On Constructing n-Entities Communication Protocol and Service with Alternative and Concurrent Functions

Bhed Bahadur BISTA, Nonmember, Kaoru TAKAHASHI, Regular Member, and Norio SHIRATORI, Fellow

SUMMARY In this paper, we consider a flexible method for designing n-entities communication protocols and services. The proposed technique considers alternative and parallel composition of service specifications and protocol specifications, where \( n \geq 2 \). The specifications are represented in Basic LOTOS, which is a Formal Description Technique (FDT). We use the weak bisimulation equivalence \( (\approx) \) to represent the correctness properties between the service specification and the protocol specification.

key words: service, protocol, FDT, alternative composition, parallel composition

1. Introduction

The number of people using computer and telecommunication networks has been increasing rapidly than experts anticipated in the last decade. Due to this increase, demands for various communication services and protocols have been increasing and changing more rapidly than developers can cope up with. What is most desirable for communication protocol engineers is a flexible method for developing communication services and protocols in a short period of time. In this paper, we propose a flexible method for designing n-entities communication protocols and services by combining service and protocol components.

There are mainly three approaches in designing communication protocols: (i) a top-down approach [1], [2], in which a service specification is decomposed into protocol specifications; (ii) a bottom-up approach [3], in which a protocol is designed without consideration of its service specification and, (iii) a compositional approach [4], [5], in which protocol components are designed and validated and then are combined to obtain a desired protocol/service specification. Due to complexities and functional behaviours of protocols, and demands for development of protocols in a short period of time, protocol designers are encouraged by the compositional approach.

Several compositional methods have been proposed in recent years. They can be broadly divided into three categories: (i) composition of service specifications and then decomposing the composite service specification into protocol specifications [6], [7], (ii) composition of protocol specifications [4], [5], [8], and (iii) composition of both service specifications and protocol specifications simultaneously [9], [10].

Most of the literatures in the compositional approach have concentrated on composition of only two communicating entities to obtain a protocol with only two communicating entities. However, there are many protocols such as broadcast, multicast, and group communication protocols where more than two communicating entities take part in a communication.

In this paper, we study the composition of n service specifications and corresponding n protocol specifications to obtain a composite service specification and a corresponding composite protocol specification containing \( x \) \((x \geq 2)\) communicating entities with alternative or concurrent functions. In other words, we combine n service and protocol components alternatively or concurrently. A protocol with alternative functions will execute one function at a time. Once a function is chosen for execution, other functions cannot be chosen. For example, a protocol with read-write functions that reads data or writes data to a node can either read the data from the node or write the data to the node, i.e. execution of one function is chosen at a time. On the other hand a protocol with concurrent functions will execute functions concurrently. If the above protocol is designed such a way that the reading and the writing data to a node are concurrent functions, then reading and writing functions will be executed concurrently or parallelly, i.e. both functions can be executed at the same time, in any order or independently from each other.

For a given service specification and a corresponding protocol specification, a correctness between them should be defined. For example, a protocol should be able to execute all the events defined in its service specification. In this paper, the correctness between a protocol specification and its service specification is represented by weak bisimulation equivalence [11]. The weak bisimulation equivalence between a service and a
protocol specification basically means that, from an external observer’s point of view, the actions that occur at SAPs (Service Access Points) of the protocol specification are indistinguishable from the actions that occur at SAPs of the protocol specification. The existence of weak bisimulation equivalence relation between a service and a protocol specification ensures that the protocol specification satisfies the logical properties such as free from deadlocks and unspecified reception. This is due to the fact that the protocol executes all the events that are defined in its service specification. If there were deadlocks or unspecified receptions in the protocol specification then all the actions at SAPs of the protocol specification would not be executed. This would mean that the service and the protocol is not bisimilar.

In this paper, for given n service components and n corresponding protocol components which satisfy the weak bisimulation equivalence, the composite service and protocol that we obtain by combining the components also satisfy the weak bisimulation equivalence.

We use LOTOS [11], which is an FDT based on process algebra, as the specification language for specifying the services and the protocols.

