An Integration-Oriented Approach for Designing Communication Protocols from Component-Based Service Specifications

M. Nakamura, Y. Kakuda, T. Kikuno

Reprint

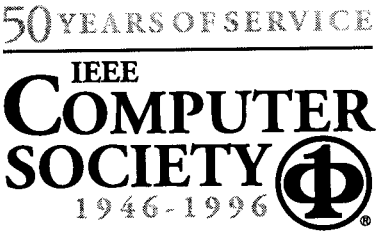
from

Proceedings INFOCOM '96

Fifteenth Annual Joint Conference of the IEEE

Computer and Communications Societies

San Francisco, California
March 24-28, 1996



Washington ◆ Los Alamitos ◆ Brussels ◆ Tokyo

PUBLICATIONS OFFICE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1314 USA

An Integration-Oriented Approach for Designing Communication Protocols from Component-Based Service Specifications

Masahide Nakamura, Yoshiaki Kakuda and Tohru Kikuno

Department of Information and Computer Sciences
Faculty of Engineering Science, Osaka University
Machikaneyama 1-3, Toyonaka City, Osaka 560, Japan

Abstract

A large and complex protocol is constructed by integrating components, each of which corresponds to subfunction specified in a service specification. The conventional approach to this construction is to integrate components on the protocol level using the existing protocol integration methods. In this approach, the reachability analysis of protocol components is required in the integration stage. So if the size of components becomes large, the integration stage would be a bottleneck because of the state explosion problem of the reachability analysis.

Therefore, we propose a new approach to construct the target protocol which at first integrates components on the service specification level and then transforms an integrated service specification into the target protocol by protocol synthesis technique. As the result, the construction of the target protocol from component service specifications can be efficiently executed in small state space without paying special attentions to the timing of protocol messages.

1 Introduction

The trend toward the enrichment of communication services in ISDN and IN has greatly increased the size and complexity of communication protocols which realize the services. In order to facilitate the design of such protocol specifications, the "component integration approach" is one of the most promising ones. The component integration approach involves the following three steps:
(1) Divide the functionality of the required service into subfunctions,
(2) Develop service specifications for the subfunctions as components (we call them *component service specifications*), and
(3) Obtain the target protocol (we call it *integrated protocol specification*) by the integration of subfunctions based on the component service specifications.

Major advantage of this approach is that we can develop each component with relatively smaller size and focus on single function without considering the interaction with other functions. The third step of deriving the integrated protocol specification from the component service specifications is the most difficult and interesting problem. Figure 1 shows the third step of derivation process schematically.

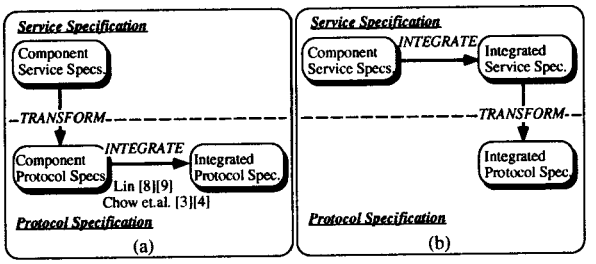


Figure 1: Derivation of integrated protocol specifications

Two approaches can be considered to obtain the integrated protocol specification from the component service specifications. Figure 1(a) shows the first and conventional approach: After transforming each component service specification into a *component protocol specification* one-by-one by applying conventional design and analysis techniques, we integrate them into single protocol specification. In this approach, we can utilize the existing methods for the integration of component protocol specifications. Several component integration methods on the protocol specification level have been proposed. Chow et.al. [3, 4] proposed a method for constructing *multiphase protocol*, which sequentially executes multiphases of behavior performing a distinct subfunction in each phase. Lin proposed two methods for integrating the component protocol specifications. One is for *alternating function protocol* [8] such that the user can select any one from multiple functions, but is restricted to execute only one function at a time. Another is for *concurrent function protocol* [9] which performs multiple functions concurrently. All of these methods require the validation of component protocols based on the reachability analysis to ensure the safety of the integrated protocol

specification. Unfortunately, if the analyzed protocol becomes large, it is known that the reachability analysis exponentially takes a lot of time and cost because of state explosion problem[1, 7]. Thus, if the component protocol specifications become large, then the integration of the components would be bottleneck of the first approach.

On the other hand, Figure 1(b) shows another approach: At first, we integrate the component service specifications into single *integrated service specification*, then transform it to the integrated protocol specification. In this approach, the component integration is carried out on the service specification level which has much smaller state space and is more abstract than protocol specification level . Even if some analyses of components are required in the integration of the service specifications, it is much easier than that on the protocol specification level. However in this approach, since the integrated service specification becomes large, it is difficult to transform the integrated service specification into the integrated protocol specification. Therefore, an efficient and reliable procedure is required for the transformation.

