システムソフトウェアと
オペレーティング・システム　60－17
マルチメディア通信と分散処理　61－17
（１９９３．　７．　９）

メッセージ衝突を含む２プロセスのプロトコル仕様の自動合成

五十嵐 裕孝　浅田 宏幸　角田良明　菊野 亨

大阪大学 基礎工学部 情報工学科

あらまし：

プロトコル合成とは，サービス仕様からプロトコル仕様を導出することである．従来，提案された方法ではサービス仕様中に，異なったサービスアクセスポイントでプリミティブを上位層から下位層へ同時に送信する記述が含まれている場合，導出されるプロトコル仕様にメッセージ衝突による未定義受信が含まれるという欠点があった．また，従来の方法では，導出されたプロトコル仕様のいかなる遷移系列を実行するときの時間がリアルタイム通信のために指定されている上限値を越えるかを効率良く判定することができなかった．

本論文ではまず，上述した性質を有するサービス仕様から２プロセスのプロトコル仕様を導出する自動合成法を提案する．この方法で合成されたプロトコル仕様には，メッセージ衝突に起因する未定義受信が含まれないと言った特徴がある．次に，その合成されたプロトコル仕様がリアルタイム性を満たしているかどうかを効率良く判定する方法を提案する．提案する検証法では，プロトコル仕様のなかの全ての遷移系列を多項式時間で導出する．このため，リアルタイム性の判定を効率良く行なうことができる．

Automated Synthesis of Two Process Protocol Specifications with Message Collisions

Hirotaka IGARASHI, Hiroyuki ASADA, Yoshiaki KAKUDA and Tohru KIKUNO

Dept. of Information and Computer Sciences
Faculty of Engineering Science
Osaka University, Toyonaka
Osaka 560, Japan

Abstract:

Protocol synthesis is to derive a protocol specification based on a service specification. In the previous methods, if the service specification includes simultaneous transmission of primitives from a high layer to a low layer through different service access points, then the derived protocol specification includes protocol errors of unspecified reception caused by message collisions. Furthermore, the previous protocol synthesis methods do not include efficient procedures to verify whether execution times along any acyclic sequence of transitions in the synthesized protocol specifications are no more than a specified bound for realtime communication.

This paper proposes a method for automated synthesis of a protocol specification with two processes from a service specification described above. Protocol specifications derived from the service specifications by the proposed synthesis method are free from protocol errors of unspecified receptions due to message collisions. This paper also proposes an efficient verification method for determining a realtime bound in the synthesized protocol specification. In this method, all sequences of transitions in the protocol specification are enumerated in a polynomial time. This scheme enables the efficient verification of the realtime bound.

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, 6, 7, 8] 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.

The principle of protocol synthesis is that a protocol specification that defines relations on messages between processes in the low layer is derived from a given service specification that defines relations on service primitives between users in a high layer and processes in a low layer. 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 [7] 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 by sending messages from process 1 to process 2 and from process 2 to process 1, then a message collision occurs between processes 1 and 2. It causes protocol errors called unspecified receptions.

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

Furthermore, the previous protocol synthesis methods do not include efficient procedures to verify whether execution times along any acyclic sequence of transitions in the synthesized protocol specifications are no more than a specified bound for realtime communication.

To cope with these drawbacks of the previous synthesis methods, this paper first proposes an automated synthesis method of a protocol specification which is free from protocol errors of unspecified receptions caused by message collisions and next proposes an efficient verification method for determining whether execution times along any sequence in the synthesized protocol specification are not beyond a given realtime bound. In this method, all sequences transitions in the protocol specification are enumerated in a polynomial time. This scheme 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 discussed in this paper. In Section 3, a protocol synthesis method is proposed, and in Section 4 details of the proposed methods and correctness of the proposed synthesis methods are described. In Section 5, a verification method for determining a realtime bound in the synthesized protocol specification is proposed. Finally Section 6 concludes the paper with future researches.

