Protocol Synthesis from Time Petri Net Based Service Specifications

Hirozumi Yamaguchi Kozo Okano Teruo Higashino Kenichi Taniguchi

Department of Information and Computer Sciences
Osaka University
Machikaneyamacho 1-3 Toyonaka Osaka 560 JAPAN

Abstract

Some methods for deriving protocol specifications from given service specifications with time constraints have been proposed. However, existing methods cannot treat the class of service specifications with both parallel synchronization and data values. They also assume that all clocks in the distributed system are synchronized. In this paper, we propose an algorithm to derive a correct protocol specification automatically from a given service specification described in an extended model of time Petri nets where the above restrictions are eliminated. Using our method, we will be free from considering the details of communication delays on the design of real-time distributed systems.

1 Introduction

In general, for a distributed system, two different levels of abstraction can be considered. On a highly abstracted level, a distributed system is treated as one protocol entity (one node). On the other hand, it is treated as a set of protocol entities on an implementation level. Specifications at the former and latter levels are called a service specification and a protocol specification, respectively. A protocol specification is a set of protocol entities' specifications (protocol entity specifications).

Figure 1 shows examples of a service specification and a protocol specification. The service specification is described as the actions (e.g. "a", "b" and "c") and their execution order (e.g. "a" and "b" can be executed in parallel and "c" must be executed after their execution). The protocol specification consists of three protocol entity specifications. For example, the protocol entity #1's specification is described as the action "a", a communicating action (sending a notification of the action a's completion to the protocol entity #3) and their execution order. Both specifications provide the same services at the Service Access Points (SAP's) "a", "b" and "c". However, only the protocol specification contains communicating actions among protocol entities.

Although a protocol specification is necessary for implementation, it is complex and troublesome to describe it because communications among protocol entities must be specified. Thus, some approaches for synthesizing a protocol specification automatically from a given service specification has been proposed [3, 4, 5, 8](see [2] for survey).

However, those approaches are not sufficient for real-time distributed systems since they do not consider the urgency of services (time constraints). For example, in Fig. 1, the time constraint "the action "c" must be executed between 3 and 10 units of time after both "a" and "b" are executed" is given. Here, we don't assume that there exist synchronous clocks among the protocol entities. Also we assume that there are communication delays from the protocol entity (PE) #1 to PE #3 (2 and 4 units of time at the shortest and longest, respectively) and from PE #2 to PE #3 (1 and 3 units of time at the shortest and longest, respectively). In this case, PE #3 can only know that at least 1 unit of time has passed after both "a" and "b" were executed, and that at most 4 units of time have passed. Therefore, even if the notification messages are sent immediately after the actions "a" and "b" are executed, the action "c" must be executed between 2 and 6 units of time (e.g. **[2, 6]**) after those messages are received by PE #3 in order to guarantee the time constraint **[3, 10]** of the service specification. It is more complex and troublesome to decide suitable time constraints at the level of protocol specifications under the presence of communication delays, satisfying the time constraints of service specifications.

For service specifications with time constraints, a timed automaton based approach in [6] and a timed LOTOS based approach in [7] have been proposed. Although these approaches can handle service specifications with time constraints between non-successive actions, they assume the existence of synchronous clocks in the distributed environment and cannot treat par-

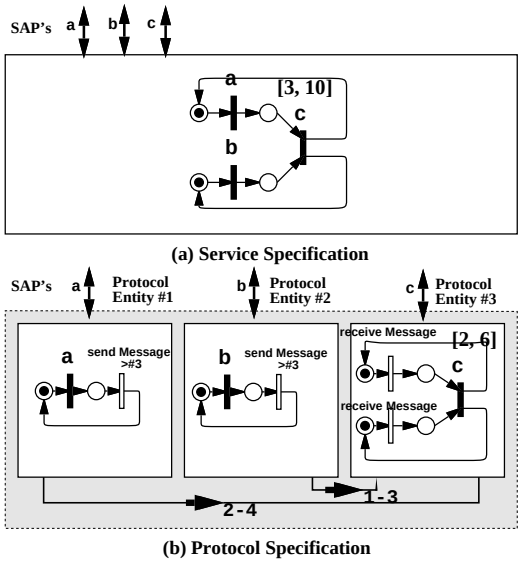

Figure 1: Service Specification and Protocol Specification.

allel synchronization in service specifications. In addition, the system's state variables (parameters) are not considered although most systems have state variables. These are restrictions for practical real-time applications.

In this paper, we propose a method for deriving a protocol specification from (a) a service specification written in an extended model of time Petri nets [1], (b) a resource allocation and (c) maximum/minimum communication delays among protocol entities. The extended model is called a Time Petri Net model with Registers (TPNR model in short) where state variables can be treated. A resource allocation specifies an allocation of SAP's (communication ports) and state variables to protocol entities. The derived protocol specification satisfies the time constraints in the service specification.