In this paper, we try to implement the second approach and present three integration methods on the service specification level which correspond to the existing protocol integration methods proposed in [8, 3, 4]. Moreover, we use a *protocol synthesis* in the transformation from an integrated service specification to the integrated protocol specification. Protocol synthesis [2, 6, 11, 13] is one of the most reliable and efficient techniques that automatically derives a protocol specification from a service specification without specification errors.

Major advantages of the proposed method are summarized as follows:
(a) Since the integration of the components is carried out on the *service specification level*, we can operate it in small state space without paying special attentions to the timing of synchronizing messages (protocol messages).
(b) Since the sufficient conditions for ensuring correctness of the integrated protocol specification are presented in this paper for *component service specifications*, we can easily check the sufficiency of the conditions without complicated analysis.
(c) The *automated* transformation is applied for the integrated service specification to get the target protocol specification.

This paper is organized as follows. Section 2 gives definitions of service and protocol specifications and Section 3 formulates the protocol derivation problem to be discussed in this paper. In Section 4, the protocol synthesis method is proposed, and Section 5 discusses component integration methods on the service specifications level. Finally Section 6 concludes the paper with future researches.

2 Preliminaries

2.1 Communication Model

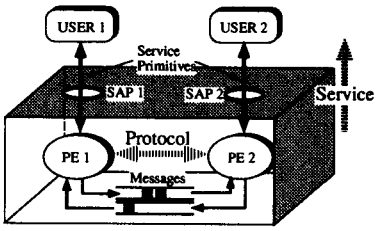


Figure 2: Communication architecture model

As shown in Figure 2, a communication service is specified by *service primitives* exchanged between users in the higher layer and processes in the lower layer through *service access points*(SAPs). The processes are also called protocol entities which are denoted by PEs in the following. A *service* is provided to Users by PEs through SAPs and is defined by *service specification*. A *protocol* is a rule that govern the exchange of the *protocol messages* among the PEs through the communication channel and is defined by *protocol specification*.

In this paper, we assume that the number of PEs is two, that the communication channel is reliable and that message is delivered in FIFO order.

2.2 Service Specification

A service specification defines sequences of primitives to be realized as communication services, which are exchanged between users and processes through SAP.

A *service specification* **S** is modeled by a *Finite State Machine*(FSM) and is represented by a directed graph, which includes two types of transitions. One is a *primitive transition* p, which has, as an attribute, an index of SAP through which p passes. If primitive p passes through SAPi ($i = 1, 2$), then we define a function $sap(p) = i$, and also represent it by p_i. Another is an *L transition* denoted by Lp_i. L transitions Lp_i's are the auxiliary transitions for the protocol synthesis procedure, which are translated into receptions of a message caused by execution of primitive p_i. For simplicity, a transition labeled by p from node u to w is denoted by (u, p, w) in the following discussion.

We assume that all service specification **S** are *deterministic*, that is, no two outgoing transitions from any node have identical labels.

A node of **S** is a *final node* iff there is no outgoing transition from it. A node of **S** is a *parallel node* iff more than one primitive transition through different SAPs is leaving from it.

For any path $\rho = (v, p, v')(v', q, v'') \ldots (w, r, w')$ in **S**, nodes v and w' are called *head node* of ρ and *tail node* of ρ, respectively, and primitive r is called *last primitive* of ρ.

A path of **S** from a node w is a *SAP1(or SAP2)-path* from w iff the path consists entirely of SAP1's (SAP2's) primitives. A SAP1(or SAP2)-path is a *SAP1(or SAP2)-cycle* iff its head node and tail node are identical. A SAP1(or SAP2)-path is a *reachable SAP1(or SAP2)-path* iff its tail node is final node.

A service specification **S** is *well-formed* iff **S** includes no parallel node or the following condition PL holds for any parallel node w in **S**.

Condition PL Consider a SAPi-path ρ and a SAPj-path μ from w $(i, j = 1, 2, i \neq j)$. Let P_i and Q_j be last primitives of ρ and μ, respectively. Then, (1) both μ and ρ form neither SAPk-cycle nor reachable SAPk-path $(k = 1, 2)$, (2) For any node v on ρ, an L transition (v, LQ_j, v) exists in **S**, and (3) For any node r on μ and the tail node t of ρ, an L transition (r, LP_i, t) exists in **S**.

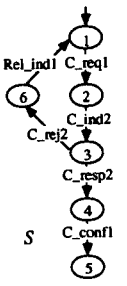


Figure 3: A service specification **S**

An example of **S** is shown in Figure 3. All of the transitions are primitive transitions, and node 5 is a final node. Since there is no parallel node (Note that node 3 is not a parallel node because both outgoing primitives pass through identical SAP2), **S** is well-formed. From node 1, there is SAP1-path (1,C_req1,2), and from node 2 there are two SAP2-paths (2,C_ind2,3)(3,C_rej2,6) and (2,C_ind2,3)(3,C_resp2,4).