The paper is organized as follows. In Sect. 2, we briefly describe the LOTOS. In Sect. 3, we describe a communication model. In Sect. 4, we describe the detail of our composition method. In Sect. 5, we present some related works and finally in Sect. 6, we conclude the paper.

2. LOTOS

LOTOS (Language Of Temporal Ordering Specification) [11] is an FDT developed by ISO for the formal description of distributed systems. A system specified in LOTOS consists of a number of processes interacting with each other. A process in LOTOS is considered as an abstract entity which is capable of performing internal actions represented by i and communicates with other processes via communication actions by synchronizing at points called gates. The process behaviour expression is expressed in terms of temporal ordering of its actions. There are various LOTOS operators to structure the behaviour expression of a system. Operators we use in this paper are shown in Table 1.

“stop” denotes the inaction of the specified process.

“exit” denotes a process which performs the successful termination action δ and becomes stop.

“||” denotes the alternative composition of P1 and P2 where it is ready to behave as P1 or as P2.

“||” denotes the behaviour expression behaving as P whose actions in H are hidden from its environment. For the environment the hidden actions are the same as the i actions.

“||” denotes the parallel composition of P1 and P2 where H specifies the list of synchronization actions. An action listed in H or δ can be executed as a common action of P1 and P2 while other actions are executed independently. When H is empty i.e. [], it is usually represented by ||.

The process instantiation is like the invocation of a procedure in a programming language. The process instantiation P[g1, . . . , gn] = B as a set of actions, Acts as a set of sequences of actions in Act and 1 as zero or more i actions. For a process P, we define Act(P) as the set of actions which P can execute.

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
<th>Axioms and inference rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>inaction</td>
<td>stop</td>
<td>exit δ → stop</td>
</tr>
<tr>
<td>successful termination</td>
<td>exit</td>
<td>a; P ⊳ a; P</td>
</tr>
<tr>
<td>action prefix</td>
<td>P1</td>
<td></td>
</tr>
<tr>
<td>choice</td>
<td>P1</td>
<td></td>
</tr>
<tr>
<td>hiding</td>
<td>hide H in P</td>
<td>hide H in P</td>
</tr>
<tr>
<td>parallel composition</td>
<td>P1</td>
<td></td>
</tr>
<tr>
<td>process instantiation</td>
<td>P[g1, . . . , gn] = B</td>
<td>P[g1, . . . , gn] = B</td>
</tr>
</tbody>
</table>

Table 1 Basic LOTOS syntax and semantics.
Definition 2.1 (LTS): A labeled transition system L is a quadruple \((S, A, T, s_0)\), where \(S\) is a nonempty set of states, \(A\) is a subset of \(Act\), \(T \subseteq S \times A \times S\) is a transition relation, \(s_0 \in S\) is the initial state of \(L\). \(\square\)

Based on the operational semantics given by the transition systems, a bisimulation relation, is defined in Definition 2.2. In the definition, if \(t = a_1 \cdots a_n \in (Act - \{i\})^*\), then \(s \Rightarrow s'\) stands for \(s \vdash r \vdash s_1 \vdash r \vdash s_2 \vdash r \vdash \ldots \vdash r \vdash s_n \vdash r \vdash s'\). The notation \(s \Rightarrow s'\) represents the transition of \(s\) to \(s'\) when \(s\) executes the action \(a\).

Definition 2.2 (Bisimulation): Let \(L_1 = (S_1, A_1, T_1, s_{10})\) and \(L_2 = (S_2, A_2, T_2, s_{20})\) be labeled transition systems. A binary relation \(R \subseteq S_1 \times S_2\) is a (weak) bisimulation relation if \((s_1, s_2) \in R\) implies that, for all \(t \in (Act - \{i\})^*\),

(1) if \(s_1 \Rightarrow s_1'\) for some \(s_1'\), then \(s_2 \Rightarrow s_2'\) and \((s_1', s_2') \in R\) for some \(s_2'\);

(2) if \(s_2 \Rightarrow s_2'\) for some \(s_2'\), then \(s_1 \Rightarrow s_1'\) and \((s_1', s_2') \in R\) for some \(s_1'\).