For any pair of actions which are time-dependent of each other, they are described as successive actions explicitly in TPNR model. Therefore, only by estimating the time when a previous action is executed from the minimum/maximum communication delays for each received message, we can propose a new synthesis technique on a distributed environment where synchronous clocks are not available.

2 Specifications

2.1 Our Model

We extend time Petri nets (TPN) so that we can formalize I/O's and calculation of state variables. TPNR model has a finite number of I/O gates and a finite number of registers. The I/O gates represent I/O ports from/to external environments, and the registers represent systems' state variables. A pair of an I/O event and a set of substitution statements of registers can be specified as the action of a transition. The formal definition of TPNR model is shown in [10].

Intuitively, the behavior of a transition t is as follows. Suppose that the transition t has a label $\langle\ R_2 > R_1,\ a?x,\ \text{“}R_1 \leftarrow R_2+x,\ R_2 \leftarrow R_1+R_2+x\text{”}\ \rangle$ and a time constraint **[3, 5]** (Fig. 2).

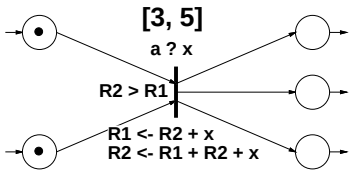

Figure 2: TPNR Model.

Assume that the current values of registers "R_1" and "R_2" are 1 and 2, respectively. If each input place of t has at least one token at time T, the value of the guard expression (condition) "$R_2 > R_1$" is checked. Since it is true in this case, it becomes enabled at time T. The transition t must fire between times $T+3$ and $T+5$ unless it becomes disabled by time $T+5$. If t fires, the I/O event "$a?x$" is executed (suppose that the input value is 3), and then the substitution statements "$R_1 \leftarrow R_2 + x,\ R_2 \leftarrow R_1 + R_2 + x$" are executed in parallel. In this case, the new values of "R_1" and "R_2" are calculated as 5 ($= 2+3$) and 6 ($= 1+2+3$), respectively and stored. Then the marking is changed to new one.

2.2 Service Specification

Figure 3(a) shows a service specification of an example protocol sub-layer in TPNR model. Hereafter we denote the service specification by *Sspec*.

The example is a simple connection management sub-layer, which may be a typical example using state variables. It communicates with its upper sub-layer, which consists of three processes A (request process), B (network observation process) and C (connection policing process), via I/O gates a, b and c. It has a topology table (register R_4). Registers R_1, R_2 and R_3 are used as temporary buffers.

The sub-layer first receives a connection request (the value of input variable x) from process A (via I/O gate a) and stores it to R_1 (request buffer). Then it gets the current network traffic information (the value of input variable y) from process B (via I/O gate b) and calculates a route using its topology table (R_4), the request (R_1) and the network traffic information (y). Then the calculated route is stored to R_3 (route info. buffer). It also receives an acknowledgment for

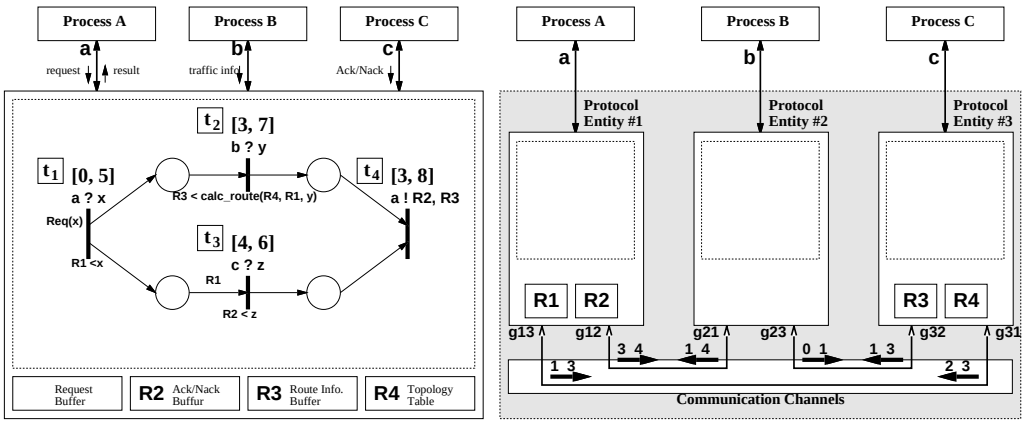

Figure 3: (a) Service Specification of Protocol Sub-layer. (b) Protocol Entities.

the request (Ack or Nack, the value of input variable z) from process C (via I/O gate c) and stores it to R_2 (Ack/Nack buffer). After those, it passes a pair of the acknowledgment and the route to process A. In all the figures in this paper, guard expressions whose values are identically "true" and empty substitution statements of registers are omitted. "**calc_route**" is a user-defined function.