2 Definitions

2.1 Service Specification and Protocol 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) $<S_s,\Sigma_s,T_s,Time_s,\sigma>$ 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 two attributes: One is direction dir(p) along which primitive is delivered. dir(p)=↓ if p is delivered from a user to process, and dir(p)=↑ if p is delivered from a process to a user. The other is service access point sap(p) through which primitive passes, that is, SAP1 or SAP2. Sap(p)=1 if p is delivered through SAP1, and sap(p)=2 if p is delivered through SAP2. Thus if primitive p with dir(p)=↓ and sap(p)=1, then p is denoted by $p_1\downarrow$. If p with dir(p)=↑ and sap(p)=1, then p is denoted by $p_1\uparrow$. Primitive $p_2\downarrow$ and $p_2\uparrow$ are defined in a similar way. (In the following, we use the notations p and $p_1\downarrow$, $p_1\uparrow$, $p_2\downarrow$, $p_2\uparrow$ 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.
- $\sigma \in S_s$ 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 represents a primitive in Σ_s and a transition time. The labels with $p\downarrow$ or $p\uparrow$ represents a primitive and the labels with t_i represents a transition time. If there is a directed path from a state (let it be $u \in S_s$)to another (let it be $v \in S_s$), 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),...,(u_n,p_1,u_{n+1}) from u_1 to u_{n+1} in a service specification implies or defines an execution order of the primitives $p_1,p_2,...,p_n$. 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.

Primitives which may cause message collisions are of type $\downarrow$ (that is, $p_1 \downarrow$ and $p_2 \downarrow$). We assume that priorities are assigned to such primitives. Priorities are used in Subsection 4.1 to avoid unspecified receptions due to the message collisions.

If there exist two transitions (u_1,p_1,v_1) and (u_2,p_2,v_2) such that u_1=u_2 and sap(p_1)=sap(p_2), then state u_1 (and thus u_2 also) is called a choice state. If there exist two transitions (u_3,p_3,v_3) and (u_4,p_4,v_4) such that u_3=u_4 and sap(p_3)$\neq$ sap(p_4), then state u_3 (and thus u_4 also) is called a parallel state.

For the explicit description for concurrent execution of primitives, this paper restricts the service specification as follows.

Restriction R1 : Any state can be at most one of choice state and parallel state, simultaneously. (This means that each state is a choice state, a parallel state or other.)

Restriction R2 : For any parallel state u_3 and any transitions (u_3,p_3,v_3),(u_3,p_4,v_4) from u_3, dir(p_3) = dir(p_4) = $\downarrow$.

Restriction R3 : For any parallel state u and each $i(i$=1,2), there is the first primitive p with dir(p)=$\downarrow$ and sap(p)=i whose origin state is reachable from u and which is followed by p' with sap(p')=$j(j \neq i)$, and neither primitives $r_j \downarrow$ nor $r_j \uparrow$ exist in any sequence of transitions from u to the origin state of p.

Restriction R4 : There do not exist three parallel states (let them be x, y and z) satisfying the following condition: Let two transitions which leave state x be t and t'. State y is reachable from x through t while state z is reachable from x through t'. However, y is not reachable from z and vice versa.

Restriction R5 : 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 $n(\geq 1)$ and u_1 is not an initial state.

R3 and R4 are necessary to easily find primitives which cause message collisions and to avoid unexpected concurrent execution of primitives, respectively. R5 is necessary for verification of realtime bound of a synthesized protocol from a given service specification. 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. Labels with $p \uparrow$ and $p \downarrow$ represent primitives while labels with t_i represent transition times. The state drawn by bold line is an initial state.

Projection (to be defined in Definition 2) is used for dividing a service specification into two service specifications with respect to SAP1 and SAP2. In the projection, service primitives that do not contribute to SAP1 and SAP2 are substituted with a primitive ε, respectively. The primitive ε is a null primitive that causes no message sending.

Definition 2

For a given service specification $<S_s,\Sigma_s,T_s,Time_s,\sigma>$, if there exists a new service specification $<S',\Sigma',T',Time',\sigma'>$ satisfying the following conditions (1) through (5), then the service specification $<S',\Sigma',T',Time',\sigma'>$ is called a projection with respect to SAP1 of service specification $<S_s,\Sigma_s,T_s,Time_s,\sigma>$.

