Automated Synthesis of Protocol Specifications with Message Collisions and Verification of Timeliness

Yoshiaki Kakuda, Hirotaka Igarashi† and Tohru Kikuno

Department of Information and Computer Sciences
Faculty of Engineering Science, Osaka University
1-3, Machikaneyama-cho, Toyonaka-shi, Osaka 560, Japan
Phone: +81 6 850 6566 Fax: +81 6 850 6569
E-mail: kakuda@ics.es.osaka-u.ac.jp

Abstract

Protocol synthesis is to derive a protocol specification based on a service specification. In the previous protocol synthesis methods, if the service specification includes simultaneous transmission of primitives, then the derived protocol specification includes protocol errors of unspecified receptions caused by message collisions.

This paper extends a class of derived protocol specifications to include message collisions which often happen in real communication protocols, and proposes a protocol synthesis method such that (1) simultaneous transmission of primitives causing message collisions can be described in service specifications, and (2) transitions for avoiding protocol errors of unspecified receptions can be generated by new transition synthesis rules. This paper also proposes a verification method for determining a real-time bound in the synthesized protocol specification using the task scheduling algorithm for multiprocessor systems.

1 Introduction

Along with the diversification of communication services in the Intelligent Network, it is strongly demanded to develop highly dependable and realtime communication protocols efficiently [5]. The research field to make such protocols is called " Protocol Engineering" [1, 2, 7, 8, 9] and many researchers are engaged in the study of that field. Protocol engineering contains many kinds of techniques. Protocol synthesis for design of protocols is to produce such a protocol specification based on a service specification and it is one of the most important design techniques to be developed.

In the protocol synthesis, given a service specification that defines relations on service primitives between a user in a high layer and processes in a low layer, a protocol specification that defines relations on messages between processes in the low layer is derived. The interface between these two layers is called *Service Access Point*(SAP).

Protocol synthesis methods based upon *Finite State Machine*(FSM) model are proposed by Kassem Saleh [8] and Ming T. Liu et al. [1, 2]. Such methods do not take into consideration about such a case that two users send primitives simultaneously to processes through different SAPs. If this case happens, then a message collision occurs by sending messages from process 1 to process 2 and from process 2 to process 1. It causes protocol errors called unspecified receptions.

Figure 1 shows a sequence chart representing a message collision. A message collision caused by simultaneous transmission of primitives Rel_req1 and C_resp2 is denoted by crossing of two dotted lines. Message collisions usually occur in real communication protocols.

To cope with the drawback of the previous synthesis methods and difficulty of synthesis of protocol specifications with message collisions, this paper proposes an automated synthesis method of a protocol specification which is free from protocol errors of unspecified receptions caused by message collisions. In addition, this paper gives an efficient verification method for determining whether execution times along any acyclic sequence in the synthesized protocol specification are not beyond a given realtime bound. In this method, the number of all sequences transitions in the protocol specification is restricted to a polynomial with the specification size. This enables the efficient verification of the realtime bound.

The organization of the rest of this paper is as follows. Section 2 gives definitions of service specifications and protocol specifications and formulates the problems to be discussed in this paper. In Section 3, details

†He is currently with Central Japan Railway Co., Ltd.

0-8186-6685-4/94 $4.00

of the proposed method is described. In Section 4, a verification method for determining a realtime bound in the synthesized protocol specification is proposed. Finally Section 5 concludes the paper with future researches.

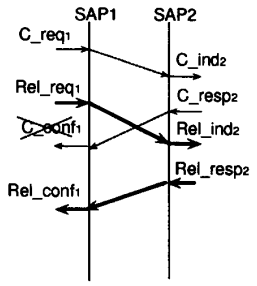


Figure 1 Sequence chart with a message collision

2 Preliminaries

2.1 Communication Model

The interactions between each pair of a user and a process through a SAP are based on the rendezvous communication such that primitives are delivered when the user and the process agree with each other. However, interactions between processes are based on the asynchronized communication such that each process independently sends and receives messages and messages sent from one process are reliably received by the other process in this order through each communication link. Protocol synthesis in cases that communication links are unreliable and synchronization loss between processes occurs is discussed in [1, 4, 5].