In this paper, we assume the following restrictions for service specifications.

[Restriction 1] The Petri net of $Sspec$ must be a live and safe free-choice net [1].

[Restriction 2] There may be two transitions t_i and t_j which can fire in parallel in $Sspec$. For a register R, if the value of R is substituted in both t_i and t_j simultaneously, or if the value of R is substituted in t_i and it is referred in t_j, we call such a case a register conflict. $Sspec$ must not include any register conflict.

[Restriction 3] $Sspec$ contains no internal events (it is not an essential restriction). 2

A Petri net is a free-choice (FC) net if, for any pair of transitions which share an input place p, they do not have other input places except p. A live and safe Petri net is deadlock-free and overflow-free. In a FC net, the decision which side of transitions fires in each choice structure is done only by a token in the input place p. Restriction 1 simplifies the control flow of $Sspec$, and therefore simplifies the derivation algorithm[1]. The reason to assume Restriction 2 is to avoid register conflict problems [8].

2.3 Protocol Specification

On the protocol level, the services are provided by p protocol entities. We assume that the example sublayer consists of three protocol entities #1, #2 and #3 (see Fig. 3(b)). Each protocol entity #k (denoted by PE_k) has some of I/O gates and registers and manages them independently of the other protocol entities. For any pair of PE_i and PE_j ($i \neq j$), we assume that there is a full duplex communication channel. The port on the side of PE_i (PE_j) of the communication channel is called gate g_{ij} (g_{ji}). We assume that each communication channel is sufficiently reliable and that the maximum/minimum communication delays are bounded by constant values. We can specify a resource allocation (denoted by Alloc($Sspec$)) which allocates the I/O gates and registers to protocol entities. We can also specify maximum/minimum communication delays from PE_i to PE_j (denoted by $Dmax_{ij}/Dmin_{ij}$). These are shown in Fig. 3(b). Here, we assume the following restriction for $Sspec$ and Alloc($Sspec$).

[Restriction 4] For any pair of transitions t_i and t_j which share an input place p in $Sspec$, their I/O gates must be allocated to one protocol entity in Alloc($Sspec$). 2

Restriction 4 prohibits distributed choice.

Hereafter, we denote the protocol entity specification of PE_k by $Pspec_k$, and the set of p protocol entity specifications by $Pspec^{\langle 1,p \rangle}$. $Pspec^{\langle 1,p \rangle}$ is a protocol specification, and $Pspec^{\langle 1,3 \rangle}$ in Fig. 4 is an example of a protocol specification.

The protocol entities communicate with each other by exchanging messages in order to provide the same behavior in $Sspec$. For example, PE_1 executes the I/O event "$a?x$" of t_1 (since PE_1 has the I/O gate a) and sends two messages to PE_2 and PE_3 for notifying the completion of the I/O event execution. The messages "$\mathrm{Mn}_{1.12}$" and "$\mathrm{Mn}_{1.13}$" are sent to PE_2 and PE_3 via I/O gates g_{12} and g_{13}, respectively. PE_2 and PE_3 receive these messages and then execute the I/O events of the next transitions t_2 and t_3, that is, "$b?y$" and

[1]For a given free-choice net, there exist polynomial algorithms for safeness (boundness) and liveness problems [9].

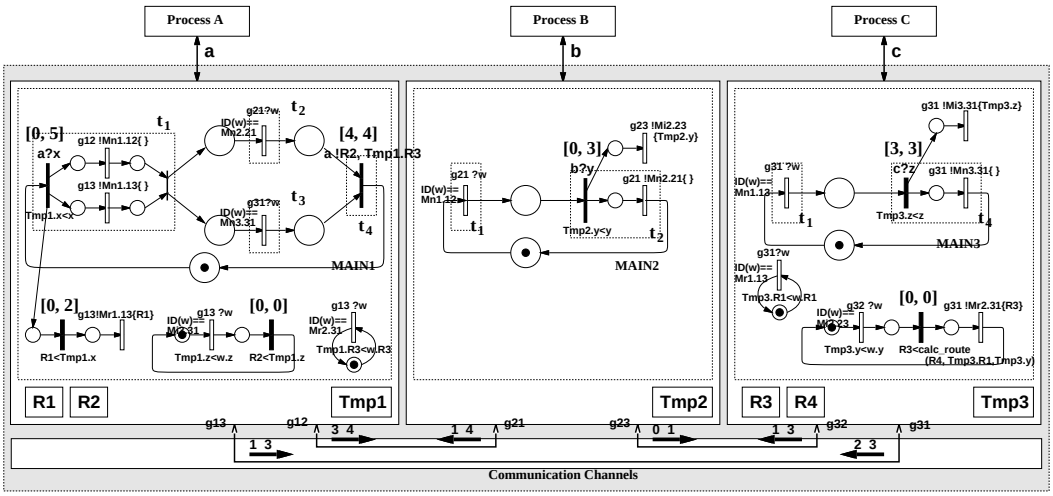

Figure 4: Protocol Specification of the Sub-layer.