This example represents a call setup function for one-way communication from user 1 to user 2. Primitives C_req, C_ind, C_resp, C_conf, C_rej and Rel_ind describe connection request, indication, response, confirmation, reject and release indication, respectively.

2.3 Integration Expression

As discussed in Section 1, several service specifications (component service specifications), each of which specifies a subfunction of the target protocol, are integrated into one. *Integration expression* gives an information on how to integrate the components into one. The syntax of the integration expression is defined by a context free grammar IG shown in Table 1, where E is a start symbol and there are eight production rules.

For example, let S_A, S_B, S_C and S_D be component service specifications, then expressions $S_A|S_B$, $S_A{\downarrow}S_B|S_C$ and $(S_A|(S_B{\downarrow}S_C)^*_{\{3,5\}})|S_D$ are all integration expressions.

Table 1: Grammar IG for integration expression

Rule1	$E ::= T$	Rule5	$T ::= (E)$
Rule2	$E ::= T\|E$	Rule6	$T ::=$ *s-spec*
Rule3	$E ::= T{\downarrow_F}E$	Rule7	$F ::=$ *set of integer*
Rule4	$E ::= T^*_F$	Rule8	$F ::= \varepsilon$

Note that there are three kinds of operators in the integration expression, that is, "|", "↓" and "*". These represent the following three integration operations on component service specifications: an *alternative integration* $S_A|S_B$, a *sequential integration* $S_A{\downarrow}S_B$ and a *recursive integration* S^*_A. The definition of these operations will be presented in Section 5.

2.4 Protocol Specification

A *protocol specification* (or simply protocol) **P** consists of two FSMs and is represented by two directed graphs PE_1 and PE_2. $PE_i (i = 1, 2)$ includes three types of transitions. The first is a *primitive transition* which is the same as that of the service specification. The second is a *sending transition* labeled by $!m$, which means that a message m is transmitted to another PE. The third is a *receiving transition* labeled by $?m$, which means the reception of a message m from another PE.

A node of PE_1(or PE_2) is a *final node* iff there is no outgoing transition from it. A node of PE_1(or PE_2) is a *receiving node* iff all outgoing transitions from it are receiving transitions.

A *global state* of protocol $\mathbf{P} = (PE_1, PE_2)$ is a quad-tuple $g = [v, w, x, y]$, where v and w are nodes in PE_1 and PE_2, respectively, and x and y are the concatenations of the protocol messages (Intuitively, v and w represent the current states of PE_1 and PE_2, respectively, and x and y represent messages stored in a communication channel from PE_2 to PE_1 and those from PE_1 to PE_2, respectively). The *initial global state* is $g_0 = [v_0, w_0, \varepsilon, \varepsilon]$ where v_0 and w_0 are the initial nodes of PE_1 and PE_2, respectively, and ε is the empty string.

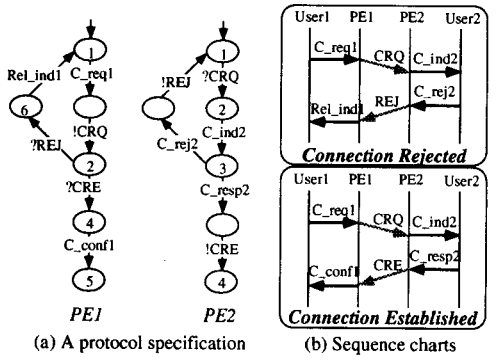
Assume that $g = [v, w, x, y]$ is a global state of protocol $\mathbf{P} = (PE_1, PE_2)$. The *next global state* g' of g is defined iff exactly one of the following six conditions is satisfied. In the following, E_1 and E_2 are the primitives in PE_1 and PE_2, respectively, e represents a protocol message, and $\cdot$ is concatenation operator.
(1)If (v, E_1, v') exists in PE_1, then $g' = [v', w, x, y]$.
(2)If (w, E_2, w') exists in PE_2, then $g' = [v, w', x, y]$.
(3)If $(v, !e, v')$ exists in PE_1, then $g' = [v', w, x, y \cdot e]$.
(4)If $(w, !e, w')$ exists in PE_2, then $g' = [v, w', x \cdot e, y]$.
(5)If $(v, ?e, v')$ exists in PE_1 and $x = e \cdot x'$, then $g' = [v', w, x', y]$.
(6)If $(w, ?e, w')$ exists in PE_2 and $y = e \cdot y'$, then $g' = [v, w', x, y']$.

A global state g is *reachable* iff g is an initial global

state or there exists at least one sequence of global states $g_0, g_1, ..., g_n(= g)$ such that g_0 is the initial global state and g_{r+1} is the next global state of g_r $(r = 0, ..., n-1)$.