2.2 Service Specification

A service specification describes the primitive's execution sequences between users and processes through SAPs. In this paper, we assume that the number of processes is two, and thus the service access points are denoted by SAP1 and SAP2.

Definition 1

A service specification is modeled by a *Finite State Machine*(FSM) $<S_s, \Sigma_s, T_s, Time_s, \sigma_s>$ where

- S_s is a non-empty finite set of service states (or simply states).
- Σ_s is a finite set of service primitives (or simply primitives). A primitive $p \in \Sigma_s$ is characterized by its attribute. That is service access point sap(p) through which primitive passes, SAP1 or SAP2. Sap(p)=1 if p is delivered through SAP1, and sap(p)=2 if p is delivered through SAP2. Thus if sap(p)=1 is satisfied, then p is denoted by p_1. Otherwise p is denoted by p_2. (In the following, we use the notations p, p_1 and p_2 interchangeably.)
- T_s is a partial transition function between service states ($\subseteq S_s \times \Sigma_s \times S_s$).
- $Time_s$ is a partial transition time function of primitives. Each primitive has a unique execution time. Transition time is non-negative.
- $\sigma_s \in S_s$ is an initial service state.

A service specification can be represented by a labeled directed graph. In the graph, a node represents a state in S_s, an edge represents a transition in T_s, and labels attached to each edge represent a primitive in Σ_s, and labels surrounded by parentheses under the primitives represent transition times.

If there is a transition (let it be $t \in \Sigma_s$) from a state u to another state v, then v is said to be adjacent state of u, and if there is a directed path from a state u to another state v, then v is said to be reachable from u. For any $(u, p, v) \in T_s$, state u is called an origin state of primitive p, and it is denoted by u = origin(p). A sequence of transitions $(u_1, p_1, u_2), (u_2, p_2, u_3), \ldots, (u_n, p_n, u_{n+1})$ from u_1 to u_{n+1} in a service specification implies or defines an execution order of the primitives $p_1, p_2, \ldots, p_n$. If $(u, p, v) \in T_s$, and u is reachable from the initial state, then the primitive p is said to be executable in the service specification.

We assume that priorities are assigned to primitives. Priorities are used in Section 3.2 to avoid unspecified receptions due to the message collisions. For comparison between priorities, we introduce the function pri(). Function pri(p) represents the value of priority primitive p has, and pri(p) > pri(q) represents priority of primitive p is higher than that of primitive q.

Definition 2

If there exist two transitions (u, p, v) and (u', p', v') such that u=u' and sap(p)=sap(p'), then state u (u') is called a *choice state*. If there exist two transitions (u, p, v) and (u', p', v') such that u=u' and sap(p)$\neq$ sap(p'), then state u (u') is called a *parallel state*.

If there exist two transitions (u, p, v) and (u', p', v') such that v=v', a state v (v') is called a *join state* and a state which has no outgoing transition is called a *final state*.

In a sequence of transitions (u_1, p_1, u_2), $(u_2, p_2, u_3), \ldots, (u_{n-1}, p_{n-1}, u_n)$, (u_n, p_n, u_{n+1}) from u_1 to u_{n+1}, if sap(p_i) = j ($1 \leq i \leq n-1$) and sap(p_n)

$\neq j$, then state u_n is called a *stop state* from u_1 with SAPj.

An example of the service specification is shown in Figure 2. In this figure, an oval represents a service state, an arrow represents a transition between states. The state drawn by bold oval is an initial state.

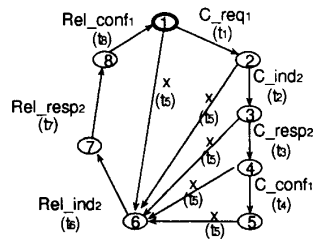


Figure 2 Example of a service specification with execution time (x = Rel_req1)

At state 1 for example, primitives C_req1 and Rel_req1 are executable, but one of two primitives is chosen since state 1 is a choice state. If primitive C_req1 is chosen, then primitive C_req1 is delivered from user 1 to process 1 through SAP1 and state 1 is changed to state 2. At state 3, primitives Rel_req1 and C_resp2 can be concurrently executed since state 3 is a parallel state. That is, primitive Rel_req1 is delivered from user 1 to process 1 through SAP1 and primitive C_resp2 is delivered from user 2 to process 2 through SAP2.