\(L_1\) and \(L_2\) are weak bisimulation equivalent (observation equivalent) written \(L_1 \approx L_2\) iff there exists a weak bisimulation relation \(R \subseteq S \times S\) with \(S = S_1 \cup S_2\) and \((s_{10}, s_{20}) \in R\). \(\square\)

For simplicity, we will write a behaviour expression in an equation form in the following sections.

3. Service and Protocol Models

A service is a black box (Fig. 1(a)) and its specification is specified by the temporal ordering of events that are executed at SAPs. For simplicity, we index each service access point by a numerical value. An event that is executed at a SAP is denoted by \(\text{event}_i\text{name}_j\) where \(\text{event}_i\text{name}\) is the service primitive and the node is the SAP at which the interaction takes place (e.g. \(a^1\) for event \(a\) at node 1).

Definition 3.1 \((A^i)\): \(A^i\) is a set of actions that are executed at the node \(i\) (SAP \(i\)). \(\square\)

Assumption 1: We assume that each service and communicating entity specifications are in monolithic style \([12]\), i.e. \(S = \sum A^i_i\) where each \(A^i_i\) is either process identifier or an expression in an action-prefix form. \(\square\)

For instance, for \(I = 1, 2\), we have \(S = a^1_1; A^1_1; A_2\) and furthermore if \(A_1 = b^2_2; \text{stop}\) and \(A_2 = b^2_2; \text{stop}\) then \(S = a^1_1; b^2_2; \text{stop}; a^1_2; b^2_2; \text{stop}\).

A protocol is a white box as shown in Fig. 1(b). The protocol specification means the communicating entities which reside at each node and communicate with each other via underlying medium. The entity \(E_1(E_2)\) puts a message to be sent to \(E_2(E_1)\) in the Medium by synchronizing with the Medium at the gate \(s_1(s_2)\). The Medium delivers the message to \(E_2(E_1)\) by synchronizing at the gate \(r_2(r_1)\). The protocol specification therefore is expressed as \((E_1||E_2)||s_1, r_1, s_2, r_2)\).

Definition 3.2 \((RR_i(SS_i))\): \(RR_i(SS_i)\) is a set of actions that are executed at the gate \(r_i(s_i)\), i.e. a set of receiving (sending) actions of messages from(to) the medium at the node \(i\). \(\square\)

In order to write the expression more concisely, we extend the synchronization gate to the structured gate of the form \(gate \text{message}\). The intended meaning is that, the synchronization takes place on message, \text{message}, at the gate, \(\text{gate}\). When the synchronization takes place the message is said to be exchanged between an entity and the Medium.

Assumption 2: We assume that a \(\text{Medium}\) is a FIFO queue with infinite capacity and there exists a medium from each node “\(i\)” to any other node “\(j\)”.

Assumption 3: We assume that the specification of \(E_i\) and \(E_j\) is derived from the service specification using decomposition method \([13]\) because in \([13]\) the semantical relationship between a process and its decomposition is weak bisimulation equivalence which serves our purpose.

For instance, for the service specification, \(S = a^1_1; b^2_2; \text{stop}; a^1_2; b^2_2; \text{stop}\), we have,

\[E_1 = a^1_1; s_1!ma^1_1; r_1!mb^2_1; \text{stop} \]
\[b^2_2; s_1!ma^2_1; r_1!mb^2_1; \text{stop} \]
\[E_2 = r_2!ma^1_1; b^2_2; s_2!mb^2_1; \text{stop} \]
\[r_2!ma^2_1; b^2_2; s_2!mb^2_1; \text{stop}. \]

The basic outline of the decomposition method is that all the events that occur sequentially at the node \(i\) in the service specification are sequentially arranged in \(E_i\). Whenever an event at the node \(i\) is followed by an event at the node \(j\) in the service specification, this change of execution of events between node \(i\) and node \(j\) is expressed by the send message event at \(E_i\) (e.g. \(s_1!ma^1_1\) in the above example) and the receive event of the same message at \(E_j\) (e.g. \(r_2!ma^2_1\) in the above example).