"$c?z$", respectively. On the other hand, PE_1 executes the substitution statement of register R_1 (since PE_1 has register R_1) using the value of input variable x and sends the message to transfer the latest value of the register. The message "$\mathrm{Mr}_{1.13}$" is sent to PE_3 to transfer the value of R_1. PE_3 receives the message and stores the received value as the latest value of R_1 to its local temporary register, Tmp_3[2]. This value is used to execute the substitution statement of register R_3 (See Fig. 4 to find $Tmp_3.R_1$ in the second argument of the substitution statement). In Fig. 4, for simplicity, we assume that sending/receiving transitions can be executed immediately. So, the time constraints of all the sending/receiving transitions (represented by white rectangles) are **[0, 0]**, and they are omitted[3].

3 Derivation Problem

In this section, we formally define the derivation problem treated in this paper.

Suppose that all I/O events executed on the I/O gates used in $Sspec$ are treated as observable, and that all sending/receiving events on the I/O gates for communication among protocol entities (such as "$g_{ij}?x$" and "$g_{ij}!E(\ldots)$") and internal events ("i") are treated as unobservable. If all the observable I/O event sequences (including a time interval between each pair of successive I/O events in the sequences) in $Sspec$ can be observed in $Pspec^{\langle 1,p\rangle}$ and vice versa, we say that $Sspec$ and $Pspec^{\langle 1,p\rangle}$ are equivalent.

[2]We assume that each PE_k has a local temporary register Tmp_k to keep the latest received value of registers and input variables. $Tmp_k.R_q$ ($Tmp_k.x$) denotes such a value of register R_q (input variable x).

[3]We can easily extended our method for treating the case that it takes some units of time to execute sending/receiving transitions (See Section 5).

It is ideal that for any given $Sspec$, $Pspec^{\langle 1,p\rangle}$ which is equivalent to $Sspec$ can be derived. However, considering communication delays, for almost cases such a derivation is impossible. For example, suppose that there is an I/O event sequence "$a;b$" in $Sspec$ and the time constraint of b is **[3, 6]**. In $Pspec^{\langle 1,p\rangle}$, also suppose that PE_1 and PE_2 have the I/O gates a and b, respectively and $Dmin_{12}$ and $Dmax_{12}$ are 4 and 5 units of time, respectively. PE_2 must know that the action a has been executed in PE_1, therefore PE_1 sends a message after the execution of the action a. PE_2 receives the message and executes the action b. If the message arrives at PE_2 in the shortest delay (4 units of time), the action b can be executed immediately after receiving the message. On the other hand, if the message arrives at PE_2 in the longest delay (5 units of time), the action b must be executed within 1 unit of time after receiving the message. As a result, the time constraint **[3, 6]** of the action b in $Sspec$ should be modified to **[4, 6]** ($=[Dmin_{12}+0, Dmax_{12}+1]$) in $Pspec^{\langle 1,p\rangle}$. In this case, although $Pspec^{\langle 1,p\rangle}$ satisfies the time constraint of $Sspec$, it is not equivalent to $Sspec$.

Here, considering the case like the above, we define the correctness of $Pspec^{\langle 1,p\rangle}$ w.r.t. $Sspec$ as follows. Suppose a specification denoted by $Sspec'$ which is obtained from $Sspec$ by narrowing time constraints of some transitions in $Sspec$. We say that $Pspec^{\langle 1,p\rangle}$ is correct w.r.t. $Sspec$ if, there exists $Sspec'$ where (A) $Sspec'$ and $Pspec^{\langle 1,p\rangle}$ are equivalent and (B) selectable transitions in $Sspec$ can be also selectable in $Pspec^{\langle 1,p\rangle}$. The condition (A) means that if the time constraints in $Pspec^{\langle 1,p\rangle}$ are narrowed due to communication delays, $Pspec^{\langle 1,p\rangle}$ is correct as far as their time constraints are within those in $Sspec$ like the

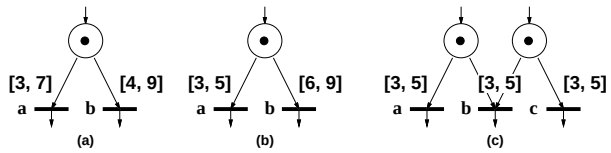

Figure 5: Narrowing Time Constraints in a Choice.