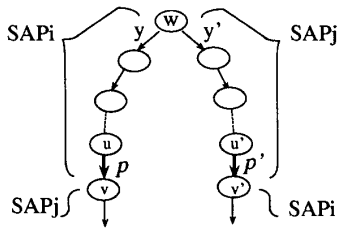


Figure 3 Conditions for finding MC primitives

Definition 3

For a given service specification $< S_s, \Sigma_s, T_s, \sigma_s >$, if there exist two transitions (u, p, v) and $(u', p', v')(\in T_s)$ satisfying the following conditions C1 through C3 (See Figure 3), then these two transitions are said to have a possibility of *Message Collision* (MC). Additionally, we simply call two primitives p and p' as MC primitives.

Condition C1 : There exists a parallel state (let it be w in Figure 3), from which two transitions labeled by $y = y_i$ and $y' = y'_j (i \neq j)$ exist.

Condition C2 : States v and v' are stop states from state w with SAPi and SAPj, respectively. (This also means sap(p)=i and sap(p')=j.)

Condition C3 : Both states u and u' are reachable from w.

Condition C1 means that primitives y and y' are concurrently executable if w is reachable from the initial state. Condition C2 means that both primitives p and p' potentially cause a message transmission from process i to process j and a message transmission from process j to process i, respectively. Condition C3 means that after primitives y and y' are delivered from their users through SAPi and SAPj, primitives p and p' are delivered through SAPi and SAPj, respectively. Thus, Conditions C1 through C3 mean that primitives p and p' can be concurrently executed. Note that primitives y and y' are not MC primitives since y and y' do not satisfy Condition C2. C_resp2 and Rel_req1 at state 3 in Figure 2 are examples of MC primitives.

2.3 Protocol Specification

Definition 4

A protocol specification is defined as a five-tuple $< (S_{1p}, S_{2p}), (\Sigma_{1p}, \Sigma_{2p}), (T_{1p}, T_{2p}), (Time_{1p}, Time_{2p}), (\sigma_{1p}, \sigma_{2p}) >$ where

- S_{1p} and S_{2p} are non-empty finite sets of protocol states (or simply states).
- Σ_{1p} and Σ_{2p} are finite sets of protocol messages (or simply messages) and service primitives of Definition 1. The protocol messages !x and ?x in Σ_{1p} and Σ_{2p} represent sending a message x and receiving a message x, respectively.
- T_{1p} ($\subseteq S_{1p} \times \Sigma_{1p} \times S_{1p}$) and T_{2p} ($\subseteq S_{2p} \times \Sigma_{2p} \times S_{2p}$) are partial transition functions between protocol states.
- $Time_{1p}$ and $Time_{2p}$ are transition time functions of messages and primitives in Σ_{1p} and Σ_{2p}, respectively.
- σ_{1p} ($\in S_{1p}$) and σ_{2p} ($\in S_{2p}$) are the initial protocol states.

Based on the protocol specification, the protocol specifications for process 1 and process 2 are

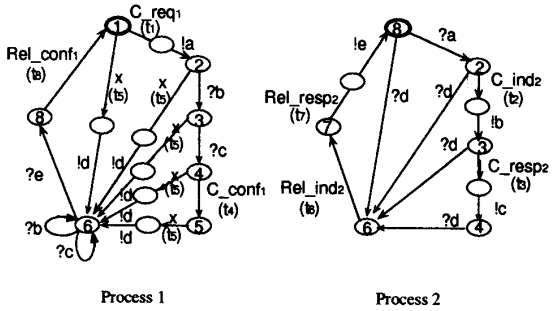


Figure 4 Example of a protocol specification
(x = Rel_req1)

defined by $< S_{1p}, \Sigma_{1p}, T_{1p}, Time_{1p}, \sigma_{1p} >$ and $< S_{2p}, \Sigma_{2p}, T_{2p}, Time_{2p}, \sigma_{2p} >$, respectively.