4. Composition Approach

In this section, we explain the composition of service and protocol specifications. For the notational convenience we introduce the following notations.

![Fig. 1](image-url)
We introduce the notation \[|B|\] as choice between a nonempty set of entities \(B\) where \(||E|\rangle = E\) and \(|\{E_1, E_2, \ldots, E_n\}| = E_1|E_2|\ldots|E_n\). Similarly we introduce the notation \[|\cdot|\] as parallel composition of nonempty set of entities \(B\) where \[|\cdot|\] is applied until \[|\cdot|\] can execute.

Informally, the composition approach is as follows.

Given the service specifications \(S_1, S_2, \ldots, S_n\) and corresponding protocol specifications, say,

\[
\begin{align*}
(E_i || E_j) &| [s_i, r_i, s_j, r_j]| \text{Medium}, \\
(F_k || F_l) &| [s_k, r_k, s_l, r_l]| \text{Medium}, \\
\vdots &
\end{align*}
\]

we combine service specifications and protocol specifications together so that the composite service specification is weak bisimulation equivalence with the composite protocol specification, that is,

\[
S_1 \ast S_2 \ast \ldots \ast S_n \approx \text{hide } s_1, r_1, s_j, r_j, \ldots, s_n, r_n, s_z, r_z \text{ in } \\
(E_i || E_j) | [s_i, r_i, s_j, r_j]| \text{Medium}, \\
(F_k || F_l) | [s_k, r_k, s_l, r_l]| \text{Medium}, \\
\vdots
\]

Before going in detail, we present an example of composition of connection establishment service specifications and protocol specifications as shown in Fig. 2, where \(S_1, S_2\) and \(S_3\) are service specifications and the composite service specification is \(S_1 \ast S_2 \ast S_3 \approx \text{hide } s_1, r_1, s_j, r_j, \ldots, s_n, r_n, s_z, r_z \text{ in } \\
(E_i || E_j) | [s_i, r_i, s_j, r_j]| \text{Medium}, \\
(F_k || F_l) | [s_k, r_k, s_l, r_l]| \text{Medium}, \\
\vdots
\)

\[
\begin{align*}
(S_1 \ast S_2 \ast S_3) \approx \text{hide } s_1, r_1, s_j, r_j, \ldots, s_n, r_n \text{ in } \\
(E_i || E_j) | [s_i, r_i, s_j, r_j]| \text{Medium}, \\
(F_k || F_l) | [s_k, r_k, s_l, r_l]| \text{Medium}, \\
\vdots
\end{align*}
\]

\[
\begin{align*}
S_1 \ast S_2 \ast \ldots \ast S_n \approx \text{hide } s_1, r_1, s_j, r_j, \ldots, s_n, r_n \text{ in } \\
(E_i || E_j) | [s_i, r_i, s_j, r_j]| \text{Medium}, \\
(F_k || F_l) | [s_k, r_k, s_l, r_l]| \text{Medium}, \\
\vdots
\end{align*}
\]
In order to distinguish which poll message is intended to which node we use poll$^i_j$ meaning the poll message is from node $i$ to node $j$.

In this paper, we propose an alternative and a parallel composition of $n$ service and protocol specifications using LOTOS choice and parallel operators respectively.

We introduce the following notation $init(B)$ first. $init(B)$: It is the set of actions which $B$ can execute initially and is defined as:

\[
init(B) = \{ a \in Act(B) \cup \{ \delta \} | B \xrightarrow{a} \}
\]

**Assumption 4**: We assume in the rest of the paper that the service and protocol specifications do not contain internal actions $i$ and initial actions of a service specification belong to only one node.

### 4.1 Alternative Composition

When two or more specifications are to be combined whose functionality are to be alternative, we combine them as alternative functionality. In this paper, we use LOTOS choice operator to combine specifications alternatively. The composite service and protocol specifications might not be bisimilar. In such case, we introduce a polling mechanism, as shown in Fig. 2(e), in the composite protocol specification.