(1) S'=S_s.

(2) $\Sigma' = \{p : p \in \Sigma_s, sap(p) = 1\} \cup \{ \varepsilon \}$.

(3) If $(u,p,v) \in T_s$ and $p = p_1\uparrow$ or $p_1\downarrow$, then $(u,p,v) \in T'$. On the other hand if $(u,p,v) \in T_s$ and $p = p_2\uparrow$ or $p_2\downarrow$, then $(u,\varepsilon,v) \in T'$.

(4) If $(u,p,v) \in T_s$ and sap(p) = 1,then $Time'(p) = Time_s(p)$. Otherwise $Time'(p)$ is undefined

(5) $\sigma' = \sigma$.

The projection with respect to SAP2 of a service specification is also defined in a similar way. The projection with respect to SAP1(SAP2) of a given service specification is simply called by SAP1(SAP2) service specification (of the given service specification).

Definition 3 For a given service specification $<S_s,\Sigma_s,T_s,\sigma>$, if there exist two transitions (u,p,v) and (u',p',v') ($\in T_s$) satisfying the following conditions C1 through C3 (See Figure 4), 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 : Two primitives p and p' are $p = p_i\downarrow$ and $p' = p'_j\downarrow$ $(i \neq j)$. Additionally there exists at least one transition (v,q,x) with sap(q)=j, and (v',q',x') with sap(q')=i.

Condition C2 : There exists a parallel state (let it be w in Figure 4), from which two transitions labeled by $y = y_i \downarrow$ and $y' = y'_j \downarrow$ $(i \neq j)$ leave.

Condition C3 : Both states u and u' of primitives are reachable from w. Additionally, no primitives q's with sap(q)=j and $q = q \downarrow$ exist between states w and u. Similarly no primitives r's with sap(r)=i and $r = r \downarrow$ exist between states w and u'.

Condition C1 means that both directions for primitives p and p' are from a user to a process, that potentially cause a message from process i to process j and a message from process j to process i, respectively. Condition C2 means that primitives y and y' are concurrently executable if w is reachable from the initial state. Condition C3 means that after primitives y and y' are delivered through SAP$_i$ and SAP$_j$, primitives p and p' are delivered through SAP$_i$ and SAP$_j$, respectively. Thus, these Conditions C1 through C3 mean 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 C1. C_resp2↓ and Rel_req1↓ at state 3 in Figure 2 are examples of MC primitives.

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. On the other hand, service primitives in Σ_{1p} and Σ_{2p} are characterized by service access points only. (Thus symbols ↓ and ↑ representing directions are omitted.)
- 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 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: the 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, 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 similar labeled directed graph as the service specification. An example of the protocol specification is shown in Figure 3. 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 line are initial states.

Definition 5

Consider a case that message x that is 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 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.

2.2 Protocol Synthesis Problem and Realtimeliness Verification Problem

This paper proposes two methods for solving the following problems: Protocol Synthesis Problem and Realtimeliness Verification Problem.

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

Input : A service specification S=$<S_s,\Sigma_s, T_s, Time_s, \sigma>$ with Restrictions R1, R2, R3, R4 and R5, and priorities assigned to primitives p's with dir(p)=↓.

Output : A protocol specification which satisfies Conditions P1 and P2.

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, 7] 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.

The outline of proof that a protocol specification synthesized by the proposed method satisfies Conditions P1 and P2 is described in Section 4.2.

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

Input : A service specification S=$<S_s,\Sigma_s, T_s,Time_s,\sigma>$, 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 prptocol 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.

3 Protocol Synthesis Method

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 in Definition 2 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 [4].

4 Details of Synthesis Method

4.1 Refinement of Service Specification

In the previous methods[1, 2, 7], it is assumed that MC primitives never exist in the service specification. If MC primitives are included in the service specification, message collisions may occur in a sequence of the protocol specification, and thus they cause unspecified receptions.