We assume that communication links between two processes are modeled by two FIFO queues: one is from process 1 to process 2 and the other is from process 2 to process 1. At the initial states of processes 1 and 2, the FIFO queues are empty. If process 1 sends a protocol message x to process 2 according to T_{1p}, then x is added into the bottom of the FIFO queue from process 1 to process 2. When x is on the top of FIFO queue from process 1 to process 2, and $(u,?x,v)$ is in T_{2p} for a current state u of process 2, then we say x can be received at the state u. Next, if process 2 receives a message x from process 1, then x is deleted from the top of FIFO queue from process 1 to process 2.

A protocol specification can also be represented by a labeled directed graph similar to the service specification. An example of the protocol specification is shown in Figure 4. In this figure, an oval represents a process state, and an arrow represents a transition. Each label of an arrow represents a protocol message or a service primitive. In process 1 and process 2, both states drawn by bold oval are initial states.

Definition 5

Consider a case that message x sent by process 1 is on the top of FIFO queue from process 1 to process 2, and a current state of process 2 is state u. If there does not exist any $(u',?x,v)$ in T_{2p} for any $v \in S_{2p}$ and any paths from state u to state u' which only includes (a,l,b)s in T_{2p} where l is a primitive or transmission of a message, then we say that an unspecified reception with respect to x occurs at process 2. Similarly, the unspecified reception at process 1 is defined by changing process 1, process 2, T_{2p} and S_{2p} to process 2, process 1, T_{1p} and S_{1p}, respectively.

2.4 Classes of Service Specifications

This paper focuses on a class of service specifications which include MC primitives because, based on the MC primitives, protocol specifications with message collisions are generated. On the other hand, this paper imposes the following restrictions R1 and R2 to exclude unnecessary descriptions for service specifications and restriction R3 to assure a finite realtime bound of execution times for sequences of primitives and messages from an initial state to the initial state or a final state.

Restriction R1 : Each state from which more than one transision leaves is either a choice state or a parallel state.

Restriction R2 : For any parallel state u, there exists a stop state from u with SAP1 or SAP2 in all sequences of transitions from u.

Restriction R3 : There is no sequence of transitions (u_1,p_1,u_2), (u_2,p_2,u_3), ..., (u_n,p_n,u_{n+1}) such that $u_{n+1} = u_1$ for some $n(\geq 1)$ and u_i is not an initial state for any i $(1 \leq i \leq n)$.

R1 divides states which have more than two outgoing transitions into choice states and parallel states, at which either one of primitives is selected or all primitives are simultaneously executed, and excludes states which share choice states and parallel states. R2 requests two successive primitives passing through different SAPs in each of two sequences of primitives, which starts from a parallel state. If such successive primitives do not exist, then message collisions do not occur. Note that service specifications satisfying Conditions C1 through C3 in Definition 3 are included in service specifications restricted by R1 and R2.

Protocol Synthesis problem (called PS problem) to be solved in this paper is formally defined as follows:

Input : A service specification $\mathbf{S}=<S_s,\Sigma_s,T_s,\sigma_s>$ with Restrictions R1 and R2, and priorities assigned to all primitives.

Output : A protocol specification which satisfies Conditions P1 and P2 as follows:

Condition P1 : Execution order of primitives given in the service specification **S** is kept in the protocol specification **P**.

Condition P2 : No unspecified reception caused by message collisions occurs in the protocol.

The previous protocol synthesis methods[1, 2, 8] could not assure Condition P2, if a service specification with MC primitives is given. That is, a protocol

specification derived from the service specification includes unspecified receptions.

Timeliness Verification problem (called TV problem) to be solved in this paper is formally defined as follows:

Input : A service specification $\mathbf{S}=< S_s, \Sigma_s, T_s, Time_s, \sigma_s >$, a protocol specification synthesized by the proposed method which solves the PS problem, transfer times to deliver a message between two processes, and a realtime bound of the protocol specification.