The reason for introducing the polling messages is that, without it, we have to either put different restrictions on the service and protocol specifications or modify them. Putting many restrictions on the specifications constrains the applicability of the composition method and modifications of the specifications lose the original behaviour of service and protocol components. It is essential that the properties and behaviour of the individual components are preserved in the resulting product of the composition method. For example, in a telecommunication system, an introduction of any new services should not affect or be affected by any existing services or change their original behaviour or properties.

**Algorithm 4.1 (Alternative):**

Let $S_1, S_2, \ldots, S_n$ be $n$ given service specifications and

\[
(E_1||E_2)[s_1, r_1, s_2, r_2]|Medium,
(F_2||F_3)[s_2, r_2, s_3, r_3]|Medium \text{ and } G_1||G_4[s_1, r_1, s_4, r_4]|Medium
\]

are corresponding protocol specifications respectively. The specifications are shown in LTS forms.

In Fig. 2(a), a communicating entity $E_1$ at the node 1 sends connection request to $E_2$ which is at the node 2.

In Fig. 2(b), a communicating entity $F_2$ at the node 2 sends connection request to $F_3$ which is at the node 3.

In Fig. 2(c), a communicating entity $G_4$ at the node 4 sends connection request to $G_1$ which is at the node 1.

We combine these component service and protocol specifications so that the composite service and protocol specifications contain the alternative functions. We use LOTOS choice operator to combine components as alternative functions. The resulting composite service and protocol specifications are shown in Fig. 2(d). It can be easily seen in Fig. 2(d) that the correctness of composite service and protocol specifications is not satisfied. For example, when $E_1||G_1$ at the node 1 sends connection request to $E_2||F_2$ at the node 2 and at the same time if $G_4$ at the node 4 sends connection request to $E_1||G_1$ at the node 1, then a deadlock occurs, i.e. the node 1 cannot receive the connection request sent by the node 4. Consequently the execution of the remaining events at the node 4 stops. This problem occurs because of the distributed choice in execution of events at different nodes, i.e. more than one node have initial actions that are executed at SAPs or are sending actions. Similar scenario can be noticed when the node 1 and the node 2 send connection request at the same time. Note that the composite protocol specification has 4 communicating entities whereas each component protocol has 2 communicating entities.

In order to solve this problem, we introduce a polling mechanism in the composite protocol specifications, as shown in Fig. 2(e) where $Q_1, Q_2, Q_3$ and $Q_4$ are the resulting composite entities obtained after the introduction of the polling messages. In the polling mechanism, an entity sends a poll message to another entity (in Fig. 2(e) $Q_3$ sends say poll$^3_2$ to $Q_2$) which after receiving the poll message either sends back a poll message (in Fig. 2(e) $Q_2$ receives poll$^3_2$ and sends back poll$^2_1$ to $Q_1$) or executes its other actions. By introducing the polling mechanism we can make the initial actions of one of the entities as only sending actions and the actions that occur at its SAP, and the initial actions of the rest of the other entities as only receiving actions thereby solving the problem of the distributed choice. In Fig. 2(e) the initial actions of $Q_1$ are only sending actions and the actions that occur at the SAP 1, and the initial actions of $Q_2, Q_3$ and $Q_4$ are only receiving actions.
(E_i[[E_j]|s_i,r_i,s_j,r_j]|Medium,
S_2 \approx \text{hide } s_k,r_k,s_l,r_l \text{ in }
(F_k[[F_l]|s_k,r_k,s_l,r_l]|Medium,
.....
S_n \approx \text{hide } s_y,r_y,s_z,r_z \text{ in }
(Z_y[[Z_z]|s_y,r_y,s_z,r_z]|Medium,

Then combine service specifications alternatively, i.e.,
S_1[S_2[...S_n]

Combine entities at each node alternatively according to the following if-then-else.

if init(S_1) \cup init(S_2) \cup ... \cup init(S_n) \subseteq A^t \text{ for some } t,
\text{then (combine all the entities belonging to a node together, no need for polling)}

((B_i|[B_j|[...[B_k]|B_l]|B_m)|[B_n]|B_{n+1}|[B_{n+2}])

\text{Medium},

where B_i, B_j, B_k, B_l, ... B_n \text{ and } B_{n+1} \text{ are set of entities at nodes i, j, k, l, ...} y \text{ and } z \text{ respectively.}

else (combine all the entities belonging to a node together at first and then use polling mechanism.)

