ELSEVIER

Computer Communications 19 (1996) 1200–1215

computer communications

On constructing communication protocols from component-based service specifications

Masahide Nakamura*, Yoshiaki Kakuda*, Tohru Kikuno*

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

Abstract

Constructing communication protocols from component service specifications, each of which specifies a subfunction of the target protocol, enables efficient development of a large and complex communication protocol. Concerning this construction, related techniques have been already proposed: integration of component protocol specifications into a single protocol specification and transformation of service specifications to protocol specifications. However, the integration needs special knowledge of communication protocols, and the transformation requires that a large and complex service specification should be developed as input to produce the target protocol. In order to cope with these problems, this paper proposes a new method which at first integrates component service specifications into a single service specification, and then transforms the service specification into the target protocol by a protocol synthesis technique. The most important point of view is that component integration is performed at the service specification level rather than the protocol specification level. Additionally, we define a class of 'well-formed' service specification which ensures correctness of the target protocol. As a result, the integration and transformation can be efficiently executed in small state space without special knowledge of communication protocols. Finally, we have shown the effectiveness of the proposed method by constructing a part of the real-life OSI protocol FTAM.

Keywords: Service specification; Component; Integration; Protocol synthesis; Correctness

1. Introduction

The recent rapid progress of computer communication systems enables the advancement and diversification of communication services. Accordingly, the communication protocols which realize the communication services become larger and more complex. As a result, development of such large and complex communication protocols has become a serious problem.

In order to attack this problem, the following approach can be considered as a practical solution: This approach consists of the three stages:

Stage 1: Divide the functionality of a service into subfunctions.

Stage 2: Describe service specifications for the subfunctions as components (we call them *component service specifications*).

Stage 3: Obtain the target protocol specification (call it an *integrated protocol specification*) based on the component service specifications.

The major advantages of this approach are summarized as follows: (a) since the service specification is used as the starting point (that is, at Stage 2), the content of the required services are clearly specified even by the service designer, who does not have special knowledge about the communication protocols; (b) since a hierarchical design is adopted, at Stage 2 we can easily develop each component with a relatively smaller size, and focus on a single function without considering interaction with other functions; (c) if an effective decomposition is found at Stage 1, we can reuse subfunctions and thus reuse components in a future development.

In this paper, we focus the discussion on Stage 3, and propose a new technique to implement Stage 3. As described before, Stage 3 derives an integrated protocol specification from the given component service specifications. Fig. 1(a) shows the content of Stage 3 schematically.

So far, two kinds of techniques, which are closely related to the implementation of Stage 3, have already been proposed. One is techniques that integrate component specifications into a single specification. All of the integration techniques are at the *protocol specification level*. That is, as shown in Fig. 1(b), these techniques integrate several

* Email: {masa-n, kakuda, kikuno}@ics.es.osaka-u.ac.jp

PII: S0140-3664(96)01154-1

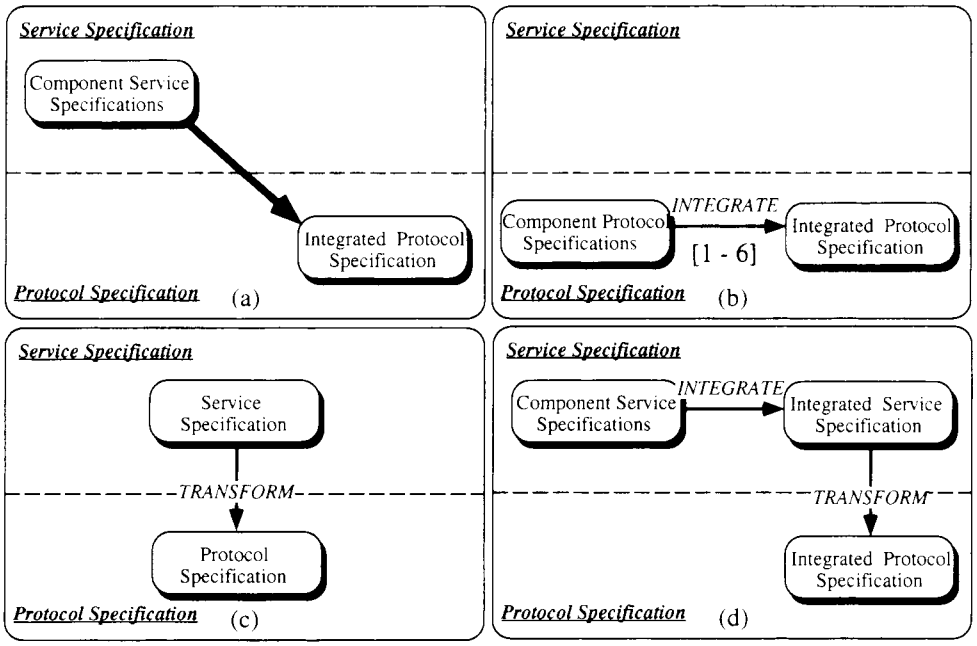


Fig. 1. Derivation of integrated protocol specification.

component protocol specifications into a single protocol specification. Chow et al. [1, 2] proposed a constructing algorithm for a *multiphase protocol*, in which multi-phases are sequentially executed, and each phase performs a distinct subfunction. Next, Lin et al. [3] and Singh et al. [4] extended this algorithm [1, 2] by removing some restrictions. Moreover, Lin proposed two nice integration algorithms [5, 6]. In an *alternating function protocol* [5], the user can select any one from several functions, but is restricted to execute only one function at a time. On the other hand, in a *concurrent function protocol* [6], several functions can be performed concurrently.

The other techniques transform service specifications into protocol specifications, as shown in Fig. 1(c). The most efficient and reliable technique is a so-called *protocol synthesis technique* [7–14] that automatically derives a protocol specification from a service specification without specification errors.

In this paper, we propose a new method for protocol derivation from component service specifications. Fig. 1(d) shows the essential parts of the proposed method. At first, we integrate the component service specifications into a single service specification. The most important point is that the component integration is performed at the *service specification level*. The integration is executed using three kinds of component integration (alternative, sequential and recursive integrations) at the service specification level, which corresponds to the existing protocol integration methods [1, 2, 5]. Then, we transform the single service specification into an integrated protocol specification by a protocol synthesis method. The protocol synthesis algorithm is fundamentally based on the protocol synthesis algorithms [9–12].

In the proposed method, we introduce a concept of 'well-formed' service specification. This 'well-formed' service specification plays an essential role in the proposed method. If the given component service specifications are well-formed, then an integrated service specification is also well-formed. Additionally, if the integrated service specification is well-formed, then we can obtain the correct protocol by the protocol synthesis. The advantages of the proposed method are summarized as follows: (a) since the component integration is carried out at the service specification level, we can efficiently execute the integration in a much smaller state space; (b) by utilizing the concept of 'well-formed' service specifications, we can ensure the correctness of the target protocol at each integration step.

This paper is organized as follows. Section 2 gives necessary definitions of service and protocol specifications, and Section 3 outlines the proposed method. Then Section 4 presents the protocol synthesis algorithm, and Section 5 presents the component integration algorithm at the service specification level. Section 6 shows the effectiveness of the proposed method by constructing a part of FTAM according to the proposed method, and evaluates the result. Finally, Section 7 concludes the paper with future research.

2. Definitions

2.1. Communication model

Fig. 2 shows the communication model which we adopt in this paper. This describes a particular layer (a layer $\langle N \rangle$) and its interaction with the upper layer (a layer $\langle N+1 \rangle$). The layer $\langle N \rangle$ *protocol entities* (PEs) provide the communication

services (called *the layer* ⟨*N*⟩ *services*) to the layer ⟨*N* + 1⟩ users. A layer ⟨*N*⟩ service is realized by exchanging *service primitives* between the layer ⟨*N* + 1⟩ users and the layer ⟨*N*⟩ PEs through the interface, called *service access points* (SAPs). The layer ⟨*N*⟩ PEs exchange *protocol messages* with each other through the underlying communication medium, whose functions are usually performed by the layer ⟨*N* − 1⟩ services. The rules that govern the exchange of protocol messages among the PEs are called the layer ⟨*N*⟩ *protocol.*

In this paper, we define a *service specification* as the specification which describes a layer ⟨*N*⟩ service, and a *protocol specification* as the one which describes a layer ⟨*N*⟩ protocol. We assume that the number of PEs is two, and that there exists one-to-one correspondence among User*i*, PE*i* and SAP*i* with the same index ($i = 1, 2$). Moreover, we assume that the communication medium is reliable, and that the protocol messages are delivered in FIFO order.

2.2. Service specification

A service specification defines sequences of service primitives to be realized as communication services, which are exchanged between users and protocol entities through SAPs. A service specification is modeled by a Finite State Machine (FSM). The FSM is usually represented by a labeled directed graph. Thus, in this paper, we define a service specification directly using a labeled directed graph.