Output : Whether all execution times for sequences of transitions in the synthesized protocol specification are within the given realtime bound (yes/no) and the maximum execution time for all the sequences.

If yes is outputted, then it is guaranteed in the synthesized protocol specification that execution time for any sequence of transitions is not beyond the given realtime bound.

Realtime bound and transfer times to deliver a message between two processes are given by protocol designers.

3 Protocol Synthesis Method

3.1 Overall

The proposed method to derive a protocol specification **P** consists of the following four steps.

Step1 : In order to avoid unspecified receptions due to message collisions, add some transitions to T_s in a service specification **S** by using priorities assigned to primitives.

Step2 : Obtain SAP1 and SAP2 service specifications by applying projection to a service specification refined at Step 1.

Step3 : Construct a protocol specification by applying transition synthesis rules (to be shown in Table 1) to SAP1 and SAP2 service specifications.

Step4 : Remove ε transitions from the protocol specification.

The details of the proposed method are explained in the following sections.

3.2 Step1

As shown in the input of the PS problem, priorities are assigned to all primitives. Based on this, priorities are assigned to primitives in each sequence.

Assignment of priorities to primitives in each sequence is performed as follows.

(1) Set current state to be a parallel state which is reachable from an initial state. (2) To primitives in each sequence from the current state to the closest stop state, initial state or final state, reassign the maximum priority among priorities assigned to all the primitives in the sequence. (3) Repeat (1) and (2) for all parallel states in a breadth first search order.

The real protocol specifications are usually designed to deal with message collisions, such that a primitive which is followed by some sequences is executed prior to other primitives which are not followed. For example, primitives with respect to "release of communication path" are given higher priority than those with respect to "connection of communication path".

In Figure 1, a sequence Rel_req1, Rel_ind2, Rel_resp2 and Rel_conf1 represents "release of communication path" and a sequence C_req1, C_ind2, C_resp2 and C_conf1 represents "connection of communication path". Since Rel_req1 is given higher priority than C_resp2, when the message collision occurs in Figure 1, Rel_req1 is followed by Rel_ind2, Rel_resp2 and Rel_conf1, while C_resp2 is not followed by C_conf1.

Conditions for finding MC primitives are already given in Definition 3 and are explained using Figure 3. Step1 is explained using Figure 5. Figure 3 is a part of Figure 5. In this figure, p_i and p'_j $(i \neq j)$ are MC primitives. Consider two transitions (s, p''_j, t) which is followed by (t, r'', x) with $\mathrm{sap}(r'')=i$, and (s', p'''_i, t') which is followed by (t', r''', x') with $\mathrm{sap}(r''')=j$. State t represents a stop state from state v with SAPj and state t' represents a stop state from state v' with SAPi. States y and y' which appear in Case 2 are the same as t and t', respectively, except that they are the first stop states after the initial state.

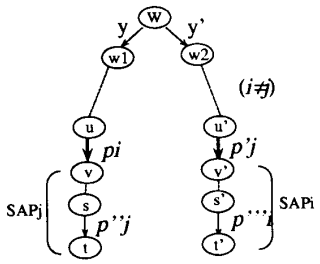


Figure 5 Explanation for Step1

Step1 is applied to each pair of MC primitives p_i and p'_j (See Figure 5), and application of this step is further divided into the following two cases, based on reassigned priorities of two primitives.

- Case 1 ($\mathrm{pri}(p_i) > \mathrm{pri}(p'_j)$. Note that there is no p_i in the path from w to u' because of Condition C2 of Definition 3.)(See Figure 5)

  For each state m such that m is in paths from v' to stop states t' and that there is no transitions (m, p_i, m') for any state m', insert a transition $(m, \mathrm{L}p_i, v)$. We say that $\mathrm{L}p_i$ is generated based on p_i. If there is a transition (m, p_i, m') for some state m', then no transition is inserted.