(Q_i|[Q_j|[...[Q_k]|Q_l]|...Q_{n-1}|Q_n)

\text{Medium},

where

Q_i = K_i[\sum\{s_i,poll_i^1; (L_i[]
\sum\{r_i,poll_i^2; Q_i|b \in [j,k,l,...,y,z]\})
\text{ for some } t \}

Q_c = L_c[\sum\{r_i,poll_i^2; K_c|s_c,poll_c^2; Q_c,
\text{ for some } t \}

and

K_m = \tilde{B}_m \text{ when } init([\tilde{B}_m] \subseteq A^m \text{ and }
L_m = \tilde{B}_m \text{ when } init([\tilde{B}_m] \subseteq RR_m,
\text{ for some } m \in [i,j,k,l,...,y,z]}

(see Definition 3.2 for RR)

\text{Theorem 4.1: Let } S_1, S_2, ..., S_n \text{ be } n \text{ given service specifications and }

(E_i|[E_j]|s_i,r_i,s_j,r_j]|Medium,

(F_k|[F_l]|s_k,r_k,s_l,r_l]|Medium,

.....

(Z_y|[Z_z]|s_y,r_y,s_z,r_z]|Medium,

be corresponding given protocols respectively where

E_i, E_j, F_k, F_l, ... Z_y, Z_z \text{ are entities at nodes i, j, k, l, ... y,} z \text{ respectively, and the service and the protocol specification satisfy the bisimulation relation, i.e.,}

S_1 \approx \text{hide } s_i,r_i,s_j,r_j \text{ in }

(E_i|[E_j]|s_i,r_i,s_j,r_j]|Medium,

S_2 \approx \text{hide } s_k,r_k,s_l,r_l \text{ in }

(F_k|[F_l]|s_k,r_k,s_l,r_l]|Medium,

.....

S_n \approx \text{hide } s_y,r_y,s_z,r_z \text{ in }

(Z_y|[Z_z]|s_y,r_y,s_z,r_z]|Medium,

Then

if init(S_1[S_2[...S_n] \subseteq A^t \text{ for some } t,
S_1[S_2[...S_n] \approx \text{hide } s_i,r_i,s_j,r_j,s_k,r_k,s_l,r_l,...,s_y,r_y,s_z,r_z \text{ in }

((B_i|[B_j|[...[B_k]|B_l]|B_m)|[B_n]|B_{n+1}|[B_{n+2}])

\text{Medium},

where B_i, B_j, B_k, B_l, ... B_n \text{ and } B_{n+1} \text{ are set of entities at nodes i, j, k, l, ...} y \text{ and } z \text{ respectively.} \quad (*1*)

if init(S_1[S_2[...S_n] \nsubseteq A^t \text{ for some } t,
S_1[S_2[...S_n] \approx \text{hide } s_i,r_i,s_j,r_j,s_k,r_k,s_l,r_l,...,s_y,r_y,s_z,r_z \text{ in }

(Q_i[Q_j[Q_k[...Q_l]|Q_{n-1}|Q_n)

\text{Medium},

where Q_i, Q_j, Q_k, Q_l, ... Q_{n-1}, Q_n \text{ are defined as in Algorithm 4.1.} \quad (*2*)

\text{Proof: The proof can be done using expansion theorem [11]. The proof for (*1*) is straightforward as we don’t need to introduce polling messages. We show the proof for (*2*).}

Let \text{RP} = Q_i[Q_j[Q_k[...Q_l]|Q_{n-1}|Q_n)

\text{RP} = [P_i[...[P_j[...[P_k[...[P_l[...[P_y[...[P_z)]

\text{where } P_i, P_j, P_k, P_l, ... P_y, P_z \text{ are the corresponding set of protocols of the set of service specifications } S_i, S_j, S_k, S_l, ... S_y, S_z \text{ whose initial actions are at the node i, j, k, l, }...y, z \text{ respectively. And also according to the expansion theorem communicating entities of a given service specification will communicate with each

(see Definition 3.2 for RR)
other only.