A reachable global state $g = [v, w, x, y]$ of protocol $\mathbf{P} = (PE_1, PE_2)$ is an *unspecified reception state* iff it satisfies at least one of the following conditions.
(1) v is either a receiving node or a final node, $x = e \cdot x'$ and no transition $(v, ?e, v')$ exists in PE_1.
(2) w is either a receiving node or a final node, $y = e \cdot y'$ and no transition $(w, ?e, w')$ exists in PE_2.

A reachable global state $g = [v, w, x, y]$ of $\mathbf{P} = (PE_1, PE_2)$ is a *deadlock state* iff both v and w are receiving nodes and $x = y = \varepsilon$. Then, a protocol $\mathbf{P} = (PE_1, PE_2)$ is *safe* iff any reachable global state of $\mathbf{P}$ is free from both unspecified reception states and deadlock states.



(a) A protocol specification (b) Sequence charts

Figure 4: A protocol specification $\mathbf{P} = (PE_1, PE_2)$

An example of protocol specification is shown in Figure 4(a). This example is a *safe* protocol specification which realizes a connection setup function prescribed by the service specification in Figure 3. The protocol messages CRQ, CRE and REJ mean connection request, response and reject, respectively. Two sequence charts in Figure 4(b) describe two execution sequences "connection rejected" and "connection established" performed by this protocol.

Because of its definitions, protocol specification generally has much larger operational state space(*i.e.* the number of reachable global states) than that of service specification (*i.e.* the number of nodes in service specification).

3 Overview of Proposed Method

The protocol derivation problem in this paper is formally defined as follows:
Input: A set of service specifications (component service specifications) $\{S_A, S_B, ..., S_C\}$ and an integration expression *exp*.
Output: A protocol specification (integrated protocol specification) $\mathbf{P} = (PE_1, PE_2)$ satisfying the following two conditions C1 and C2.
Condition C1: $\mathbf{P}$ is safe.
Condition C2: In $\mathbf{P}$, the execution ordering of primitives prescribed in the component service specifications is kept in accordance with *exp*.

A protocol specification $\mathbf{P}=(PE_1, PE_2)$ is *correct* iff both of conditions C1 and C2 are satisfied.

The proposed method consists of the following two stages.
Stage 1(Component Integration) All of the component service specifications are integrated into one integrated service specification in accordance with the given integration expression.
Stage 2(Protocol Synthesis) The integrated service specification obtained at Stage 1 is transformed into an integrated protocol specification.

Since the result of the protocol synthesis is needed to illustrate the dynamic behavior of the result of component integration, at first we explain the protocol synthesis stage in the next section, and then we discuss the component integration stage in Section 5.

4 Protocol Synthesis Stage

4.1 Protocol Synthesis Method

In this section, we discuss the protocol synthesis method to be applied in the "protocol synthesis stage", in which the transformation from an integrated service specification into an integrated protocol specification is performed.

The input/output relation of the protocol synthesis is as follows.
Protocol Synthesis:
Input: An integrated service specification $\mathbf{S}$ obtained at the component integration stage.
Output: An integrated protocol specification $\mathbf{P} = (PE_1, PE_2)$ satisfying the following conditions R1 and R2.
Condition R1: $\mathbf{P}$ is safe.
Condition R2: The execution ordering of primitives prescribed in $\mathbf{S}$ is kept in $\mathbf{P}$.

For limited pages, we will briefly explain the protocol synthesis. The detail of protocol synthesis can be referred in [6, 11]. In the following discussion, we suppose $i, j = 1, 2$, $i \neq j$ unless especially specified.

Protocol synthesis in the proposed method consists of the following three steps.

Step 1: This step obtains two service specifications SAPi-S $(i = 1, 2)$ by projecting a given service specification $\mathbf{S}$ onto each SAPi $(i = 1, 2)$. In the projection, each primitive transition of $\mathbf{S}$ not associated with SAPi is substituted by ε in SAPi-S.

Step 2: This step synthesizes the protocol specification $\mathbf{P} = (PE_1, PE_2)$ from the projected service specifications. This synthesis is performed by applying transition synthesis rules shown in Table 2. In Table 2, Ei denotes some primitives in SAPi-S, and e denotes a protocol message which is uniquely generated by a primitive. Additionally, we define a function $OUT(w)$ which returns a set of indices of primitives

outgoing from a node w in the service specification. Each pair of rules Ak and Bk $(1 \leq k \leq 3)$ is applied to pairs of transitions (v, Ei, w) in SAPi-S and (v, ε, w) in SAPj-S, respectively.