Definition 1. *A service specification S is defined by a labeled directed graph* $S = \langle V_s, \Sigma_s, T_s, v_0 \rangle$ *where,*

- V_s *is a set of nodes representing* ***service states*** *(or simply states).*
- $\Sigma_s = \mathcal{SP} \cup \mathcal{L}$ *is a set of labels attached to the edges.* $\mathcal{SP}$ *is a set of* ***service primitives*** *(or simply primitives).* Each primitive $p \in \mathcal{SP}$ *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*. Next,* $\mathcal{L} = \{Lp \mid p \in \mathcal{SP}\}$*, and each element* $Lp \in \mathcal{L}$ *is called an* ***L primitive.***
- T_s *is a set of directed edges representing* ***service state transitions*** *(or simply transitions). For simplicity, we use a triple* (u, p, v) *to represent a directed edge from a node* $u \in V_s$ *to a node* $v \in V_s$ *with a label* $p \in \Sigma_s$*. (The directed edge* (u, p, v) *intuitively implies that state u of the service specification S is changed into state v by executing primitive p.) We call an edge* (u, p, v) *with* $p \in \mathcal{SP}$ *a* ***primitive transition*** *and call an edge* (u, p, v) *with* $p \in \mathcal{L}$ *an* ***L transition.***
- $v_0 \in V_s$ *is an* ***initial service state.*** □

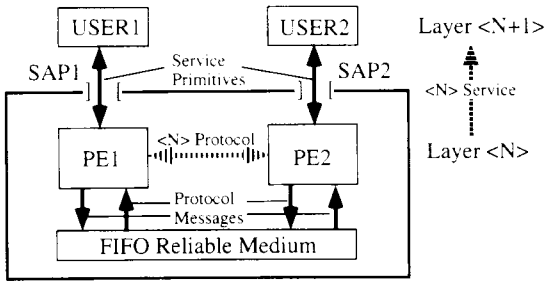


Fig. 2. Communication architecture model.

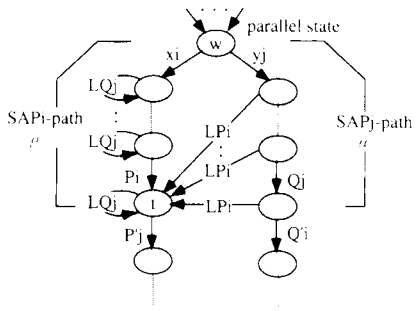


Fig. 3. Explanation for condition P.

In this paper, we assume that all service specifications *S* are *deterministic*, that is, no two outgoing edges from any node have identical labels.

Remark 1. The service specification is different from the previous ones proposed in [13, 14], in the sense that it may include the L primitives. L primitives *Lp* are the auxiliary primitives for the protocol synthesis algorithm discussed in Section 4, and are translated into receptions of a message caused by execution of primitives *p*.

Definition 2. *A path* $p = (v_1, p^1, v_2), (v_2, p^2, v_3), \ldots, (v_n, p^n, v_{n+1})$ *in a service specification S derives an* ***execution ordering*** *among primitives* $p^1 \geq p^2 \geq \cdots \geq p^n$ *which implies that any primitive* $p^i (1 \leq i \leq n)$ *must be executed earlier than other primitives* $p^j (i + 1 \leq j \leq n)$. □

Definition 3. *A state* $u \in V_s$ *is called a* ***final state*** *iff there is no outgoing transition* (u, p, v) *for any p and v. A state* $u \in V_s$ *is called a* ***parallel state*** *iff u has at least two primitive transitions* (u, p, v) *and* (u, q, v') *with* $sap(p) = 1$ *and* $sap(q) = 2$. □

Definition 4. *A path* $(v_1, p^1, v_2), \ldots, (v_{n-1}, p^{n-1}, v_n), (v_n, p^n, v_{n+1})$ *in a service specification S is called a* ***SAPi-path*** ($i = 1, 2$) *iff* $sap(p^k) = i$ ($k = 1, \ldots, n$) *and* $(v_{n+1}, p^{n+1}, v_{n+2})$ *such that* $sap(p^{n+1}) = j$ ($j = 1, 2$, $j \neq i$) *exists in S. Additionally, the states* v_1 *and* v_{n+1} *are called a* ***head state*** *and a* ***tail state*** *of the SAPi-path, respectively, and the primitive* p^n *is called a* ***last primitive*** *of the SAPi-path. A SAPi-path* ($i = 1, 2$) *is called a SAPi-cycle iff its head state and its tail state are identical. A SAPi-path* ($i = 1, 2$) *is called a* ***reachable SAPi-path*** *iff its tail state is a final state of S.* □

Now, we define a class of service specification.

Definition 5. *A service specification S is called a* ***well-formed*** *service specification iff no parallel state exists in S or the following condition* P *holds for any parallel state w in S.*

Condition P. Consider a SAPi-path from w (call it path ρ) and SAPj-path from w (call it path μ) $(i,j = 1,2, i \neq j)$. Suppose that P_i and Q_j are the last primitives of ρ and μ, respectively. Then, (1) the ρ is neither reachable SAPi-path nor SAPi-cycle. Similarly, the μ is neither reachable SAPj-path nor SAPj-cycle; (2) for any state v on ρ, an L transition (v, LQ_j, v) exists in S; and (3) for any state r on μ and the tail state t of ρ, an L transition (r, LP_i, t) exists in S (see Fig. 3). □

Remark 2. The class of service specification proposed in Refs. [9, 10] is a subclass of the class of the well-formed service specification, in the sense that service specification in this paper allows outgoing primitives with an identical SAP index from any parallel state.

An example of service specification S is shown in Fig. 4. All of the transitions are primitive transitions, and there is no L transition. State 1 is an initial state. State 4 is a final state because there are no outgoing transitions from it. Since there is no parallel state, S is well-formed. From state 1, there are two SAP1-paths (1,Dt_req1,2) and (1,DtEnd_req1,3), and from state 2 there is a SAP2-path (2,Dt_ind2,1), and so on. There is no reachable SAPi-path and no SAPi-cycle for any $i\,(i = 1,2)$.

This example models a data transfer service function from user 1 to user 2. Primitives Dt_req, Dt_ind, DtEnd_req and DtEnd_ind represent Data request, Data indication, Data End request and Data End indication, respectively. The scenario of this example is briefly described as follows: User 1 repeatedly transmits data with a Data request primitive. If the transmission is completed, user 1 informs user 2 of the completion of the data transfer using a Data End request primitive.

2.3. Integration expression

As discussed in Section 1, several service specifications (we call them component service specifications), each of which specifies a subfunction of the target protocol, are integrated into one. The *integration expression* gives us information on how to integrate the component service specifications.

Definition 6. *Consider a context free grammar* $G_I = (\{E,T,F\}, \Sigma, P, E)$, *where* $\Sigma = \{|, \downarrow, *, (,), [,], x_i (i = 1,2, \ldots, n), y_j (j = 1,2, \ldots, m)\}$. *The set of production rules P is shown in Table 1. Let* $L(G_I)$ *denote a language generated by* G_I. *We introduce a substitution* τ *which substitutes* x_i *and* y_j *by a certain component service specification and a certain set of positive integers, respectively, and also transforms* $[\alpha,\beta,\gamma,\downarrow]$ *and* $[\alpha,\gamma,*]$ *into* $\alpha\downarrow_\gamma\beta$ *and* α^*_γ, *respectively. Then an* ***integration expression*** *is a string obtained by applying the substitution* τ *to any terminal string in* $L(G_I)$. □

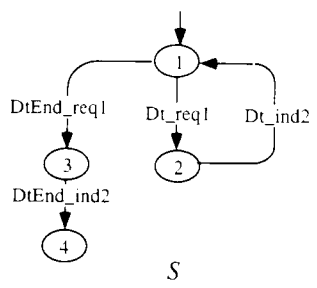


Fig. 4. Example of service specification.

Table 1
Production rules P

No.	Production Rule	No.	Production Rule	
1	E::= T	5	T::= (E)	
2	E::= T|E	6	T::= xi	$(1 \le i \le n)$
3	E::= [T, E, F, ↓]	7	F::= yj	$(1 \le j \le m)$
4	T::= [T, F, *]	8	F::= ε	

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

Remark 3. There are three kinds of symbols '|', '↓' and '*' in the integration expression. 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 definitions of the operations will be presented in Section 5.

2.4. Protocol specification

A protocol specification consists of a pair of specifications for protocol entities (PEs). As in Definition 1, we also define the protocol entity specification using a labeled directed graph.