above case. However, narrowing time constraints may change selectability in a choice structure. In the specification in Fig. 5(a), both actions a and b can be executed. On the other hand, in the specification in Fig. 5(b), which is time-narrowed specification in Fig. 5(a), the action b cannot be executed according to the firing rule of TPN (the action a MUST be executed within 5 units of time). That is, the action b is a "dead " (unexecutable) transition. The condition (B) is used to prevent "dead" transitions appeared in $Pspec^{\langle 1,p\rangle}$. Note that in a choice structure as shown in Fig. 5(c) which is not a free-choice net, checking the selectability is difficult, since it depends on time when two independent tokens come into the places. In our method, we restrict the class of the Petri net of a service specification to a free-choice net (See Restriction 1 in Section 2.2). Free-choice nets have a simple choice structure where input places are at most 1. It facilitates to guarantee the correctness of derived $Pspec^{\langle 1,p\rangle}$, which will be discussed in Section 4.2.

In this paper, we treat a problem to derive a correct $Pspec^{\langle 1,p\rangle}$ from a given tuple of $Sspec$, Alloc($Sspec$) and $Dmin_{ij}/Dmax_{ij}$ for each communication channel where Restrictions 1, 2, 3 and 4 hold.

4 Algorithm

Our algorithm consists of two steps : (1) deriving each protocol entity specification without time constraints from $Sspec$ according to a simulation policy, and then (2) deciding its time constraints.

4.1 Deriving Nets

At the step (1), from the given $Sspec$ and Alloc($Sspec$), the actions and their execution order at each protocol entity are decided uniquely based on the following simulation policy (e.g. PE_1 executes the I/O event "a?x" of t_1, and then sends messages "$Mn_{1.12}$" and "$Mn_{1.13}$"). According to the policy, $Pspec^{\langle 1,p\rangle}$ without time constraints are constructed.

[Simulation Policy]

(a) For each t_j in $Sspec$, the protocol entity which has the I/O gate of the I/O event of t_j executes the I/O event. Such a protocol entity is called a responsible protocol entity of t_j and denoted by RPE(t_j). RPE(t_j) receives messages from all of the protocol entities, each of which has executed the I/O event of a previous transition of t_j. These messages are called I/O completion notification messages. Also RPE(t_j) sends a message to each protocol entity which executes the substitution statement for a register R_q of t_j. This is explained in (b).

(b) Each protocol entity executes the substitution statement for a register R_q of t_j if it has the register R_q. If it receives a message (called an input value transfer message), which includes the value of input variables, from RPE(t_j), then it calculates the new value of the register and stores it to R_q. Then it sends the new value in advance to all protocol entities which may need it to execute future I/O events and substitution statements of registers, or to evaluate guard expressions of future transitions. These messages are called register value transfer messages.

In Fig. 4, the components MAIN-1, MAIN-2 and MAIN-3 are derived according to the policy (a). Each of the components is derived by replacing a transition in $Sspec$ with a subnet. For example, PE_2 and PE_3 execute the I/O events of t_2 and t_3 ("$b?y$" and "$c?z$") and send I/O completion notification messages "$Mn_{2.21}$" and "$Mn_{3.31}$"[4] to PE_1, respectively. In this case, in PE_2, the transition t_2 in $Sspec$ is replaced by a subnet which has two transitions. One executes the I/O event of t_2 and the another sends the message "$Mn_{2.21}$" after that. PE_1 can know that the execution of the I/O events of t_2 and t_3 has been finished. The rest of the components in Fig. 4 are derived according to the policy (b). They are derived by adding some transitions or sub-nets. PE_2 sends an input value transfer message "$Mi_{2.23}$" to PE_3, which includes the value of input variable y. In this case, a transition sending the message "$Mi_{2.23}$" is added to the net MAIN-2 in PE_2. PE_3 receives the message, calculates the new value of R_3, and sends PE_1 the register value transfer message "$Mr_{2.31}$", which includes the new value of R_3. Also in this case, in PE_3, the net which is a sequence of three transitions (a receiving transition, a transition executing the substitution statement and a sending transition) is added. PE_1 receives the message and it can know the latest value of R_3, which is necessary for executing the I/O event of t_4. PE_1 has a transition receiving the message which forms a self-loop.

4.2 Deciding Time Constraints

At the step (2), time constraints of $Pspec^{\langle 1,p\rangle}$ are decided. In $Pspec^{\langle 1,p\rangle}$ derived at the step (1), only

[4] A message ID is defined as "$Mt_{i.xy}$", where t is its message type (n, i and r denote I/O completion notification, input value transfer and register value transfer, respectively), i is the transition name concerned with the message, and $\#x/\#y$ are the source/destination protocol entities, respectively.