A concrete method to add some transitions to the service specification S in Step1 is described in this subsection. Though some transitions are added to the service specification S, execution order of primitives in the service specification is kept in the derived protocol specification **P**. Since priorities are necessary when adding the transitions, we assume that priorities are given by protocol designers for primitives which may cause message collisions.

As shown in the input of the PS problem, priorities are assigned to primitives p's with dir(p)=↓. Based on this, priorities must be assigned to primitives in each sequence. Before describing this assignment, some definitions are necessary. Choice states and parallel state are called branch states, a state which has more than one incoming transition is called join state and a state which has no outgoing transition is called final state.

Assignment of priorities to primitives in each sequence is performed as follows. (1) Set current state be a branch state which is reachable from an initial state. (2) To primitives in each sequence from the current state to most close another branch states, an initial state, a final state or a join states, assign the priority of the maximum of primitive p with dir(p)=↓ in that sequence. (3) Compare the maximum priority assigned to primitives in sequences entering the current state and maximum priority assigned to primitives in sequences leaving the current state, and reassign the larger one to primitives in sequences to which is smaller are assigned. (4) Repeat (1), (2) and (3) for all branch 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 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 4. Step1 is explained using Figure 4. In Figure 4, $p_i\downarrow$ and $p'_j\downarrow (i \neq j)$ are MC primitives. There exist two transitions $(s, p''_j \downarrow, t)$ which is followed by (t, r'', x) with sap(r'')=i, and $(s', p'''_i \downarrow, t')$ which is followed by (t', r''', x') with sap(r''')=j. The primitive $p''_j\downarrow$ represents the first primitive p'' such that sap(p'')=j and dir(p'')=↓ and that appears after $p_i\downarrow$. Similarly, the primitive $p'''_i\downarrow$ represents the first primitive p''' such that sap(p''')=i and dir(p''')=↓ and that appears after $p'_j\downarrow$. Primitives q'' and q'''s' attribute are the same to those of p'' and p''', respectively, but they are the first primitives after the initial state.

Step1 is applied to each pair of MC primitives $p_i\downarrow$ and $p'_j\downarrow$ (See Figure 4), and application of this step is further divided into the following two cases, based on redefined priorities of two primitives.

- Case 1 (Priority of $p_i\downarrow$ is higher than that of $p'_j\downarrow$. Note that there is no $p_i\downarrow$ in the path from w to u' because of Condition C3 of Definition 3.)(See Figure 4)

  For each state m only such that m is in paths from v' to origin($p'''_i\downarrow$)s and that there is no transitions $(m, p_i\downarrow, m')$ for any state m', insert a transition (m, L_i, v). We say that L_i is generated based on $p_i\downarrow$. If there is a transition $(m, p_i\downarrow, m')$ for some state m', then no transition is inserted.

  For each state n such that n is in paths from v to origin($p''_j\downarrow$)s and that there is no transitions $(n, p'_j\downarrow, n')$ for any state n', insert a transition (n, L_j, n). We say that L_j is generated based on $p'_j\downarrow$. If there is a transition $(n, p'_j\downarrow, n')$ for some state n', then no transition is inserted.

- Case 2 (Priorities of $p_i\downarrow$ and $p'_j\downarrow$ are the same.)

  For each state $m1$ only such that $m1$ is in paths from v' to origin($p'''_i\downarrow$)s and that there is no transitions $(m1, p_i\downarrow, m')$ for any state m', insert a transition $(m1, \mathrm{L}_i, init)$ where $init$ is an initial state. We say that L_i is generated based on $p_i\downarrow$. If there is a transition $(m1, p_i\downarrow, m')$ for some state m', then no transition is inserted.

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

  For each state $n1$ such that $n1$ is in paths from v to origin($p''_j\downarrow$)s and that there is no transitions $(n1, p'_j\downarrow, n')$ for any state n', insert a transition $(n1, \mathrm{L}_j, init)$. We say that L_j is generated based on $p_i\downarrow$. If there is a transition $(n1, p'_j\downarrow, n')$ for some state n', then no transition is inserted.