Definition 7. *A* ***protocol entity specification*** $PE_i\,(i = 1,2)$ *is defined by a labeled directed graph* $PE_i = \langle V_{pi}, \Sigma_{pi}, T_{pi}, v_{0i}\rangle$ $(i = 1,2)$ *where,*

- V_{pi} *is a set of nodes representing protocol states (or simply states).*
- $\Sigma_{pi} = \mathcal{SP} \cup \mathcal{MES}$ *is a set of labels attached to the edges.* $\mathcal{SP}$ *is the same as that in Definition 1.* $\mathcal{MES}$ *is a set of* ***protocol message events****, and each element in* $\mathcal{MES}$ *is specified either* !m *or* ?m, *where* m *is a protocol message.*
- T_{pi} *is a set of directed edges representing* ***protocol state transitions*** *(or simply transitions). As in Definition 1, we use a triple* (u,p,v), $u,v \in V_{pi}$, $p \in \Sigma_{pi}$ *to represent a directed edge from node* u *to node* v *with label* p. *(Intuitively,* (u,p,v) *implies that state* u *in* PE_i *is changed into state* v *by executing* p. *Especially,* $p =$!m *and* $p =$?m *represent sending* m *and receiving* m, *respectively.)*
- $v_{0i} \in V_{pi}$ *is an* ***initial protocol state****.*

A ***protocol specification*** *(or simply protocol) P consists of two protocol entity specifications* PE_1 *and* PE_2. *Thus, we represent it by* $P = (PE_1, PE_2)$. □

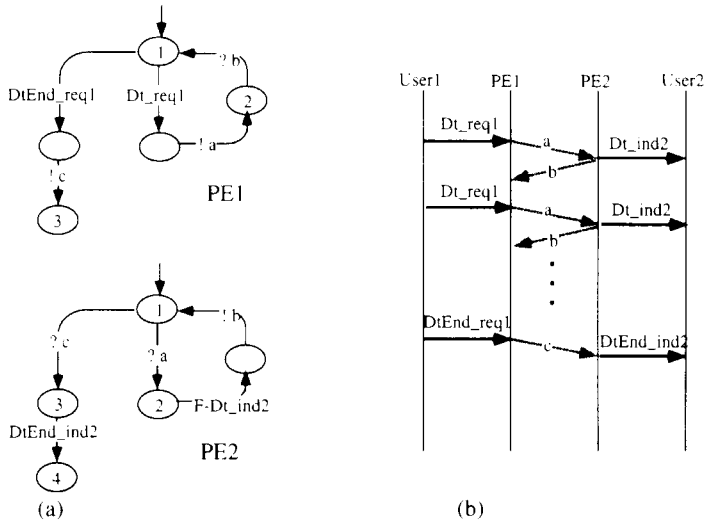


Fig. 5. Example of protocol specification. (a) A protocol specification; (b) sequence chart.

Definition 8. *A state $u \in V_{p1} \cup V_{p2}$ is called a **final state** iff there is no outgoing transition (u,p,v) for any p and v. A state $u \in V_{p1} \cup V_{p2}$ is called a **receiving state** iff any outgoing transition from u is a message receiving event $(u,?m,v)$ for any m and v.* □

Definition 9. *A global state of a protocol $P = (PE_1, PE_2)$ is a quad-tuple $g = [v,w,x,y]$, where $v \in V_{p1}$, $w \in V_{p2}$, and x and y are strings over the protocol messages in $\mathcal{MES}$. (Intuitively, node v represents the current state of PE_1 and string x represents messages stored in a communication medium from PE_2 to PE_1. Similarly, node w and string y have the same meaning for 'state of PE_2' instead of 'state of PE_1' and 'from PE_1 to PE_2' instead of 'from PE_2 to PE_1'.) The **initial global state** is $g_0 = [v_{01}, v_{02}, \varepsilon, \varepsilon]$ where ε is the empty string.* □

Definition 10. *Let $g = [v,w,x,y]$ be a global state of a protocol $P = (PE_1, PE_2)$. Then, the global state g' defined by the following is called the **next global state** of g. In the following, E_1 and E_2 are primitives in PE_1 and PE_2, e represents a protocol message, and $\cdot$ is a concatenation operator.*

Case 1: *If* $(v,E_1,v') \in T_{p1}$, *then* $g' = [v',w,x,y]$.
Case 2: *If* $(w,E_2,w') \in T_{p2}$, *then* $g' = [v,w',x,y]$.
Case 3: *If* $(v,!e,v') \in T_{p1}$, *then* $g' = [v',w,x,y \cdot e]$.
Case 4: *If* $(w,!e,w') \in T_{p2}$, *then* $g' = [v,w',x \cdot e,y]$.
Case 5: *If* $(v,?e,v') \in T_{p1}$ *and* $x = e \cdot x'$, *then* $g' = [v',w,x',y]$.
Case 6: *If* $(w,?e,w') \in T_{p2}$ *and* $y = e \cdot y'$, *then* $g' = [v,w',x,y']$. □

Definition 11. *A global state g of a protocol P is **reachable** iff g is the initial global state of the P or there exists at least one sequence of global states $g_0, g_1, \ldots, g_n = g$ such that each $g_{r+1} (r = 0, \ldots, n-1)$ is the next global state of g_r.* □

In this paper, we focus on the following two types of protocol errors: unspecified reception and deadlock.

Definition 12. *A reachable global state $g = [v,w,x,y]$ of a protocol P is called an **unspecified reception state** iff g satisfies the following conditions* (1) *or* (2):

(1) *v is either a receiving state or a final state, $x = e \cdot x'$ and $(v,?e,v') \notin T_{p1}$.*
(2) *w is either a receiving state or a final state, $y = e \cdot y'$ and $(w,?e,w') \notin T_{p2}$.* □

Definition 13. *A reachable global state $g = [v,w,x,y]$ of a protocol P is called a **deadlock state** iff both v and w are receiving states and $x = y = \varepsilon$.* □

Definition 14. *A protocol P is **safe** iff any reachable global state of the P is neither an unspecified reception state nor a deadlock state.* □

Fig. 5(a) shows an example of protocol specification. State 3 of PE_1 and state 4 of PE_2 are final states, and state 2 of PE_1 and state 1 of PE_2 are receiving states. This protocol is safe, since any reachable global state is neither an unspecified reception state nor a deadlock state.

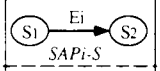
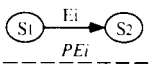
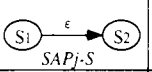
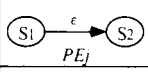
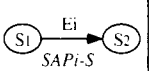
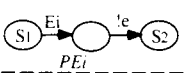
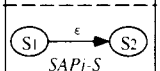
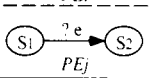
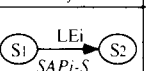
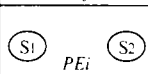
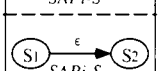
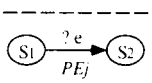
This protocol realizes a data transfer function prescribed by the service specification in Fig. 4. The protocol messages a, b and c are caused by executions of primitives Dt_req1, Dt_ind2 and DtEnd_req1, respectively. A sequence chart in Fig. 5(b) describes the execution sequence of data transfer performed by this protocol.

3. Outline of protocol derivation

The protocol derivation problem to be discussed in this paper is defined as follows:

Input. A set of component service specifications $\{S_A, S_B, \ldots, S_C\}$ and an integration expression *exp*.

Table 2
Transition synthesis rules

Rule	Input	Condition	Output
A.1		OUT(S2)={i}	
B.1			
A.2		OUT(S2)={j} or OUT(S2) ={1,2}	
B.2			
A.3		Message e is caused by Ei.	
B.3			

Output. A protocol specification $P = (PE_1, PE_2)$ which satisfies the following conditions C1 and C2:

Condition C1. The protocol P is safe.

Condition C2. The execution ordering of primitives derived in each component service specification is kept in the protocol P. □

The protocol derivation algorithm consists of the following stages:

Stage 1 (Component Integration). The component service specifications are integrated into an integrated service specification in accordance with the given integration expression *exp*.

Stage 2 (Protocol Synthesis). The single service specification obtained in Stage 1 is transformed into a protocol specification.

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

4. Protocol synthesis

4.1. Protocol synthesis method

In this section, we describe a protocol synthesis method which transforms an integrated service specification into an integrated protocol specification. The protocol synthesis algorithm is essentially the same as that in [9–12], and is an extension of Saleh's synthesis algorithm [13, 14].

The protocol synthesis problem is defined as follows:

Input. A service specification S obtained by Stage 1.

Output. A protocol specification $P = (PE_1, PE_2)$ satisfying the following conditions R1 and R2.

Condition R1. The protocol P is safe.

Condition R2. The execution ordering of primitives derived in S is kept in the protocol P.

The protocol synthesis algorithm consists of the following steps. In the following, we suppose that $i, j \in \{1, 2\}$, $i \neq j$ unless specified otherwise.