Table 2: Transition Synthesis Rules

Rule	Input	Condition	Output
A.1	Ei, SAPi-S	OUT(w)={i}	Ei, PEi
B.1	ε, SAPj-S		ε, PEj
A.2	Ei, SAPi-S	OUT(w)={j} or OUT(w)={1,2}	Ei, !e, PEi
B.2	ε, SAPj-S		?e, PEj
A.3	LEi, SAPi-S	Message e is caused by Ei.	PEi
B.3	ε, SAPj-S		?e, PEj

Step 3: Finally, ε transitions are removed from the protocol specification by applying the ε removal algorithm in [5]. Using this algorithm, the protocol specification obtained in Step2 can be reduced to an equivalent FSM.

Figure 4 shows a protocol specification which is synthesized from the service specification shown in Figure 3.

4.2 Well-Formed Service Specification

In general, protocol specifications satisfying the conditions R1 and R2 cannot be *always* obtained from any service specifications by the protocol synthesis. In other words, a service specification from which a protocol satisfying R1 and R2 is synthesized belongs to a special class of the service specifications defined as *well-formed*(see Section 2.2).

Consider a service specification **S** shown in Figure 5(a). According to the definition, **S** is not well-formed because Condition PL is not satisfied for parallel node 1. This service specification specifies a connection setup function for two-way communication: from user 1 to user 2 and from user 2 to user 1. A protocol specification $\mathbf{P} = (PE_1, PE_2)$ shown in Figure 5(b) is synthesized from **S**. **P** performs correctly if only one of two connection requests is triggered by one user. However if user 1 and user 2 *simultaneously* execute two primitives C_req1 and C_req2, respectively, then two connection requests cause a collision as shown in Figure 5(c) and **P** may reach an unspecified reception state via the following sequence of global states: $[1, 1, \varepsilon, \varepsilon]$, $[100, 1, \varepsilon, \varepsilon]$, $[100, 106, \varepsilon, \varepsilon]$, $[2, 106, \varepsilon, CRQ]$, $[2, 7, CRQ', CRQ]$. Then **P** is not *safe*.

In order to resolve the unspecified reception caused by the above parallel execution of primitives, some

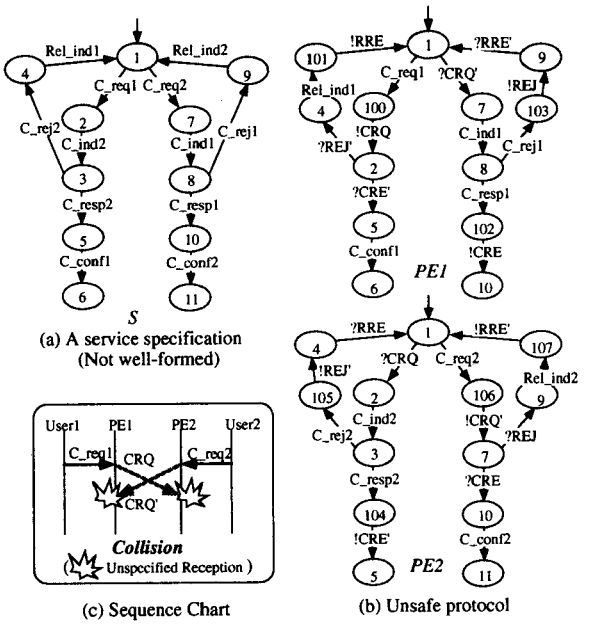


Figure 5: Example of parallel execution

receiving transitions must be added to the protocol specification. Before applying protocol synthesis, if we add some L transitions to the service specification so that Condition PL holds for any parallel state, we can avoid the unspecified reception. L transition Lp_i is such an auxiliary transition in a service specification which is translated into receiving transitions of a message caused by execution of primitive p_i(see transition synthesis rule A3,B3 shown in Table 2).

In the example of Figure 5(a), if two L transitions $(2, LC\_req2, 2)$ and $(7, LC\_req1, 2)$ are added to **S** before applying protocol synthesis, two extra transitions $(2, ?CRQ', 2)$ and $(7, ?CRQ, 2)$ are synthesized in PE_1 and PE_2 respectively. Then, we can avoid the unspecified reception even if the parallel execution of C_req1 and C_req2 occurs. It is observed by the following sequence of global states from a global state $[2, 7, CRQ', CRQ]$: $[2, 7, CRQ', CRQ]$, $[2, 7, \varepsilon, CRQ]$, $[2, 2, \varepsilon, \varepsilon]$, ...

Lemma 1 If a service specification **S** is well-formed, then a protocol specification $\mathbf{P} = (PE_1, PE_2)$ synthesized from **S** satisfies Conditions R1 and R2.