  For each state $n2$ such that $n2$ is in paths from $init$ to origin($q''_j\downarrow$)s and that there is no transitions $(m2, p'_j\downarrow, m2')$ for any state $m2'$, insert a transition $(m2, \mathrm{L}_j, m2)$. We say that L_j is generated based on $p'_j\downarrow$. If there is a transition $(m2, p'_j\downarrow, m2')$ for some state $m2'$, then no transition is inserted.

For the limit of pages, explanation of Step2, Step3 and Step4 are ommited. Refer to [3] for their details.

4.2 Correctness of Proposed Synthesis Method

It is shown here that no unspecified receptions caused by message collisions exist in the synthesized protocol specification. After Case 1 of Step1 is applied to the service specification with MC primitives in Figure 4, it is divided into SAPi and SAPj service specification in at Step2. By applying transition synthesis rules in Table 1 and removing ε transitions at Step4, they are finally transformed into a protocol specification in Figure 5.

As shown in Figure 5, there are three execution sequences of primitives and messages as shown in the following **Case 1** through **Case 3**, since at state w in processes 1 and 2 only y_i and/or y'_j can be executed.

Case 1 : Process i executes y_i and p_i, and sends message c. Process j receives message c and there is no message collision.

Case 2 : Process j executes y'_j and p'_j, and sends message d. Process i receives message d and there is no message collision.

Case 3 : Process i executes y_i and p_i and process j executes y'_j and p'_j, concurrently. As a result, a message collision by messages c and d occurs. However, there are not unspecified receptions because they are surely received by transitions labeled by ?c and ?d as shown in Figure 5. Priorities of primitives are reassigned to the maximum of each sequence, so execution sequences of each process are always coordinated to the primitives' sequences with higher priorities.

As in the **Case 3**, there are no unspecified reception and execution order are kept in the protocol specification. Thus Conditions P1 and P2 are satisfied.

5 Verification of Realtimeliness

5.1 Outline of Verification

In order to solve the RV problem, in other words, to verify realtimeliness 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.

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.

5.2 Procedure 1

In general, the number of all sequences of transitions from an initial state to the initial state or a final state grows exponentially with respect to the size of an arbitrary protocol specification because a message sent from a process can be received at states in many sequences in another process as shown in Figure 6.

However, 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 two sequences as shown in Section 4.2. For example, in the protocol specification in Figure 5, three corresponding sequences are obtained (see **Case 1, 2** and **3**). In general, for each parallel state three sequence are generated.

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

5.3 Procedure 2

As a subproblem of the RV problem, the realtimeliness 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 into 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 of 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 realtimeliness problem is reduced to the task scheduling problem.

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 realtimeliness problem.

An example of this result is shown below. For a sequence of transitions depicted by bold arrows in Figure 3, a sequence chart representing relations among transitions and times they take is drawn in Figure 7. In this example, the execution time is $t_1 + t_{12}(a) + t_5 + t_{12}(c) + t_6 + t_7 + t_{21}(d) + t_8$.

6 Conclusions

This paper has proposed two methods. One is a synthesis method of a protocol specification from a given service specification for design of dependable and realtime protocols. The other is a verification method based on the above result. The characteristics of the synthesis method include that no unspecified receptions are caused by message collisions and those of the verification method are that the verification problem of the synthesized protocol specification is reduced to the task scheduling problem and sequences of transitions are enumerated in a polynomial time for verification based on the synthesis method.

Therefore, more dependable and realtime protocol specifications are efficiently produced by this method than those by the previous synthesis methods [1, 2, 7]. An extension of the protocol method to n (≥2) processes is now being studied.

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] Igarashi, H., Kakuda, Y. and Kikuno, T., "Synthesis of protocol specifications for design of responsive protocols," to appear in IEICE Trans. on Information and Systems, Nov. 1993.

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

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

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

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

[8] 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.