[For Condition (a)]

- For each pair of a transition t_j and its previous transition t_i :

$$\begin{aligned} ETmin(t_j) &\leq ETmax(t_j) && (1)\\ \mathrm{Eft}(t_j) \leq Dmin_{uv} + ETmin(t_j) &\leq Dmax_{uv} + ETmax(t_j) \leq \mathrm{Lft}(t_j) && (2) \end{aligned}$$

where we assume that RPE(t_i) and RPE(t_j) are protocol entities $\#u$ and $\#v$, respectively.

[For Condition (b)]

- For each transition t_j in $Sspec$ and each register R_q whose value is substituted in t_j :

$$RTmin(R_q, t_j) \leq RTmax(R_q, t_j) \quad (3)$$

- For each transition t_i in $Sspec$ and each register R_q whose value is substituted in t_i, suppose that the new value of R_q is used for the execution of the I/O event of a future transition t_j in $Sspec$. Assume that RPE(t_i) and RPE(t_j) are protocol entities $\#u$ and $\#w$ respectively, and that a protocol entity $\#v$ has the register R_q. Then the following must hold :

$$min\_seq(t_i, t_j)_h \geq Dmax_{uv} + RTmax(R_q, t_i) + Dmax_{vw} \quad (4)$$

where each $min\_seq(t_i, t_j)_h$ $(h = 1, 2, ..., r)$ is the minimum time from the I/O event of t_i has been executed to the I/O event of t_j becomes executable in $Pspec^{\langle 1,p \rangle}$ (the minimum execution time of I/O event sequences in $Pspec^{\langle 1,p \rangle}$).

[For Condition (c)]

- For each pair of transitions t_{j_a} and t_{j_b} which share an input place p and a transition t_i which has p as an output place, assume that RPE(t_i) is a protocol entity $\#u$ and RPE(t_{j_a}) and RPE(t_{j_b}) are the same protocol entity $\#v$ (See Restriction 4 in Section 2.3). If $\mathrm{Eft}(t_{j_a}) \leq \mathrm{Lft}(t_{j_b})$ and $\mathrm{Eft}(t_{j_b}) \leq \mathrm{Lft}(t_{j_a})$ hold:

$$\begin{aligned} ETmin(t_{j_a}) &\leq ETmax(t_{j_b})\\ ETmin(t_{j_b}) &\leq ETmax(t_{j_a}) && (5) \end{aligned}$$

[Objective Function]

$$OBJ = \sum_{t_i} (\ ETmax(t_i) - ETmin(t_i)\ ) + \sum_{R_q} \sum_{t_j} (\ RTmax(R_q, t_j) - RTmin(R_q, t_j)\ )$$

Figure 6: Integer Linear Inequalities and Objective Function.

the execution order of the I/O events in $Sspec$ is implemented by exchanging I/O completion notification messages. Therefore, in order to obtain a correct $Pspec^{\langle 1,p \rangle}$, the following three issues must be guaranteed at the step (2).

(a) Each time interval between a pair of successive I/O events in $Pspec^{\langle 1,p \rangle}$ must satisfy that in $Sspec$.

(b) The input/output values of I/O event sequences in $Pspec^{\langle 1,p \rangle}$ must be equivalent to those in $Sspec$. In our simulation policy in Section 4.1, the execution of an I/O event which needs the latest value of a register does not wait for the arrival of the register value transfer message (as a result, the I/O event may be executed with an old value). Therefore, we need guarantee that each register value transfer message has always arrived before the I/O event becomes executable.

(c) Considering the case discussed in Section 3, selectable transitions in each choice structure in $Sspec$ must be also selectable in $Pspec^{\langle 1,p \rangle}$. 2

However, since there are message delays, such time constraints of $Pspec^{\langle 1,p \rangle}$ that satisfy all the above (a), (b) and (c) may not exist. For example, the message "$\mathrm{Mr}_{3.31}$" includes the value of R_3 necessary for the execution of the I/O event of t_4 in PE_1. Although it is sent immediately after the execution of the substitution statement of R_3, it may not arrive at PE_1 in time (therefore, it may not be able to satisfy the condition (b)). On the other hand, if we delay the earliest executable time of the I/O event of t_4, the message may be able to arrive at PE_1 before I/O event of t_4 becomes executable.