  For each state n such that n is in paths from v to stop states t and that there is no transitions (n, p'_j, n') for any state n', insert a transition $(n, \mathrm{L}p'_j, n)$. We say that $\mathrm{L}p'_j$ is generated based on p'_j. If there is a transition (n, p'_j, n') for some state n', then no transition is inserted.

- Case 2 ($\mathrm{pri}(p_i) = \mathrm{pri}(p'_j)$.)

  For each state $m1$ such that $m1$ is in paths from v' to stop states t' and that there is no transitions $(m1, p_i, m')$ for any state m', insert a transition $(m1, \mathrm{L}p_i, init)$ where $init$ is an initial state. We say that $\mathrm{L}p_i$ is generated based on p_i. If there is a transition $(m1, p_i, m')$ for some state m', then no transition is inserted.

  For each state $m1'$ such that $m1'$ is in paths from $w2$ to states u', delete a transition $(m1', \mathrm{L}p_i, init)$.

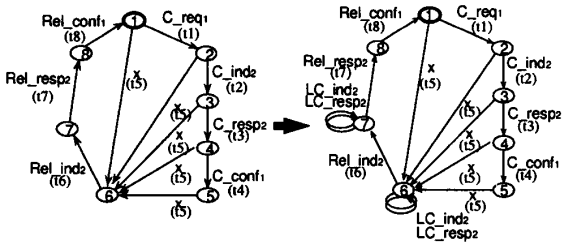
  For each state $m2$ such that $m2$ is in paths from $init$ to stop state y's from $init$ with SAPi and that there is no transitions $(m2, p_i, m2')$ for any state $m2'$, insert a transition $(m2, \mathrm{L}p_i, m2)$. We say that $\mathrm{L}p_i$ is generated based on p_i. If there is a transition $(m2, p_i, m2')$ for some state $m2'$, then no transition is inserted.

  For each state $n1$ such that $n1$ is in paths from v to stop states t, each state $n1'$ such that $n1'$ is in paths from $w1$ to states u, and each state $n2$ such that $n2$ is in paths from $init$ to stop states y from $init$ with SAPj, similar operations are performed as for $m1$, $m1'$ and $m2$.

Consider a service specification shown in Figure 2. In Figure 2, there are two pairs of MC primitives ($\mathrm{Rel\_req_1}$, $\mathrm{C\_ind_2}$) and ($\mathrm{Rel\_req_1}$, $\mathrm{C\_resp_2}$). The protocol designer assumes that $\mathrm{pri(Rel\_req_1)} > \mathrm{pri(C\_ind_2)}$ and $\mathrm{pri(Rel\_req_1)} > \mathrm{pri(C\_resp_2)}$. Then, Figure 6 shows a resultant service specification with two new transitions labeled by $\mathrm{LC\_resp_2}$ at states 6 and 7 and two new transitions labeled by $\mathrm{LC\_ind_1}$ at state 6 and 7, which are obtained by applying Case 1 of Step1.

3.3 Step2

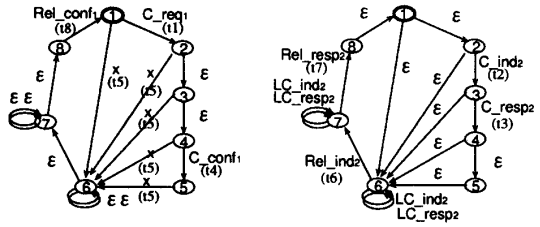
A SAP1 service specification is obtained from a service specification by substituting each transition associated with SAP2 for ε transition. Similarly, a SAP2 service specification is obtained by substituting each transition associated with SAP1 with ε transition.



Service specification in Figure 2 Service specification after Step1

Figure 6 Example of a service specification at Step1
(x = $\mathrm{Rel\_req_1}$)

As an example, consider a service specification shown in Figure 6. Then Figure 7 shows resultant SAP1 and SAP2 service specifications. In the SAP1 service specification, all primitives associated with SAP2 in the service specification such as C_ind2 and C_resp2 are substituted with ε. The primitives in the SAP2 service specification are also substituted in a similar way.