Step1. At this step, two service specifications SAP1-S, SAP2-S are obtained by projecting a given service specification S onto each SAP1, SAP2, respectively. In the projection, each primitive transition of S, which is not associated with SAPi, is substituted by ε in SAPi-S.

Step2. This step synthesizes the protocol specification $P = (PE_1, PE_2)$ from the projected service specifications SAP1-S, SAP2-S. Actually, the synthesis is performed by applying transition synthesis rules shown in Table 2. In Table 2, Ei denotes a primitive in SAPi-S, and e denotes a protocol message caused by the primitive Ei. Additionally, a function $OUT(s)$ returns a set of indices of primitives outgoing from a node s in the service specification. Each pair of rules Ak and Bk $(1 \leq k \leq 3)$ is together applied to pairs of input transitions (S_1, Ei, S_2) in SAPi-S and (S_1, ε, S_2) in SAPj-S, respectively. As a result, pairs of input transitions are substituted by pairs of corresponding output transitions if a corresponding condition holds.

The intuitive concepts for these rules are explained briefly as follows:

Rule A1,B1. These rules imply that any messages need not be exchanged among PEs because two primitives are successively executed at the same SAP(SAPi).

Rule A2,B2. After the primitive Ei occurs at SAPi, other primitive can be executed at other SAP (SAPj). Therefore, a protocol message e is transmitted to the other PE (PEj) for the synchronization. The protocol message e is uniquely generated for each primitive Ei. Then we say that message e is *caused* by primitive Ei. (The name of each message can be determined by the protocol designer. In the real-life protocol, there exists a correspondence between a primitive and a protocol message, e.g. Connection Request primitive corresponds to CR PDU.)

Rule A3,B3. These rules transform the L primitive LEi into a message reception $?e$ where e is caused by Ei.

Step3. Finally, ε transitions are removed from the protocol specification by applying the ε removal algorithm in [15]. □

For a more detailed description of this protocol synthesis algorithm, the reader is referred to Refs. [9–12]. In the following, we use the term 'synthesize' to represent 'synthesize by the proposed method in Section 4.1' unless specified otherwise.

Fig. 5(a) shows a protocol specification which is obtained from the service specification shown in Fig. 4 by the proposed synthesis method.

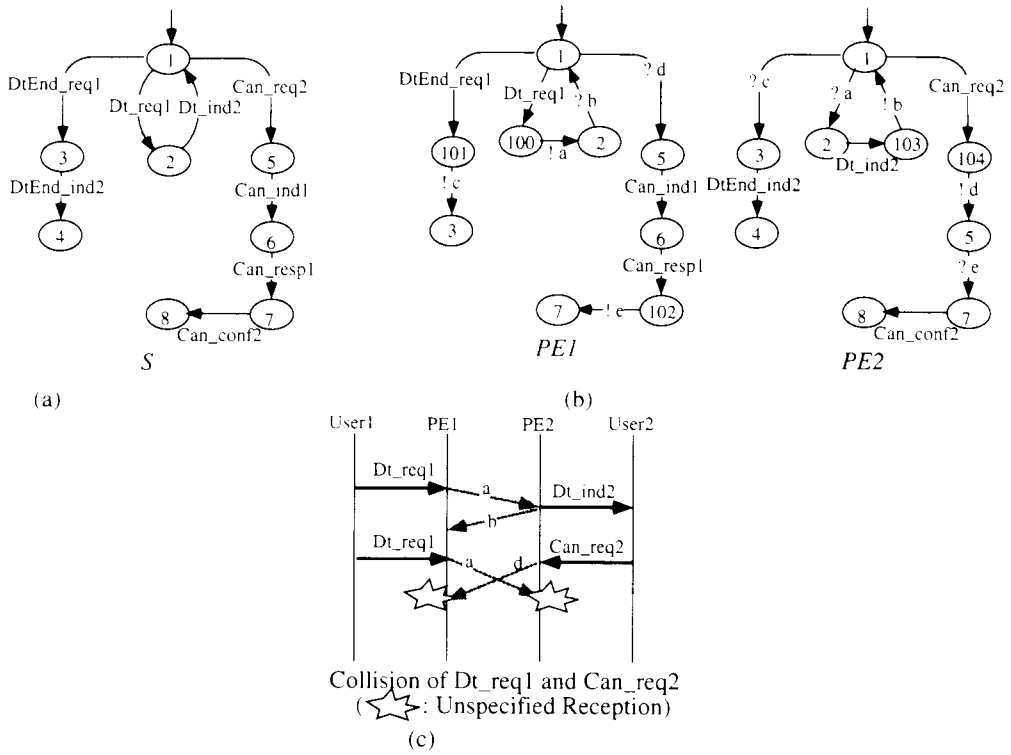


Fig. 6. Example of parallel execution. (a) A service specification; (b) a protocol specification; (c) sequence chart.

4.2. *Well-formed service specification*

By the protocol synthesis algorithm presented in Section 4.1., Conditions R1 and R2 *always* cannot be assured to hold.

Consider a service specification S shown in Fig. 6(a). It specifies the function of a data transfer with a cancel. The part of states 1, 2, 3 and 4 specifies a data transfer function from user 1 to user 2, which is the same as the service specification in Fig. 4. Next, the part of states 1, 5, 6, 7 and 8 specifies a cancel function from user 2 to user 1. Primitives Can_req, Can_ind, Can_resp and Can_conf represent Cancel Request, Indication, Response and Confirmation, respectively. Note that the executions of data transfer and transfer cancel are triggered by user 1 and user 2, respectively.

Then Fig. 6(b) shows a protocol specification $P = (PE_1, PE_2)$ synthesized from S. This protocol does work correctly if only one of two functions is triggered by users. However, if user 1 and user 2 *simultaneously* execute two primitives Dt_req1 and Can_req2, respectively, then two requests cause a collision as shown in Fig. 6(c). Then P may reach an unspecified reception state via the following global state transition sequence: $[1, 1, \varepsilon, \varepsilon]$, $[100, 1, \varepsilon, \varepsilon]$, $[100, 104, \varepsilon, \varepsilon]$, $[2, 104, \varepsilon, a]$, $[2, 5, d, a]$. Thus P is not *safe*. Similarly, a collision of DtEnd_req1 and Can_req2 induces the unspecified reception in P.

The protocol synthesis algorithm proposed in Section 4.1 is an extension of Saleh's synthesis method [13, 14] in the sense that our method allows the *parallel execution of primitives* at different SAPs. As explained in the above example, the parallel execution of primitives outgoing from a parallel state (for example, state 1 in Fig. 6(a)) may induce the unspecified reception in the synthesized protocol specification. To avoid the unspecified reception, some extra receiving transitions must be added in the protocol specification. So, as an extension to Saleh's method, we have newly introduced L primitives into the service specification as shown in Definition 1. Then, we define Transition Synthesis Rules A3,B3 such that an L transition Lp_i is translated into a reception of a message caused by the execution of primitive p_i in the protocol synthesis. As a result, even when the service specification includes parallel states, we can obtain the correct protocol.

Consider again the service specification in Fig. 6(a). If we add four L transitions (2,LCan_req2,5), (3,LCan_req2,5), (5,LDt_req1,5) and (5,LDtEnd_req1,5) to S, then for a new service specification extra transitions $(2, ?d, 5)$, $(3, ?d, 5)$ in PE_1 and $(5, ?a, 5), (5, ?c, 5)$ in PE_2 are newly generated and appended to the protocol specification in Fig. 6(b) by the transition synthesis rules A3,B3. As a result, even if the parallel execution of two primitives Dt_req1 and Can_req2 occurs, unspecified reception never occurs.

A set of service specifications, from which protocol specifications satisfying Conditions R1 and R2 are synthesized, forms a special class of service specifications. Such a service specification is in a class of *well-formed* service specifications as defined in Definition 5.

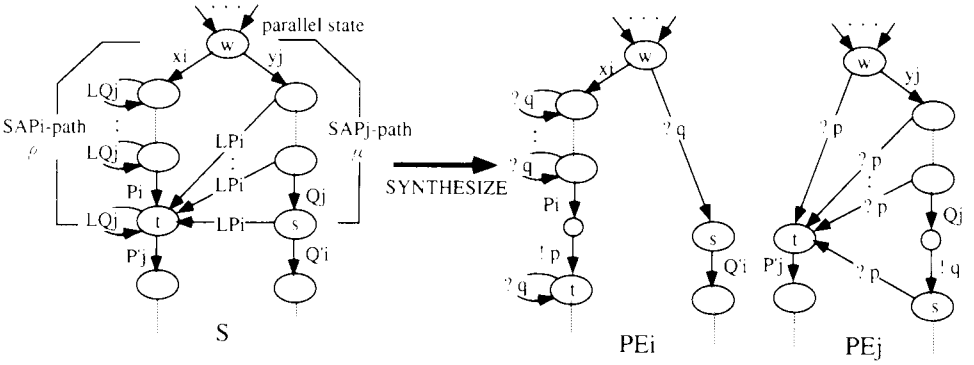


Fig. 7. Explanation for Lemma 1.