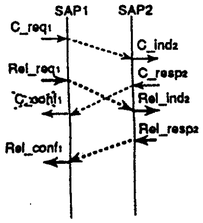

Figure 1 Sequence chart with a message collision

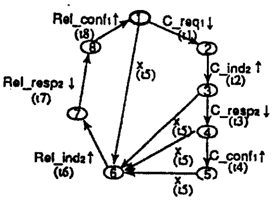

Figure 2 Example of a service specification
(x = Rel_req1↓)

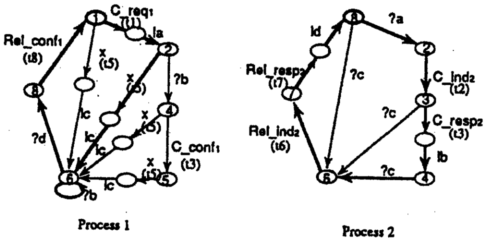

Figure 3 Example of a protocol specification
(x = Rel_req1)

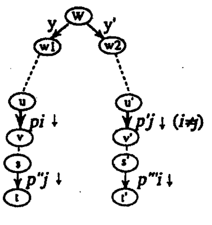

Figure 4 Explanation for Step1

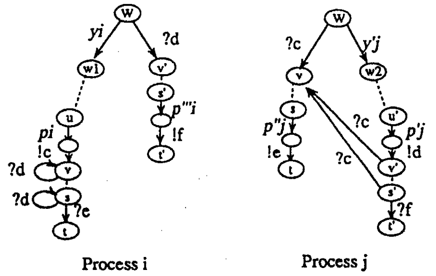

Figure 5 Explanation of correctness of synthesis

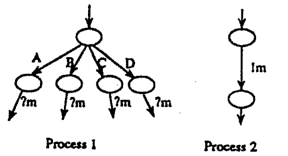

Figure 6 Example of sequences in arbitrary protocol specification

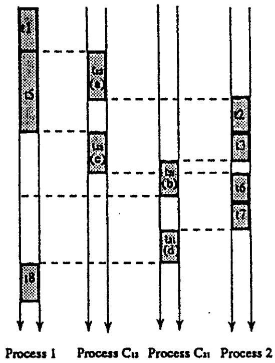

Figure 7 Example execution sequence

Transition Rule	Input	Condition	Output
A.1	$s1 \xrightarrow{EI} s2$	s2 is not initial state, OUT(s2)* = SAPi and dir(EI) = ↓	$s1 \xrightarrow{EI} s2$
B.1	$s1 \xrightarrow{\epsilon} s2$		$s1 \xrightarrow{\epsilon} s2$
A.2	$s1 \xrightarrow{EI} s2$	s2 is not initial state, OUT(s2) ≠ SAPi and dir(EI) = ↓	$s1 \xrightarrow{EI} s3 \xrightarrow{Ie} s2$
B.2	$s1 \xrightarrow{\epsilon} s2$		$s1 \xrightarrow{?e} s2$
A.3	$s1 \xrightarrow{EI} s2$	s2 is initial state and dir(EI) = ↓	$s1 \xrightarrow{EI} s3 \xrightarrow{Ie} s2$
B.3	$s1 \xrightarrow{\epsilon} s2$		$s1 \xrightarrow{?e} s2$
A.4	$s1 \xrightarrow{EI} s2$	dir(EI) = ↑	$s1 \xrightarrow{EI} s2$
B.4	$s1 \xrightarrow{\epsilon} s2$		$s1 \xrightarrow{\epsilon} s2$
A.5	$s1 \xrightarrow{U} s2$		$s1$ $s2$
B.5	$s1 \xrightarrow{\epsilon} s2$		$s1 \xrightarrow{?e} s2$

* OUT(s2) is a function, it is defined as follows.
In SAPi service specification ,

$$OUT(s2) = \begin{cases} SAPj(j \neq i) & \text{:if there exists at least one transition } (s2, \epsilon, v). \\ SAPi & \text{:otherwise.} \end{cases}$$

Table 1 Transition synthesis rules