SAP1 service specification SAP2 service specification

Figure 7 Example of SAP1 and SAP2 service specifications
(x = $\mathrm{Rel\_req_1}$)

3.4 Step3

In this step, a protocol specification is obtained from SAP1 and SAP2 service specifications. This transformation is performed by applying transition synthesis rules shown in Table 1.

For a SAP1 service specification $< S_s, \Sigma_s, T_s, \sigma_s >$ and any $u \in S_s$, the function OUT(u) is defined as follows: OUT(u) = SAP2 if there exists at least one transition (u, ε, v) in T_s, otherwise OUT(u) = SAP1. The function OUT(u) for the SAP2 service specification is defined in a similar way.

In Table 1, E_i and $\mathrm{L}E_i (i = 1, 2)$ denote some primitives in the SAPi specification. Each pair of transition synthesis rules A.k and B.k ($1 \leq k \leq 4$) is applied to

Table 1 Transition synthesis rules

Transition Rule	Input	Condition	Output
A.1	Ei	s2 is not initial state, OUT(s2) = SAPi	Ei
B.1	ε		ε
A.2	Ei	s2 is not initial state, OUT(s2) ≠ SAPi	Ei le
B.2	ε		?e
A.3	Ei	s2 is initial state	Ei le
B.3	ε		?e
A.4	LEi		
B.4	ε		?e

a pair of transitions (s_1,E_i,s_2) and (s_1,ε,s_2), respectively. Message "e" is uniquely generated for each primitive E_i in Rules A.k and B.k with (k=2,3). In Rules A.4 and B.4, assuming that LE_i is generated based on primitive E_i in Step 1, message "e" that is uniquely generated for E_i is produced.

Conditions for A1 through A.3 are checked for each transition (s_1,E_i,s_2) to select exactly one transition rule. On the other hand, Rules A.4 and B.4 are applied to a pair of transitions (s_1,LE_i,s_2) and (s_1,ε,s_2), respectively. Rules A.4 and B.4 are applied for receiving a message caused by a message collision.

Consider SAP1 and SAP2 service specifications in Figure 7. Then, by applying transition synthesis rules to this input, protocol specification shown in Figure 8 is obtained. In Figure 8, messages "a", "b", "c", "d" and "e" are uniquely generated for C_req_1, C_ind_2 C_resp_2, Rel_req_1 and Rel_resp_2, respectively.

3.5 Step4

In this step, ε transitions are removed from the protocol specification by applying the ε removal algorithm in [3]. Figure 4 shows a protocol specification that is obtained by applying Step4 to Figure 8.

4 Verification of Timeliness

4.1 Outline of Verification

In order to solve the TV problem, in other words, to verify timeliness of the protocol specification synthesized by Steps 1 to 4, the following method is proposed. It consists of two procedures.

Procedure 1 : Obtain all sequences of transitions from an initial state to the initial state or to a final state included in the synthesized protocol specification.

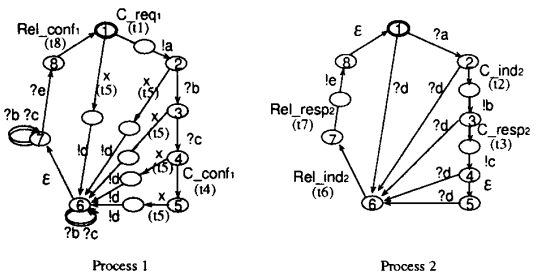


Figure 8 Example of a protocol specification after Step3 (x = Rel_req1)

Procedure 2 : Compute an execution time for each sequence obtained in Procedure 1.

The longest time among execution times for all the sequences is the realtime bound of the synthesized protocol specification.

4.2 Procedure 1

Since Procedure 1 depends on the proposed synthesis method, all the sequences of transitions can be efficiently obtained so that a message sent from a process can be received at states in at most three sequences for each parallel state and each choice state.

If parallel states and choice states exist in the service specification, then the number of sequences generated in the protocol specification is at most 3 x (the number of parallel states) x (the number of choice states) x (the maximum number of outgoing transitions from the choice states). This number is a polynomial with respect to the size of the protocol specification.