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

Proof (Sketch). From Definition 5, in the service specification S no parallel state exists or Condition P holds for any parallel state in S.

Case 1. If no parallel state exists in S, parallel execution of primitives does not occur. The proof in this case is already given in [13, 14].

Case 2. Consider the case that S includes some parallel states. Suppose that w is a parallel state in S and that x_i and y_j $(i, j \in \{1, 2\},\ i \neq j)$ are primitives leaving from w. By Condition P, there exists SAPi-path ρ and SAPj-path μ, both of which start from w. Let the last primitives of the SAPi-path ρ and SAPj-path μ be P_i and Q_j, respectively. Note that P_i and Q_j may be identical with x_i and y_j, respectively.

Fig. 7 shows a protocol specification $P = (PEi, PEj)$ synthesized from S. Without loss of generality, suppose that $i = 1, j = 2$. Consider a subcase that at the global state $[w, w, \varepsilon, \varepsilon]$ the parallel execution of primitives x_i and y_j does not occur. Then it is already proved [13, 14] that P reaches neither an unspecified reception state nor a deadlock state, and execution ordering derived in S is kept in the protocol P.

Next, consider a subcase that the parallel execution of x_i and y_j occurs at $[w, w, \varepsilon, \varepsilon]$. PEi moves on the path corresponding to ρ. Then after the primitive P_i is executed, PEi transmits a message p to PEj and enters state t. On the other hand, PEj moves on the path corresponding to μ. Then, PEj surely receives the message p transmitted by PEi and enters t via state s, because state s is a receiving state according to transition synthesis rules. A message q which may be transmitted by PEj is also received by PEi on the path from w to t, because t is surely a receiving state. Thus, the protocol P surely reaches a global state $[t, t, \varepsilon, \varepsilon]$ without dropping in some unspecified reception states or deadlock states. Additionally, the execution ordering is kept between primitives P_i and P_j' (in Fig. 7), although some redundant primitives may be interleaved. Next, a global state $[t, t, \varepsilon, \varepsilon]$ is nothing more than a global state which P reaches after only the primitive x_i is executed at $[w, w, \varepsilon, \varepsilon]$. After P reaches $[t, t, \varepsilon, \varepsilon]$, this subcase is reduced to the case that no parallel execution occurs. So, even if parallel execution of primitives occurs, Conditions R1 and R2 are satisfied. □

5. Component integration

In this section, we discuss the component integration state, in which component service specifications are integrated into a service specification.

The component integration problem is defined as follows:

Input. A set of component service specifications $\{S_A, S_B, \ldots, S_C\}$ and an integration expression *exp*.

Output. An integrated service specification S.

The component integration is carried out by executing *sequential integration*, *recursive integration* and *alternative integration*. In Sections 5.1 and 5.2 we describe these three integrations, then in Section 5.3 we summarize main results obtained for component integrations.

5.1. Sequential integration and recursive integration

The *sequential integration* combines two component service specifications S_A and S_B by joining some final state of S_A with the initial state of S_B and gets an integrated service specification. We denote the resultant service specification by $S_A \downarrow S_B$. The protocol, which is obtained by applying protocol synthesis to $S_A \downarrow S_B$, performs two service functions of S_A and S_B successively. Although the sequential integration can be executed on the service specification level, similar sequential integration is already presented on the protocol specification level in [1, 2].

In the following, we use $V_f(S)$ to denote a set of all final states in the service specification S. The sequential integration $(S_A \downarrow_F S_B)$ is defined as follows:

Input. Two service specifications S_A and S_B, and a set of final states $F \subseteq V_f(S_A)$.

Output. An integrated service specification, denoted by $S_A \downarrow_F S_B$. Especially when $F = V_f(S_A)$, we omit F and denote it just by $S_A \downarrow S_B$.

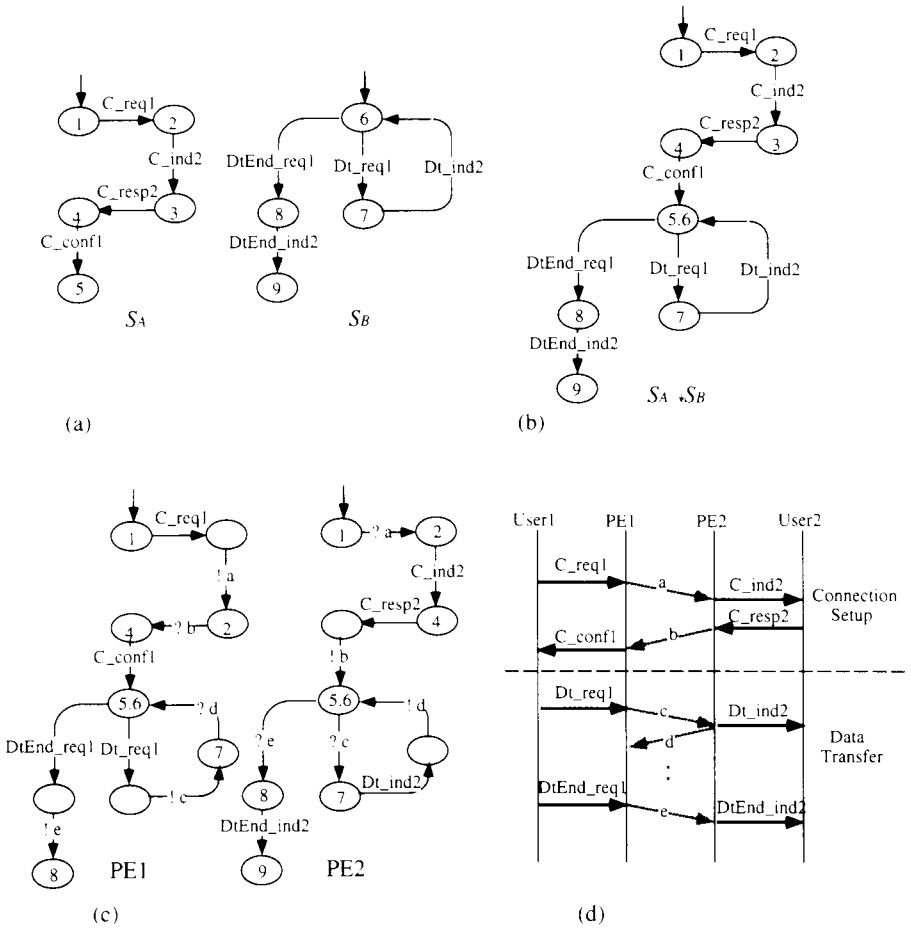


Fig. 8. Explanation for sequential integration. (a) Component service specifications; (b) an integrated service specification; (c) a protocol specification; (d) sequence chart.

Procedure ($S_A \downarrow_F S_B$). Join any final state $w \in F$ of S_A with the initial state v_0 of S_B, and generate a new state $w.v_0$ (concatenation of w and v_0). The initial state of S_A becomes newly the initial state of the integrated service specification $S_A \downarrow_F S_B$. □

For example, consider two component service specifications S_A and S_B shown in Fig. 8(a). S_A specifies a connection setup function from user 1 to user 2, where primitives C_req, C_ind, C_resp and C_conf denote Connection request, Connection indication, Connection response and Connection confirmation, respectively. S_B represents a data transfer function as mentioned before. By combining the final state 5 of S_A with the initial state 6 of S_B, we can obtain a new service specification $S_A \downarrow S_B$ shown in Fig. 8(b). Note that the service specification $S_A \downarrow S_B$ is transformed into a protocol specification P in Fig. 8(c) by the protocol synthesis. This protocol P implements the functions of connection setup and data transfer as two sequential phases as shown in Fig. 8(d).

Next, the *recursive integration* combines one component service specification S_A with itself. We denote the resultant service specification by S_A^*. The protocol, which is synthesized from S_A^*, performs the service function S_A repeatedly. The recursive integration ($S_{A_F}^*$) is defined as follows:

Input. A service specification S_A and a set of final states $F \subseteq V_f(S_A)$.

Output. An integrated service specification, denoted by $S_{A_F}^*$. Especially when $F = V_{sf}$, we omit F and denote it by S_A^*.

Procedure ($S_{A_F}^*$). Join any final state $w \in F$ with the initial state v_0 of S_A, and generate a new state $w.v_0$. The initial state of S_A newly becomes the initial state of the integrated service specification $S_{A_F}^*$. □

Remark 4. In the sequential and recursive integrations, we assume that S_A includes at least one final state. If S_A does not have any final state, these integrations cannot be applied.

For sequential integration and recursive integration, the following lemmas hold.

