(B_1), q \in ALL} \{t + d_{pq}\max \leq t_0\}.$$ □

3.2.6 Process Invocation

In our specification language, time is reset to 0 at every moment processes are invoked, avoiding accumulation of time constraints. To simulate this in distributed environments, we make all nodes to pretend as if they invoke a process simultaneously. In order to do so,

1. Fix one node for a *responsible node*, which decides the time to invoke a process (the time just before invoking a process). In this paper, from Restriction 5, the context of each process invocation must be the form of $a; B$ or $a[P(t, x)]; B$. So we fix the node place(a) as the responsible node w.r.t. the process P.
2. The responsible node notifies the invocation time of the process to all nodes, and immediately invokes the process locally.
3. The other nodes except the responsible node receive the notification, and invoke the process whose time constraints are modified to make the invocation time be virtually equal to that of the responsible node. Recall that the actual local time is reset to 0 just after the process invocation of each node.

To implement 3., we modify each process P without parameters in service specifications to $P(e_P)$ with just one parameter e_P in protocol specifications (Restriction 4), and replace every occurrence of t in the right hand of the process definition of P with $t + e_P$. The parameter e_P represents the difference between the actual invocation time and virtual invocation time. For example, $P(3)$ means the process P with replacing its time constraint, for instance, $t \leq 5$, with $t + 3 \leq 5$. Corresponding to each process invocation of P in the service specification, we derive a protocol specification such that (1.)the responsible node sends the current time t_P to every other node just before invoking $P(0)$, and (2.)the other nodes invoke $P(t - t_P)$ after receiving t_P from the responsible node. The time $t - t_P$ corresponds to the actual communication delay from the responsible node.

Note that the process P may be called by another process Q. In such a case, the variable t in $P(t - t_P)$ should be adjusted to represent the virtual time at which Q had been invoked. So it should be modified to $P(t + e_Q - e_P)$ if this process invocation occurs in the right hand of the definition of process Q, where

$t + e_Q$ represents the virtual invocation time of Q.

Example 8

$$P := a^1[2 \le t \le 4 \wedge x = t]; b^2[t \le x + 5]; P \;[]\; c^1[5 \le t]; \mathbf{exit}$$
$$d_{12}\max = 4\ ,\ d_{21}\max = 3$$

$$Node_1 = P(e_P) := a^1[2 \le t + e_P \le 4 \wedge x = t + e_P \wedge \exists t'(t' \ge t + e_P + d_{12}\max \wedge t' \le x + 5)]; s_{12}(m1, x); r_{21}(m3, t_P); P(t + e_P - t_P) >> s_2(m4); \mathbf{exit} [] c^1[5 \le t + e_P]; \mathbf{exit}$$

$$Node_2 = P(e_P) := r_{12}(m1, x); b^2[t + e_P \le x + 5]; s_{21}(m2, x); s_{21}(m3, t + e_P); P(0) \;[]\; r_{12}(m4); \mathbf{exit}$$

□

To make this simulation possible, we check whether the starting action of each process cannot be late if the notification from the responsible node would reach in maximum delay. For consistency, we include this checking into *Restr*(). If the checking is false, the time constraint of the starting action becomes "false."

Definition 9 If $S = P$ where $P := B$, $Restr(S)$ is defined inductively as follows:

$$Restr(S) \stackrel{\text{def}}{=} P \text{ where } P := h(Restr(B))$$

where $h(Restr(B))$ is an expression obtained by replacing the time constraint $Q_k(t, x_k)$ of each starting action a_k of $Restr(B)$ with $Q_k(t, x_k) \wedge Q'_k$. Here Q'_k is a Presburger formula defined as follows :

$$Q'_k \stackrel{\text{def}}{=} \bigwedge_{p=1,\ldots,n} \{\exists t' \exists x_k [t' \ge 0 + d_{p,\text{place}(a_k)}\max \wedge Q(t', x_k)]\}$$

□

3.3 Synthesis Algorithm

The synthesis algorithm consists of two parts:

1. For a given service specification S, an assignment of each action to a node, and a maximum delay $d_{ij}\max$ for each pair of nodes, construct $S' = Restr(S)$.
2. If $S \sim_u S'$, i.e., S and S' are bisimulation equivalent when time is ignored[6], derive a protocol entity specification $Node_i$ of each node i from S'. Otherwise, do not derive and halt.

In [10], we have defined a transformation $T_p(B)$ which derives a protocol entity specification of node p from a service specification described by the behaviour expression B. Although we omit the precise definition of $T_p(B)$ in this paper because of the space limitation, we summarize our result by the following theorem:

Theorem 1 For a given service specification S, let $S' = Restr(S)$. If $S \sim_u S'$, the protocol specification $\{T_i(S')\}_{i=1,2,\ldots,n}$ is $\sqsubseteq_t$-correct w.r.t. the service specification S. □

4 Concluding Remarks

In this paper, we have proposed a method to synthesize protocol specifications from timed service specifications written in LOTOS/T+. The proposed method enables us to synthesize protocol specifications from both timed and structured service specifications. In contrast to [5], our method restricts the time constraints of service specifications, not of the communication media, because the delay of the media depends on the physical lines, so it is more difficult to change them than those of the specifications. Moreover, our correctness criterion guarantees that the control structure of the derived protocol specification is a full, not partial, implementation of that of the service specification. Using the same timing extension as ours, our result should easily apply to other process models such as CCS.

The future work is to extend the class of service specifications and to establish a framework for evaluating performance aspects of the derived protocol entity specifications.

References

[1] R. L. Probert and K. Saleh, "Synthesis of communication protocols: Survey and assessment," *IEEE Trans. Comput.*, vol. 40, pp. 468–475, 1991.

[2] P. M. Chu and M. T. Liu, "Protocol synthesis in a state transition model," in *Proc. IEEE COMPSAC '88*, pp. 505–512, 1988.

[3] C. Kant, T. Higashino, and G. v. Bochmann, "Deriving protocol specifications from service specifications written in LOTOS," in *Proc. of 12th Annual Int'l Phoenix Conf. on Computers and Communications (IPCCC'93)*, pp. 310–318, IEEE, 1993.

[4] H. Yamaguchi, K. Okano, T. Higashino, and K. Taniguchi, "Synthesis of protocol entities' specifications from service specifications in a Petri net model with registers," in *Proc. of 15th IEEE Int'l Conf. on Distributed Computing Systems*, pp.510–517, 1995.

[5] A. Khoumsi, G. v. Bochmann, and R. Dssouli, "On specifying services and synthesizing protocols for real-time applications," in *Protocol Specification, Testing and Verification, XIV*, pp. 185–200, IFIP, Chapman & Hall, 1995.

[6] A. Nakata, T. Higashino, and K. Taniguchi, "LOTOS enhancement to specify time constraints among non-adjacent actions using first order logic," in *Formal Description Techniques, VI (FORTE'93)*, pp. 451–466, IFIP, North-Holland, 1994.

[7] ISO, *LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour.* IS 8807, 1989.

[8] K. Yasumoto, T. Higashino, and K. Taniguchi, "Software process description using LOTOS and its enaction," in *Proc. of 16th IEEE Int'l Conf. on Software Engineering (ICSE-16)*, pp. 169–179, 1994.

[9] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[10] A. Nakata, T. Higashino, and K. Taniguchi, "Synthesis of protocol entity specifications from timed and structured service specifications," I.C.S. Research Report 95-ICS-5, Dept. of Information and Computer Sciences, Osaka University, 1995.