4.3 Procedure 2

As a subproblem of the TV problem, the timeliness problem to compute an execution time for each sequence of transitions in the synthesized protocol specification is formally defined as follows.

Input : A protocol specification synthesized by Step1 to Step4, a sequence of transitions obtained in Procedure 1, a transition time for each transition in the sequence, and transfer times to deliver a message between two processes.

Output : Execution time to execute all transitions in the sequence.

It will be shown that this problem can be reduced to the following task scheduling problem.

Input : A multiprocessor system, tasks, execution time of each task, and precedence relation on tasks.

Output : Completion time to execute all tasks.

First, a partial order relation on transitions which corresponds to the precedences relation on tasks is defined as follows: For a sequence of transitions (p_1, l_1, p_2),(p_2, l_2, p_3),...,(p_n, l_n, p_{n+1}), the following two relations are execution order constraints of transitions.

Relation 1 : After transition $l_h(1 \leq h \leq n-1)$ is executed, $l_i(2 \leq i \leq n,\ h < i)$ is executed where l_h and l_i are primitives and/or messages in the same process.

Relation 2 : After transition $l_j(1 \leq j \leq n-1)$ is executed by sending message m through channel C_{pq}, transitions $l_k(2 \leq k \leq n,\ j < k)$ is executed by receiving it.

Next, it is shown that how the timeliness problem is reduced to the task scheduling problem for the multiprocessor systems.

By mapping processes (including channels), transitions and the partial order relation on transitions into processors, tasks and the precedence relation on tasks, the reduction is performed. Therefore, execution time of the sequence of transitions are computed by applying well known task scheduling algorithms to the timeliness problem.

5 Conclusion

This paper has proposed two methods for efficient design of communication protocols. One is an automated synthesis method of a protocol specification with message collisions from a given service specification. The other is a timeliness verification method based on the result of the synthesis method. The characteristics of the synthesis method include that no unspecified receptions are caused by message collisions and those of the verification method include that the verification problem of the synthesized protocol specification is reduced to the task scheduling problem for the multiprocessor system. Therefore, more reliable protocol specifications are efficiently produced by this method than those by the previous synthesis methods [1, 2, 8].

A protocol synthesis system based on the proposed synthesis method has been implemented. Some experiment conducted on the system shows that time for synthesizing a protocol specification with message collisions from a service specification is almost proportional to the size of service specification including parallel states. An extension of the protocol synthesis method to n ($\geq$2) processes is studied in [6].

References

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

[2] Chu, P. M. and Liu, M. T., "Synthesizing protocol specifications from service specifications in the FSM model," *Proc. Computer Networking Symp.* , pp.173-182, April 1988.

[3] Hopcroft, J. E. and Ullman, J. D., "Introduction to Automata Theory, Language, and Computation," Chapter 3, Addision-Wesley, 1979.

[4] Igarashi, H., Kakuda, Y. and Kikuno, T., "Synthesis of protocol specifications for design of responsive protocols," *IEICE Trans. on Information and Systems*, pp.1375-1385, Nov. 1993.

[5] Kakuda, Y. and Kikuno, T., "Issues in responsive protocols design," *Proc. of the Second International Workshop on Responsive Computer Systems*, Oct. 1992, Dependable Computing and Fault-Tolerant Systems, 7, pp.17-26, Springer-Verlag, 1993.

[6] Kakuda, Y., Nakamura, M. and Kikuno, T., "Automated synthesis of protocol specifications from service specifications with parallelly executable multiple primitives," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, Oct 1994 to appear.

[7] Liu, M. T.,"Protocol engineering," Advances in Computers, 29, pp.79-195, 1989.

[8] Saleh, K.,"Automatic synthesis of protocol specifications from service specifications," *Proc. Int'l. Phoenix Conference on Computers and Communications*, pp.615-621, March 1991.

[9] Saleh, K. and Probert, R. L., "Synthesis of communication protocols: Survey and assessment," *IEEE Trans. on Computers*, Vol.40, No.4, pp.468-475, April 1991.