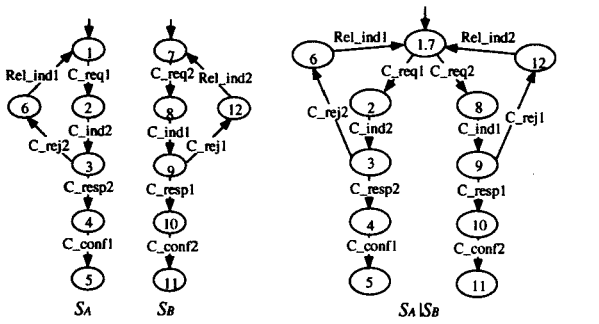
5 Component Integration Stage

This section presents the integration method of the component service specifications. The component integration is carried out by the following three integration operations: *alternative integration*, *sequential integration* and *recursive integration*. We explain the alternative integration in Section 5.1, and the sequential and recursive integrations in Section 5.2. Then,

Section 5.3 summarizes the component integration method.

5.1 Alternative Integration ($S_A|S_B$)

Alternative integration combines the initial nodes of two component service specifications S_A and S_B. The integrated service specification is denoted by $S_A|S_B$. The protocol $\mathbf{P} = (PE_1, PE_2)$ synthesized from $S_A|S_B$ can perform the functions of either S_A or S_B, but not simultaneously.



(a) Component service specifications (b) An integrated service specification (Not well-formed)

Figure 6: Example of an integrated service specification

For example, Figure 6(a) shows two component service specifications S_A and S_B. As discussed before, S_A specifies a call setup function for one-way communication from user 1 to user 2. On the other hand, S_B specifies the one from user 2 to user 1. By joining the initial nodes of S_A and S_B, we get the integrated service specification $S_A|S_B$ shown in Figure 6(b). $S_A|S_B$ specifies a half-duplex connection setup function, that is, either of two-way call setups can be executed.

Note that the above description is only for the minimal requirement of an alternative integration. It does not necessarily maintain the correctness of the synthesized protocol. To attain correctness, we must address a problem of *component competition* to be explained below.

A component competition arises when the protocol tries to initiate the execution of both components simultaneously. Consider again the integrated service specification in Figure 6(b). $S_A|S_B$ is the same as **S** shown in Figure 5(a) and the protocol synthesized from $S_A|S_B$ is the same as $\mathbf{P} = (PE_1, PE_2)$ in Figure 5(b). As discussed in Section 4.2, $S_A|S_B$ is not a well-formed service specification and parallel execution of C_req1 and C_req2 induces the unspecified reception in the synthesized protocol.

The reason of the unspecified reception is considered as the competition of two service functions S_A and S_B. Primitives C_req1 and C_req2 initiate the execution of S_A and S_B, respectively. Suppose that two primitives C_req1 and C_req2 are executed simultaneously by user 1 and user 2, respectively. Then PE_1 initiates the function of S_A (call setup from user 1 to user 2), while PE_2 initiates the function of S_B (call setup from user 2 to user 1). Thus, functions of S_A and S_B compete with each other and the coordination between PE_1 and PE_2 is lost.

The component competition happens when the following condition CC holds.

Competition Condition (CC): Let v_0 and w_0 be the initial nodes of two component service specifications S_A and S_B, respectively. If v_0 has at least one outgoing edge of primitive p such that $sap(p) = i$, then w_0 has at least one outgoing edge of primitive q such that $sap(q) = j$ $(i, j = 1, 2\ i \neq j)$.

To resolve the competition, we prioritize the component service specifications in advance. When the competition occurs, the execution of low priority function is aborted. In order to realize such mechanism, we systematically add some L transitions to the integrated service specification.

Now, we present the alternative integration. In the following, we say that a SAPi-path ρ in $S_A|S_B$ is *inherited from* S_A (or S_B) iff, before the integration, ρ is included in S_A (or S_B).

Alternative Integration $S_A|S_B$:
Input: Two service specifications S_A and S_B. Let v_0 and w_0 be the initial nodes of S_A and S_B, respectively. Without loss of generality, we assume that the priority of S_A is higher than that of S_B.
Output: The alternative integrated service specification denoted by $S_A|S_B$, obtained by Procedure $S_A|S_B$.
Procedure $S_A|S_B$:

Step 1: $S_A|S_B$ is formed by combining the initial nodes of S_A and S_B. The initial node of $S_A|S_B$ is denoted by $v_0.w_0$.

Step 2: Check the condition CC. If CC is satisfied, then repeat the following substeps 2-a and 2-b as long as new L transitions can be added to $S_A|S_B$.

Substep 2-a: If [v is the last node of a SAPi-path from $v_0.w_0$ which is inherited from S_A, and p_i is the last primitive of the SAPi-path] and [w is a node which is reachable from $v_0.w_0$ over a SAPj-path inherited from S_B], then for each w, add a transition (w, Lp_i, v) to $S_A|S_B$.

Substep 2-b: If [q_j is the last primitive of a SAPj-path from $v_0.w_0$ which is inherited from S_B] and [r is a node which is reachable from $v_0.w_0$ over a SAPi-path inherited from S_A], then for each r add a self loop (r, Lq_j, r).