Lemma 2. *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.*

Proof. Let w be any final state in F and v_0 be the initial state of S_B. The outgoing transitions from the new state $w.v_0$ in

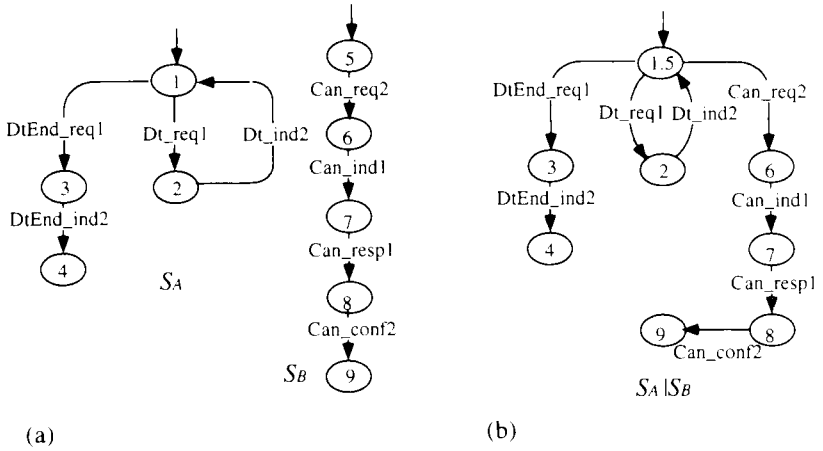


Fig. 9. Explanation for alternative integration. (a) Component service specifications; (b) an integrated service specification (not well-formed).

$S_A \downarrow_F S_B$ are also the outgoing transitions from v_0, because w has no outgoing transitions in S_A. Therefore, no other parallel states except for those in S_A and S_B are newly generated by sequential integration. Since S_A and S_B are well-formed, no parallel state exists in S_A and S_B, or Condition P is satisfied for any parallel state in S_A and S_B. Therefore, no parallel state exists in $S_A \downarrow_F S_B$ or Condition P is satisfied for any parallel state in $S_A \downarrow_F S_B$. □

Lemma 3. *An integrated service specification $S^*_{A_F}$ is well-formed if the component service specification S_A is well-formed.*

Proof. As in sequential integration, no other parallel states except for those in S_A are newly generated by recursive integration. So if S_A has no parallel state, then no parallel state exists in $S^*_{A_F}$. Next, consider the case that S_A has some parallel states. Subconditions (2) and (3) of Condition P are clearly satisfied for any parallel state in $S^*_{A_F}$ because Condition P is satisfied for any parallel state in S_A. By recursive integration, a cycle ρ containing parallel state r may be newly generated in $S^*_{A_F}$ by recursive integration. However, ρ never forms SAPi-cycle because S_A has no reachable SAPi-path from which r starts according to subcondition (1). Hence, subcondition (1) holds for any parallel state in $S^*_{A_F}$. Thus, Condition P is satisfied for any parallel state in $S^*_{A_F}$. □

5.2. Alternative integration

The *alternative integration* combines two component service specifications S_A and S_B by joining two initial states of S_A and S_B. We denote the resultant service specification by $S_A|S_B$. The protocol, which is obtained by applying protocol synthesis to the service specification $S_A|S_B$, performs the function of either S_A or S_B, but not simultaneously. Although the alternative integration can be executed on the service specification level, similar alternative integration is already presented on the protocol specification level in Ref. [5].

For example, Fig. 9(a) shows two component service specifications S_A and S_B. As discussed before, S_A specifies a data transfer function from user 1 to user 2, and S_B specifies a cancel function from user 2 to user 1. By joining the initial states of S_A and S_B, we get the integrated service specification $S_A|S_B$ shown in Fig. 9(b). $S_A|S_B$ specifies the function of a data transfer with cancel.

However, just combining only two initial states of two component service specifications does not necessarily assure the safety of the resultant protocol. To assure the safety, we must address a new other problem *component competition.*

The component competition arises when the protocol tries to initiate the executions of both functions of S_A and S_B simultaneously. Consider again the integrated service specification in Fig. 9(b). Since the service specification $S_A|S_B$ is the same as S shown in Fig. 6(a), the protocol synthesized from $S_A|S_B$ becomes the protocol specification P in Fig. 6(b). As already discussed in Section 4.2, $S_A|S_B$ is not a well-formed service specification, and parallel execution of Dt_req1 and Can_req2 leads the protocol P to unspecified reception state. The reason why P reaches the unspecified reception state is considered as the competition of two service functions S_A and S_B: when two primitives Dt_req1 and Can_req2 are executed in parallel by user 1 and user 2, respectively, PE_1 initiates the data transfer function of S_A while PE_2 initiates the data cancel function of S_B. Then, two functions compete with each other and the coordination between PE_1 and PE_2 is lost.

The component competition happens if the following condition Q holds:

Condition Q. *Let v_0 and w_0 be the initial states of S_A and S_B, respectively. For $i, j \in \{1, 2\}$, $i \neq j$, when a transition (v_0, p, v) such that $sap(p) = i$ exists in S_A, at least one transition (w_0, q, w) such that $sap(q) = j$ also exists in S_B simultaneously.* □

To resolve the competition, we introduce the priority into component service specifications in advance. When the competition occurs, the execution of function with low priority is aborted. In order to realize such mechanism,

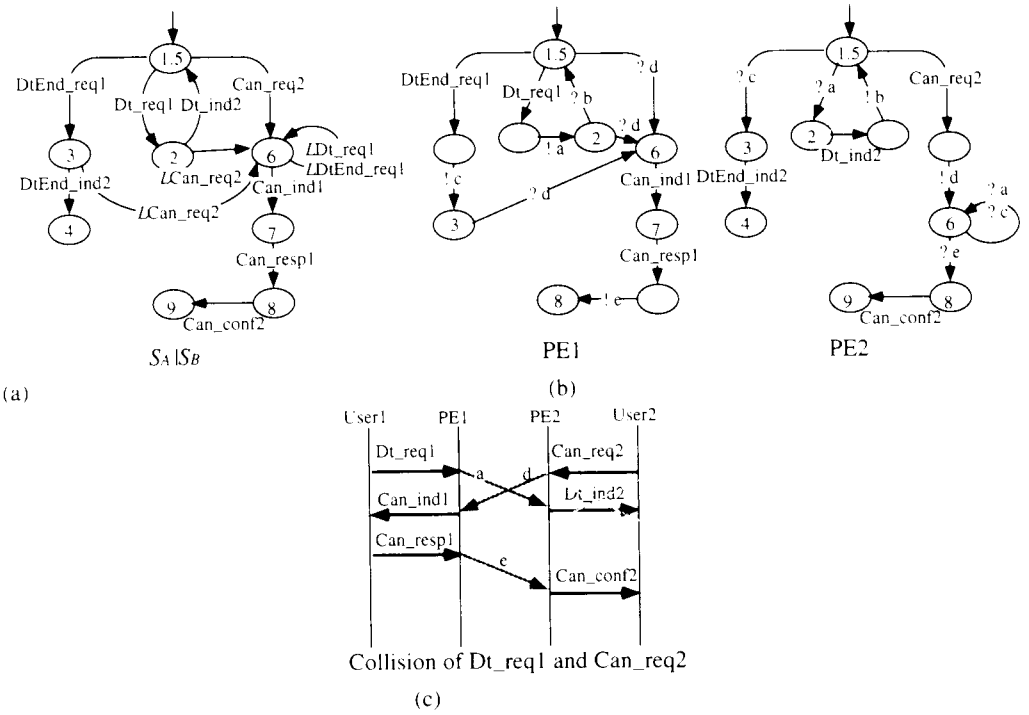


Fig. 10. Illustration of alternative integration. (a) An integrated service specification (well-formed); (b) data transfer protocol with cancel function; (c) sequence chart.

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 ($i = 1, 2$) in $S_A|S_B$ is *inherited from* S_A (or S_B) iff the ρ is included in $S_A|S_B$ because ρ is in S_A (or S_B). The alternative integration ($S_A|S_B$) is defined as follows:

Input. Two service specifications S_A and S_B. Let v_0 and w_0 be the initial states of S_A and S_B, respectively. Without loss of generality, we assume that the priority of S_B is higher than that of S_A.

Output. An integrated service specification, denoted by $S_A|S_B$.

Procedure ($S_A|S_B$):

Step1: Combine the initial states of S_A and S_B, and generate a new initial state $v_0.w_0$ of $S_A|S_B$.

Step2: If Condition Q is satisfied, then repeat the following Substeps a and b as long as new L transitions can be added to $S_A|S_B$.

Substep a: if (w is a *tail state* of a SAPi-path inherited from S_B from $v_0.w_0$, and p_i is a *last primitive* of the SAPi-path) and (v is a state which is reachable from $v_0.w_0$ by a SAPj-path inherited from S_A), then add a transition (v, Lp_i, w) to $S_A|S_B$.