At the step (2), the time constraints of $Pspec^{\langle 1,p \rangle}$ derived at the step (1) are represented by some non-negative integer variables and the conditions (a), (b) and (c) are represented as integer linear inequalities over those variables. If there exists a solution which

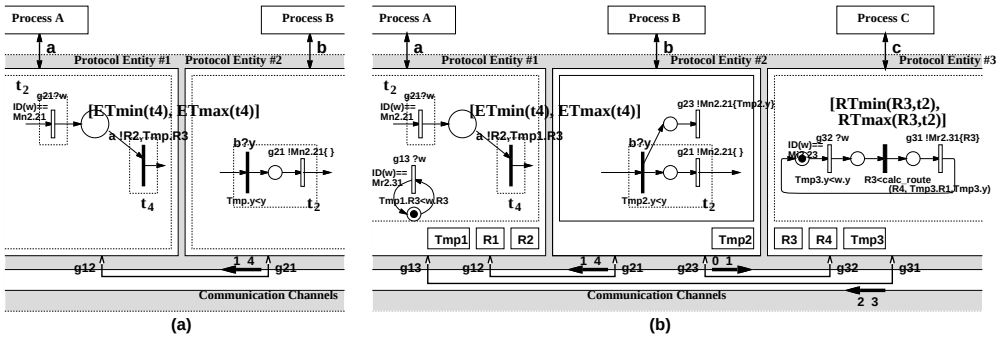

Figure 7: Example.

satisfies all of the inequalities, the time constraints of $Pspec^{\langle 1,p\rangle}$ satisfy the above conditions (a), (b) and (c). Using a procedure to solve integer linear programming problems, a solution can be obtained where the total sum of time ranges of time constraints in $Pspec^{\langle 1,p\rangle}$ is maximized.

[Introduced Variables]

• For each transition in $Pspec^{\langle 1,p\rangle}$ executing the I/O event of t_i, we introduce two non-negative integer variables, and represent the time constraint of the transition as $[ETmin(t_i),\ ETmax(t_i)]$. $ETmin(t_i)$ and $ETmax(t_i)$ represent the minimum and maximum time from the time when it receives all the I/O completion notification messages to the time when it executes the I/O event of t_i, respectively.

• For each transition in $Pspec^{\langle 1,p\rangle}$ executing the substitution statement of register R_q in t_i, we also introduce two non-negative integer variables, and represent the time constraint of the transition as $[RTmin(R_q,t_i),\ RTmax(R_q,t_i)]$. $RTmin(R_q,t_i)$ and $RTmax(R_q,t_i)$ represent the minimum and maximum time from the time when it receives the input value transfer message to the time when it calculates the new value of R_q, respectively.

• We assume that the time constraint of each sending/receiving transition is **[0, 0]**. That is, we assume every sending/receiving event is executed immediately after it becomes executable, as explained in Section 2.3.

[Integer Linear Inequalities]

Fig. 6 shows integer linear inequalities over the above variables to guarantee the conditions (a), (b) and (c).

For the condition (a), two types of inequalities are given. Inequality (1) is obviously necessary from the definition of those variables. Inequality (2) guarantees that even if there is the delay of an I/O completion notification message between two successive I/O events, the time interval between the I/O events in $Pspec^{\langle 1,p\rangle}$ must be within that in $Sspec$.

For the condition (b), two types of inequalities are also given. Inequality (3) is also necessary from the definition of those variables like Inequality (1). Inequality (4) guarantees that the new value of R_q necessary for the execution of an I/O event of t_j must arrive at PE_w before the I/O event becomes executable. We consider the time when the I/O event of t_i has been executed as the base time (denoted by T). The earliest time when the I/O event of t_j becomes executable can be represented as $T + min\_sequence(t_i,t_j)_h$ $(h = 1,2,...,r)$. On the other hand, the latest time when the value of R_q arrives at PE_w can be represented as $T + Dmax_{uv} + RTmax(R_q,t_i) + Dmax_{vw}$. Therefore if Inequality (4) holds, the new value of R_q necessary for the execution of an I/O event of t_j is in time for the executable time of the I/O event of t_j. This is the main idea of the register value transfer.

For the condition (c), Inequality (5) is given. $\mathrm{Eft}(t_{j_a}) \leq \mathrm{Lft}(t_{j_b})$ and $\mathrm{Eft}(t_{j_b}) \leq \mathrm{Lft}(t_{j_a})$ represent that both t_{j_a} and t_{j_b} are selectable in $Sspec$. Inequality (5) represents the condition to guarantee that t_{j_a} and t_{j_b} are selectable in $Pspec^{\langle 1,p\rangle}$.

Finally, we get a solution which satisfies all of the inequalities (1) - (5) using a procedure for solving integer linear programming problems. If such a solution exists, we regard that the time constraints in $Sspec$ also hold in $Pspec^{\langle 1,p\rangle}$. Here, we would like to get elastic time constraints whose ranges are possibly wide (this is our evaluation basis for optimization). Therefore, we represent the total sum of the range of time constraints in $Pspec^{\langle 1,p\rangle}$ as the objective function OBJ, where the first and second terms represent the time range of the I/O events and the register value substitutions, respectively. Then we get an optimal solution to maximize the objective function OBJ.

In general, it may take much time to solve integer linear programming problems. However, since coefficients of the inequalities in Fig. 6 are all "1" and the

maximum/minimum communication delays are integers, our experience shows that, in most cases, the solution can be derived using a procedure to solve linear programming problems such as simplex method. And the simplex method can generate a solution very efficiently in most cases.