Note that if S_A and S_B commonly include the same primitives outgoing from the initial states, then $S_A|S_B$ is no longer deterministic as required. However, this problem can be resolved by relabeling the primitive in one of S_A and S_B.

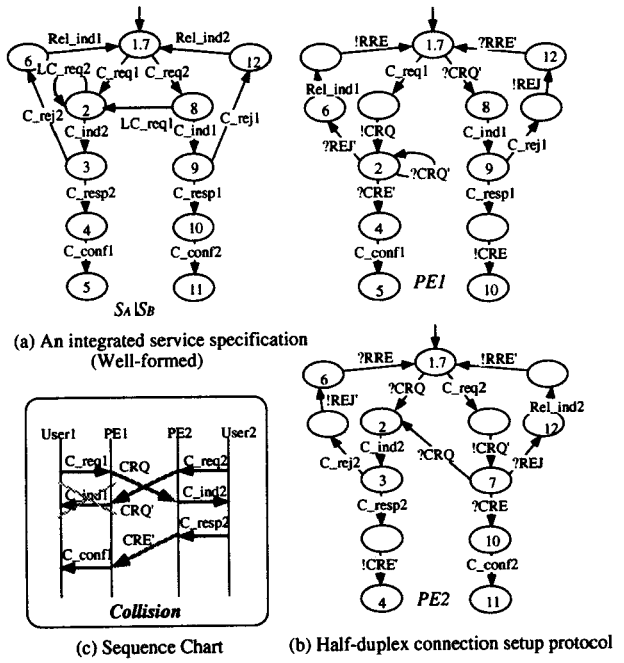


Figure 7: Illustration of alternative integration

Figure 7 (a) shows an integrated service specification $S_A|S_B$ obtained from the component service specifications S_A and S_B shown in Figure 6 by applying the alternative integration. Figure 7(b) shows the half-duplex connection setup protocol synthesized from $S_A|S_B$. Here, we suppose that the priority of S_A is higher than that of S_B. Therefore, the connection request of user 2 is discarded when the collision happens as shown in Figure 7(c).

Now, we give the following lemma on the alternative integration.

Lemma 2 An integrated service specification $S_A|S_B$ is well-formed if [condition CC does not hold, but the following (1) holds] or [condition CC holds and the followings (1)-(3) hold].
(1) Both S_A and S_B are well-formed.
(2) Neither S_A nor S_B includes any SAPi-cycle containing the initial node.
(3) Neither S_A nor S_B includes any reachable SAPi-path starting from the initial node.

5.2 Sequential Integration ($S_A{\downarrow_F}S_B$) and Recursive Integration ($S_{A_F^*}^*$)

Sequential integration combines two service specifications S_A and S_B by joining some final nodes of S_A with the initial node of S_B. The integrated service specification is denoted by $S_A{\downarrow}S_B$. The protocol synthesized from $S_A{\downarrow}S_B$ can perform two functions of S_A and S_B as successive two phases. On the other hand, *recursive integration* combines one service specification S_A with itself. The integrated service specification is denoted by S_A^*. The protocol synthesized from S_A^* can perform one function of S_A repeatedly.

In the following, we use $V_f(S)$ to denote a set of all final nodes in service specification S. Now, we present the sequential and recursive integrations.

Sequential Integration $S_A{\downarrow_F}S_B$:
Input: Two service specifications S_A and S_B, and a set of nodes $F \subseteq V_f(S_A)$.
Output: The sequential integrated service specification denoted by $S_A{\downarrow_F}S_B$, obtained by Procedure $S_A{\downarrow_F}S_B$. If $F = V_f(S_A)$, then it is denoted by $S_A{\downarrow}S_B$ omitting F.
Procedure $S_A{\downarrow_F}S_B$: Join all the final nodes of S_A in F to the initial node of S_B. The initial node of S_A becomes the initial node of $S_A{\downarrow_F}S_B$.

Recursive Integration $S_{A_F^*}^*$:
Input: A service specifications S_A and a set of nodes $F \subseteq V_f(S_A)$
Output: The recursive integrated service specification denoted by $S_{A_F^*}^*$, obtained by Procedure $S_{A_F^*}^*$. If $F = V_f(S_A)$, then it is denoted by $S_A{}^*$ omitting F.
Procedure $S_{A_F^*}^*$: Join all the final nodes of S_A in F to the initial node of S_A.

Note that the above two integrations cannot be executed if S_A has no final node. As for these two integrations, the following Lemmas hold:

Lemma 3 An integrated service specification $S_A \downarrow_F S_B$ is well-formed if both component service specifications S_A and S_B are well-formed.

Lemma 4 An integrated service specification $S_{A_F^*}^*$ is well-formed if both component service specifications S_A is well-formed.