Substep b: if (q_j is a *last primitive* of a SAPj-path inherited from S_A from $v_0.w_0$) and (r is a state which is reachable from $v_0.w_0$ by a SAPi-path inherited from S_B), then add a transition (r, Lq_j, r) to $S_A|S_B$. □

Figure 10(a) shows an integrated service specification $S_A|S_B$ obtained from the component service specifications S_A and S_B shown in Fig. 9(a). Fig. 10(b) shows a protocol specification for the data transfer protocol with cancel function which is synthesized from $S_A|S_B$. As shown in Fig. 10(c), when the competition occurs, the cancel function (specified in S_B) is executed while the data transfer (specified in S_A) is aborted in accordance with the priority assignment.

Note that if two component service specifications S_A, S_B have commonly the same primitive outgoing from the initial state, then the integrated service specification is no longer deterministic as required. However, this problem can be resolved by relabeling the primitive in one of component service specifications S_A, S_B.

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

Lemma 4. *An integrated service specification $S_A|S_B$ is well-formed if (condition* Q *does not hold, but the following condition* (1) *holds) or (condition* Q *holds and the following conditions* (1)–(3) *hold)*:

(1) *Both component service specifications S_A and S_B are well-formed.*
(2) *Neither S_A nor S_B includes SAPi-cycle containing the initial state.*
(2) *Neither S_A nor S_B includes reachable SAPi-path starting from the initial state.*

Proof. For S_A and S_B, if Condition Q is not satisfied, then the initial state of $S_A|S_B$ is not a parallel state. Even if $S_A|S_B$

includes some parallel states, for any parallel state condition P must hold by condition (1). Otherwise, if condition Q holds, then the initial state of $S_A|S_B$ becomes a parallel state. By conditions (2), (3) and substeps (a), (b) of the alternative integration, L transitions are added so that Condition P is satisfied for the initial state of $S_A|S_B$. For other parallel states in $S_A|S_B$, if they exist, Condition P holds by condition (1). □

5.3. Component integration algorithm

Several component service specifications, which are given as the input of the component integration problem, 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 using a so-called parsing tree for the integration expression (that is, a context free language). After the integrated service specification is generated by the component integration operations, it is transformed into the target protocol specification by the protocol synthesis algorithm.

As for three integration operations presented in Subsections 5.1 and 5.2, the following lemma holds:

Lemma 5. *The integrated service specifications $S_A \downarrow_F S_B$, preserves the execution ordering of primitives derived in both component service specifications S_A and S_B. Similarly, $S_A|S_B$ preserves the execution ordering derived in both S_A and S_B, and $S^*_{A_F}$ preserves the execution ordering derived in S_A.*

Proof. It is obvious that none of the three component integration operations delete, duplicate or reorder the primitive transitions in the component service specifications. Therefore, the execution ordering derived in the component service specifications is also derived in the integrated service specification. □

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

Theorem 1. *If the integrated service specification S obtained by the component integration algorithm is well-formed, then the protocol specification P finally derived from S by the protocol synthesis algorithm satisfies Conditions* C1 *and* C2.

Proof. Since S is well-formed, P satisfies both Conditions R1 and R2 according to Lemma 1. Condition R1 is just the same as Condition C1. Additionally, by Lemma 5, the integrated service specification preserves the execution ordering of primitives derived in the component service specifications. Therefore, Condition C2 is also satisfied if Condition R2 is satisfied. □

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 constructing the correct protocol specification.

Component integration algorithm

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, 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, go to Step 2. Otherwise, we can obtain the well-formed integrated service specification, which will be transformed into a correct protocol specification by the protocol synthesis algorithm.

Note that Step 2 can be easily implemented by using a simple path trace algorithm for the service specifications (This fact implies that any special knowledge of protocol verification is not necessary.)

6. Application

6.1. Part of FTAM

As an example, we try to construct a protocol for Bulk Data Transfer part of the FTAM (File Transfer, Access and Management ISO 8571) [15, 16] OSI Application layer using the proposed method.

In FTAM service, two service users *service initiator* (simply initiator) and *service responder* (simply responder) take part in an FTAM association. The Initiator is a user who begins the FTAM association and activates all operations of FTAM. The Responder is a corresponding user who responds to all requests from the Initiator. Bulk Data Transfer is triggered when the initiator executes either the primitive F-READ request or F-WRITE request. If the initiator executes the F-READ request, then the F-READ service starts. Similarly, if the F-WRITE request is executed, then the F-WRITE service starts.

The F-READ service specifies a data transfer from the responder to the initiator (that is, the initiator and responder are the *receiver* and *sender* of the data, respectively). This direction of data transfer is fixed until the F-READ service is completed. The data transfer continues until the responder informs the initiator of the completion by executing an F-DATA-END request primitive. When data transfer is

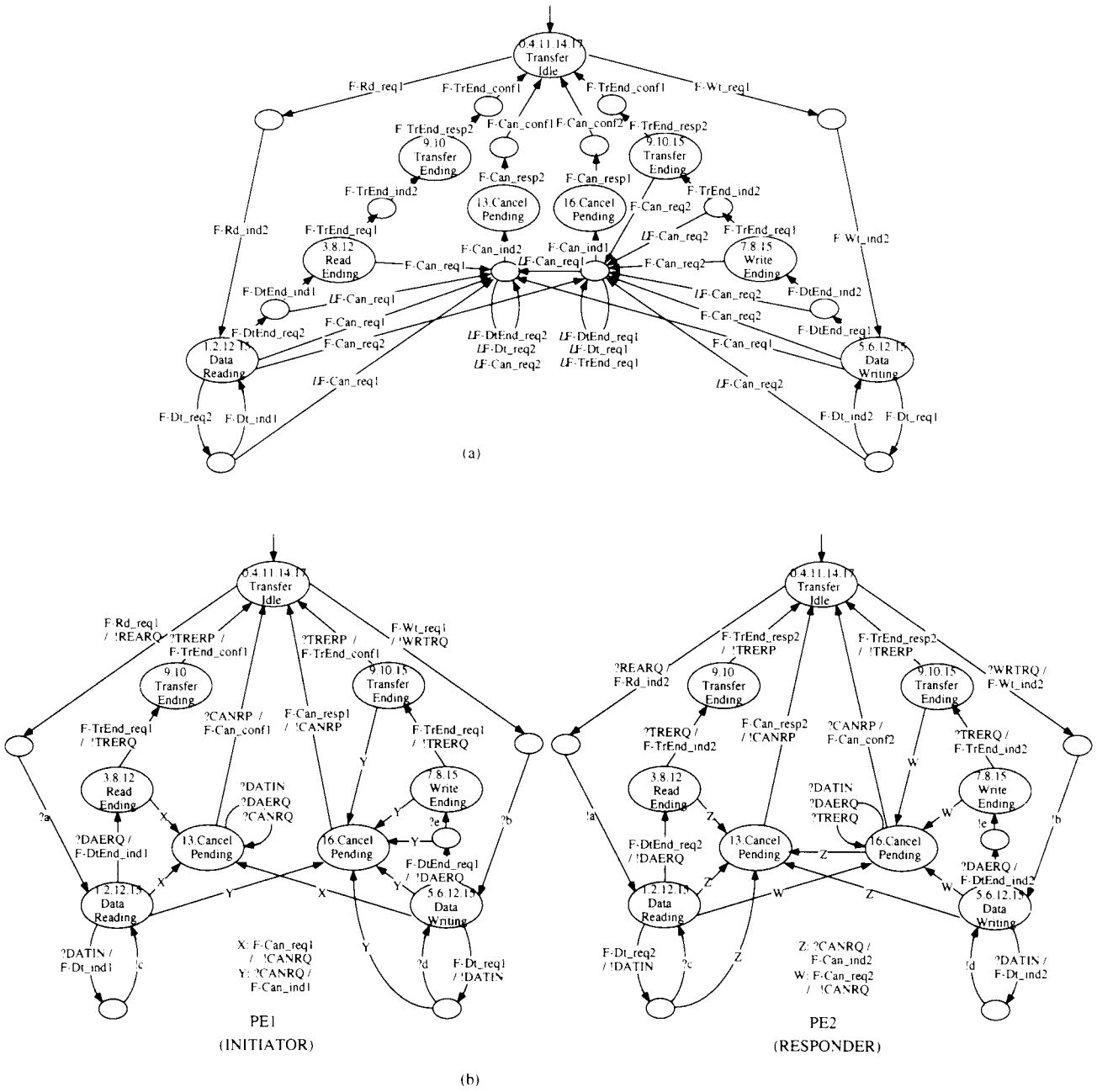


Fig. 12. Bulk Data Transfer part of FTAM. (a) Service specification; (b) protocol specification.

the collision of an F-CANCEL requests happens). Additionally, we assign a higher priority to $S_H|S_G$ than that of S_B (since the F-CANCEL service has a higher priority than any other services). We can verify that at each stage of the integrations, the integrated service specification is well-formed.