4.2.1 Example

For the transition executing the I/O event of t_4 and one of its previous transitions t_2, the following inequalities must hold according to Inequalities (1) and (2) (see Fig. 7(a)) :

$$\begin{aligned} ETmin(t_4) &\leq ETmax(t_4) \\ \mathrm{Eft}(t_4) &\leq Dmin_{21} + ETmin(t_4) \\ &\leq Dmax_{21} + ETmax(t_4) \\ &\leq \mathrm{Lft}(t_4). \end{aligned}$$

That is,

$$2 \leq ETmin(t_4) \leq ETmax(t_4) \leq 4.$$

Moreover, for transition t_2 and register R_3, the calculated value of R_3 is used for the I/O event of t_4. Therefore, according to Inequality (4), the following inequality must hold (see Fig. 7(b)) :

$$min\_seq(t_2, t_4)_1 \geq Dmax_{23} + RTmax(R_3, t_2) + Dmax_{31}.$$

Here, $min\_seq(t_2, t_4)_1 = Dmin_{21} + ETmin(t_4)$. The left-side expression represents the earliest time when the I/O event of t_4 becomes executable, from the time T (when the I/O event of t_2 has been executed). On the other hand, the right-side expression represents the latest time when the register value transfer message "$\mathrm{Mr}_{2.31}$" arrives at PE_1, from the same time T. The inequality can be transformed to :

$$ETmin(t_4) - RTmax(R_3, t_2) \geq 3.$$

Of course, inequalities for the other transitions are needed.

5 Conclusion

In this paper, we have proposed a method to derive a correct protocol specification from a given service specification in TPNR model, a resource allocation and maximum/minimum communication delays among protocol entities.

For practical use, many system-specific things must be considered. For example, in distributed real-time databases, data with large capacity may be treated. Therefore the execution time of actions, such as the retrieval time of data, may not seem to be zero, while we assume it is zero in our method. In order to treat such a case, we only change Inequality (4) in Fig. 6 to :

$$\begin{aligned} min\_seq(t_i, t_j)_h > {} & Dmax_{uv} + RTmax(R_q, t_i) \\ & + max\_exec(R_q, t_i) + Dmax_{vw} \end{aligned}$$

where $max\_exec(R_q, t_i)$ represents the maximum execution time of the substitution statement of R_q. Our concept itself can be applied to many real-time distributed systems by slightly modifying the inequalities in Fig. 6, even though there are some system-specific aspects practically.

References

[1] Murata, T. : "Petri Nets: Properties, Analysis and Applications," *Proc. of the IEEE*, Vol. 77, No. 4, pp. 541-580, 1989.

[2] Saleh, K. : "Synthesis of Communication Protocols: an Annotated Bibliography," *ACM SIGCOMM Computer Communication Review*, Vol. 26, No. 5, pp. 40-59, 1996.

[3] Chu, P.-Y.M. and Liu, M.T. : "Protocol Synthesis in a State-Transition Model," *Proc. of the COMPSAC '88*, pp. 505-512, 1988.

[4] Chao, D. Y. and Wang, D. T. : "A Synthesis Technique of General Petri Nets," *Journal of System Integration*, Vol. 4, pp. 67-102, 1994.

[5] Kant, C., Higashino, T. and Bochmann, G. v. : "Deriving Protocol Specifications from Service Specifications Written in LOTOS," *Distributed Computing*, Vol. 10, No. 1, pp. 29-47, 1996.

[6] Khoumsi, A., Bochmann, G.v. and Dssouli, R. : "On Specifying Services and Synthesizing Protocols for Real-time Applications," *Proc. of the 14th IFIP WG6.1 Symp. on Protocol Specification, Testing and Verification (PSTV-XIV)*, pp.185-200, 1994.

[7] Nakata, A., Higashino, T. and Taniguchi, K. : "Protocol Synthesis from Timed and Structured Specifications," *Proc. of the Int. Conf. on Network Protocols (ICNP'95)*, pp. 74-81, 1995.

[8] Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K. : "Synthesis of Protocol Entities' Specifications from Service Specifications in a Petri Net Model with Registers," *Proc. of the 15th Int. Conf. on Distributed Computing Systems (ICDCS-15)*, pp. 510-517, 1995.

[9] P. Kemper and F. Bause : "An Efficient Polynomial-Time Algorithm to Decide Liveness and Boundedness of Free-Choice Nets," *Proc. of the Int. Conf. on Application and Theory of Petri Nets 1992, LNCS* Vol. 616, pp. 263-278, 1992.

[10] Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K. : "Protocol Synthesis from Time Petri Net Based Service Specifications," *I.C.S Research Report*, 97-ICS-2, 1997.