5.3 Component Integration Method

Several component service specifications, which are given as the input of the component integration stage, are integrated into one by successive applications of three integration operations according to the given integration expression. The order of the applications is uniquely determined by syntax analysis of the given integration expression. After the integrated service specification is generated by the component integration operations, it is transformed into the target protocol specification in the protocol synthesis stage.

Now, we give the following theorem with respect to the correctness of the target protocol.

Theorem 1 If the integrated service specification **S** obtained at the component integration stage is well-formed, then the protocol specification **P** finally derived from **S** by the protocol synthesis method is correct.

Theorem 1 implies that the correctness of the target protocol can be checked on the service specification level. In other words, to check if the target protocol is

correct or not can be reduced to the decision problem if the integrated service specification is well-formed or not. Lemmas 2, 3 and 4 provide the sufficient conditions for the integrated service specifications to be well-formed. So, we can find the following guideline for designing the correct protocol specification.

Component Integration:

Step 1: Develop the component service specifications so that all of them are well-formed.

Step 2: Based on the integration expression, select two service specifications as the components which are integrated at this time (In the case of recursive integration, we select one service specification). Then, for the service specifications, check if the integrated service specification will be well-formed or not by using Lemmas 2, 3 or 4. If it will not be well-formed, then abort the procedure and redesign the component service specifications (at Step 1 again).

Step 3: Apply the integration operation to the service specifications. If some integration operations still remain, then go to Step 2. Otherwise, we can obtain the well-formed integrated service specification, which will be transformed into a correct protocol at protocol synthesis stage.

Step 2 can be easily implemented by using simple path trace algorithm for the service specifications, and it takes at most $O(n)$ state space, where n is the total number of nodes of the component service specifications. Additionally, all of the three integration operations are quite simple procedures. These facts imply that the component integration requires no special knowledge or techniques of the protocol specification (e.g., reachability analysis) and that it can be executed in a reasonable time.

On the other hand, the previous component integration methods [3, 4, 8], which correspond to the proposed three integration operations, require the validation of the component protocol specifications based on the reachability analysis for checking some conditions (just like Step 2) in order to ensure the correctness of the integrated protocol. However, since the operational state space of protocol specification is generally even much larger than that of service specification, this validation exponentially takes a lot of time and space because of the state explosion problem of the protocol[1, 7], especially when the size of component becomes large.

6 Concluding Remarks

In this paper, we have proposed a framework for designing communication protocols from component service specifications. Important advantage of the proposed method is that the component integration can be performed with in the small state space and without special knowledge or techniques of communication protocol since it is carried out on the service specification level. The proofs of lemmas and theorem, numerical comparison with the previous method and application to more practical protocol are delegated to the paper[12].

Finally, we summarize the further research studies in the following.

(a) An extension of the proposed technique to n (≥ 2) entities protocol model.

(b) An examination of the integration other than the three integrations.

References

[1] Brand, D. and Zafiropulo, P., "On communicating finite state machines," *Journal of the ACM*, Vol.30, No.2, pp-323-342, 1983.

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

[3] Chow, C.-H., Gouda, M. G., and Lam, S. S., "An exercise in constructing multi-phase communication protocols," *Proc. ACM SIGCOMM'84*, pp.493-503, June. 1984.

[4] Chow, C.-H., Gouda, M. G., and Lam, S. S., "A discipline for constructing multi phase communication protocols," *ACM Trans. on Computer Systems*, Vol.3, No.4, pp.315-343, Nov. 1985.

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

[6] Kakuda, Y., Igarashi, H. and Kikuno, T., "Automated synthesis of protocol specifications with message collisions and verification of timeliness," *Proc. of Second Int'l. Conf. on Network Protocols(ICNP'94)*, pp.143-150, Oct. 1994.

[7] Lin, F.J., Chu, P.M. and Liu, M.T., "Protocol verification using reachability analysis: The state space explosion problem and relief strategies," *Proc. ACM SIGCOMM'87 Workshop*, pp.126-135, 1987.

[8] Lin, H.-A., "Constructing protocols with alternative functions," *IEEE Trans. on Computers*, Vol.40, No.4, pp.376-386, Apr. 1991.

[9] Lin, H.-A., "A methodology for constructing communication protocols with multiple concurrent functions," *Distributed Computing*, Vol.3, pp.23-40, Dec. 1988.

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

[11] Nakamura M., Kakuda Y. and Kikuno T., "Protocol synthesis from acyclic formed service specifications," *Proc. of Int'l. Conf. on Information Networking(ICOIN'94)*, pp.177-182, Dec. 1994.

[12] Nakamura M., Kakuda Y. and Kikuno T., "On constructing communication protocols from component-based service specifications," *Computer Communications*, Aug. 1996, to appear.

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

[14] 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.

Information about The IEEE Computer Society and its services is available by calling our Customer Service Department at **+1-714-82-8380** or by e-mail: **cs.books@computer.org**. The society maintains its home page on the World Wide Web at **http://www.computer.org**