Similarly, we get the integrated service specification for the F-WRITE service based on the integration expression $(S_C \downarrow (S_D|(S_H|S_G)) \downarrow_{\{7\}} (S_E|S_H) \downarrow_{\{9\}} (S_F|S_H))^*$. Finally, two service specifications for F-READ service and F-WRITE service are integrated by the alternative integration, and we get the well-formed service specification for Bulk Data Transfer Service shown in Fig. 12(a).

This integrated service specification is then translated into a protocol specification by applying the protocol synthesis algorithm. As a result, we can obtain the target protocol specification $P = (PE_1, PE_2)$ shown in Fig. 12(b). In this figure, for convenience, two successive transitions (v_1, a, v_2), (v_2, b, v_3) are simply represented by one transition $(v_1, a/b, v_3)$.

6.2. Evaluation

The derived protocol shown in Fig. 12(b) is *almost* same as the Bulk Data Transfer Protocol Machine which is described using a state transition table in ISO 8571-4

Table 3
Comparison of state space

Service Function/Integration	S-Int.	P-Int.	Service Function/Integration	S-Int.	P-Int.	Service Function/Integration	S-Int.	P-Int.
SA	3	5	SH|SG	9	26	SC$\downarrow$(SD|(SH SG))	14	60
SB	4	10	SB (SH|SG)	12	52	SE SH	7	22
SC	3	5	SA$\downarrow$(SB (SH SG))	14	64	SC$\downarrow$(SD (SH SG))$\downarrow_{\{7\}}$ (SE|SH)	21	85
SD	4	10	SE|SG	7	13	SF|SH	7	13
SE	3	5	SA$\downarrow$(SB (SH|SG))$\downarrow_{\{3\}}$ (SE SG)	20	76	SC$\downarrow$(SD (SH|SG))$\downarrow_{\{7\}}$ (SE|SH)$\downarrow_{\{9\}}$(SF|SH)	26	98
SF	3	5	SA$\downarrow$(SB|(SH|SG))$\downarrow_{\{3\}}$ (SE|SG)$\downarrow_{\{9\}}$SF	22	80	(SC$\downarrow$(SD|(SH|SG))$\downarrow_{\{7\}}$ (SE SH)$\downarrow_{\{9\}}$(SF|SH))*	21	99
SG	5	9	(SA$\downarrow$(SB|(SH|SG))$\downarrow_{\{3\}}$ SE|SG)$\downarrow_{\{9\}}$SF)*	18	80	(SA$\downarrow$(SB|(SH SG))$\downarrow_{\{3\}}$ (SE|SG)$\downarrow_{\{9\}}$SF)*	38	178
SH	5	9	SD (SG SH)	12	52	(SC$\downarrow$(SD|(SH|SG))$\downarrow_{\{7\}}$ (SE|SH)$\downarrow_{\{9\}}$(SF|SH))*		

[16]. However, in our protocol, some redundant protocol messages *a*, *b*, *c*, *d* and *e* in Fig. 12(b) are generated, although these do not appear in ISO 8571-4.

Messages *a*, *b* and *e* may be deleted, but messages *c* and *d* cannot be erased. Suppose that we remove transitions !*c* and ?*c* from PE1 and PE2, respectively. Then we get into the following situation: if user 2 infinitely executes the F-DT_req2 at Data Reading state of PE2, then PE2 may never receive the cancel request message CANRQ from PE1. In the real-life protocol, such situation is resolved by service functions in the lower layer (like the flow control function, and so on). Really, messages *c* and *d* seem to correspond to Receive Ready PDUs in the Network Layer. However, since we model the lower layer only by a simple FIFO queue, and also model the protocol by a simple finite state machine, such messages *c* and *d* are essentially necessary for the correctness of the protocol. The optimization of redundant messages is one of our future works.

6.3. Comparison

In this paper, we propose three kinds of integration operations on the *service specification level* (we call them service integrations). On the other hand, there exist several component integration methods on the *protocol specification level* (call them protocol integrations). For example, alternative integration in Ref. [5], and sequential and recursive integrations in Refs. [1, 2] are protocol integrations. Here we try to compare the effectiveness of the service integration and protocol integration. In both integrations, we must verify some conditions to ensure the correctness of the target protocol by analyzing component specifications.

As discussed elsewhere [1, 2, 5], the conventional protocol integrations require validation of the protocols using reachability analysis. Unfortunately, it is known that reachability analysis exponentially takes a lot of time and cost, because of the state explosion problem [17, 18]. Therefore, if we integrate components with a very large size using protocol integration, verification of the conditions would be a bottleneck of the component integration, even though some state reduction techniques can be borrowed.

On the other hand, in our service integration, such a verification can be performed at the service specification level, which generally has a much smaller state space than the protocol specification. Moreover, the verification can be easily done by an algorithm for a simple path trace on the directed graph representing a service specification (as mentioned in Section 5.3).

Here, we compare the service and protocol integrations with respect to the operational state spaces of specifications. Table 3 shows the operational state space which is generated in the construction of the FTAM protocol discussed in Section 6.1. The column 'Service Function' represents the service function prescribed in the service specification. The column 'S-Int' represents the number of states in the service specification obtained using service integration, and 'P-Int' shows the number of reachable global states in the protocol specification obtained using protocol integrations.

From Table 3, we can observe that the state space on the protocol integration exponentially increases along with the progress of integrations, when comparing with that on the service integration.

Thus, a major advantage of the service integration is that we can operate it in a much smaller state space, compared with the protocol integration, especially when the size of the components is large.

7. Conclusion

In this paper, we have proposed a framework for designing communication protocols from component service specifications by using component integration and protocol synthesis techniques. The most important point is that component integration is performed at the service specification level, which generally has a much smaller state space than the protocol specification. Using the concept of a 'well-formed' service specification, we can guarantee at

the service specification level the correctness of the target protocol specification. Also, we have applied the proposed method to the construction of a real-life OSI protocol, and evaluated the effectiveness of the proposed method.

However, the following further research still remains, and is being studied:

(a) An extension of the proposed technique to n (≥ 2) entities protocol and unreliable communication medium.
(b) An optimization of the redundant messages in synthesized protocols.

References

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

[2] C-H. Chow, M.G. Gouda and S.S. Lam, A discipline for constructing multi phase communication protocols, ACM Trans. Computer Systems, 3(4) November 1985, pp. 315–343.

[3] H.-A. Lin, An improved method for constructing multiphase Communication Protocols, IEEE Trans. Computers, 42(1) (January 1993) 15–26.

[4] G. Singh and M. Sammeta, On the construction of multiphase communication protocols, Proc. Second Int. Conf. on Network Protocols (ICNP'94), October 1994, pp. 151–158.

[5] H.-A. Lin, Constructing protocols with alternative functions, IEEE Trans. Computers, 40(4) (April 1991), 376–386.

[6] H.-A. Lin, A methodology for constructing communication protocols with multiple concurrent functions, Distributed Computing, 3 (December 1988) 23–40.

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

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

[9] H. Igarashi, Y. Kakuda and T. Kikuno, Synthesis of protocol specifications for design of responsive protocols, IEICE Trans. Information and Systems, E76-D (11) (November 1993) 1375–1385.

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

[11] Y. Kakuda, M. Nakamura and T. Kikuno, Automated synthesis of protocol specifications from service specifications with parallelly executable multiple primitives, IEICE Trans. Fundamentals, E77-A (10) (October 1994) 1634–1645.

[12] M. Nakamura, Y. Kakuda and T. Kikuno, Protocol synthesis from acyclic formed service specifications, Proc. Int. Conf. Information Networking (ICOIN'94), December 1994, pp. 177–182.

[13] K. Saleh, Automatic synthesis of protocol specifications from service specifications, Proc. Int. Phoenix Conf. on Computers and Communications, March 1991, pp. 615–621.

[14] K. Saleh, A service-based method for the synthesis of communication protocols, Special issue on distributed computing and systems, Int. J. Mini and Microcomputers, 12(3) (1990) 97–103.

[15] ISO 8571-3, ISO – File service definition: File Transfer, Access and Management – Part 3, 1989.

[16] ISO 8571-4, ISO – File protocol specification: File Transfer, Access and Management – Part 4, 1989.

[17] D. Brand and P. Zafiropulo, On communicating finite state machines, J. ACM, 30(2) (1983) 323–342.

[18] F.J. Lin, P.M. Chu and M.T. Liu, Protocol verification using reachability analysis: The state space explosion problem and relief strategies, Proc. ACM SIGCOMM Workshop, 1987, pp. 126–135.