Thus when $S_i = \{S_1\}$ = $S_1$ then
$$P_i = \{(E_i|||E_j)||s_i, r_i, s_j, r_j\}|Medium\}$$
$$= (E_i|||E_j)||s_i, r_i, s_j, r_j\]|Medium\}$$
as shown as given statement above.

If $X$ and $Y$ are processes, and $T = X||Y||i:T$, then $T \approx X||Y$, that is $\approx$ ignores internal loops [14]. Since $RP$ makes internal loops and each $P$ is the corresponding protocol of the $S$ and are bisimilar with each other we can say that
$$S_1[|S_2|] \ldots [|S_n| \approx$$

\textbf{hide} $s_i, r_i, s_j, r_j, s_k, r_k, s_l, r_l, \ldots, s_y, r_y, s_z, r_z$ \textbf{in} $\{(RP)\}|s_i, r_i, s_j, r_j, s_k, r_k, s_l, r_l, \ldots, s_y, r_y, s_z, r_z\]|Medium$.

The above general proof can be visualized clearly with the following specific example proof. Let $S_1, S_2, \ldots, S_n$ be $n$ service specifications where
$$S_1 = \sum\{a_i^1; A_u\} u \in U\},$$
$$S_2 = \sum\{b_i^1; B_v\} v \in V\},$$
$$\ldots,$$
$$S_n = \sum\{c_i^1; C_w\} w \in W\},$$
and
$$\{(E_i|||E_j)||s_i, r_i, s_j, r_j\}|Medium\},$$
$$\{(F_k|||F_l)||s_k, r_k, s_l, r_l\}|Medium\},$$
$$\ldots,$$
$$\{(Z_y|||Z_z)||s_y, r_y, s_z, r_z\}|Medium\}$$
be corresponding protocol specification respectively where
$$E_i = \sum\{a_i^1; s_i!ma_u^1; E_{iua}\} u \in U\},$$
$$E_j = \sum\{b_i^1; s_j!ma_u^1; E_{jua}\} u \in U\},$$
$$F_k = \sum\{b_i^1; s_k!mb_k^1; F_{kuv}\} v \in V\},$$
$$F_l = \sum\{r_l!mb_l^1; F_{lvw}\} v \in V\},$$
$$\ldots,$$
$$Z_y = \sum\{c_i^1; s_y!mc_y^1; Z_{yvw}\} w \in W\},$$
$$Z_z = \sum\{r_z!mc_z^1; Z_{zvw}\} w \in W\}.$$

Let the node $i$ be the initiator of the sending poll messages and the rest of the nodes be the initiators of receiving poll messages, then we have
$$Q_i = K_i[:|\sum\{s_i!poll^m_i(); (L_i\}$$
$$\sum\{r_i!poll^m_i(); Q_i|m \in M, M = \{j, k, l, \ldots, y, z\}\}$$
$$\sum\{m \in M, M = \{j, k, l, \ldots, y, z\}\}$$
Since $K_i = E_i$ and $L_i$ is empty i.e. does not have any entities, we have
$$Q_i = \sum\{a_i^1; s_i!ma_u^1; E_{iua}\} u \in U\}$$
$$|| s_l!poll_l^1; X || s_l!poll_l^1; X || s_l!poll_l^1; X$$
$$|| \cdots || s_l!poll_l^1; X || s_l!poll_l^1; X$$
$$\ldots$$

Where
$$X = r_i!poll_l^1; Q_i || r_i!poll_l^1; Q_i || r_i!poll_l^1; Q_i$$
$$|| \cdots || r_i!poll_l^1; Q_i || r_i!poll_l^1; Q_i$$

Similarly, since $K_j, L_k, K_l, L_y, K_z$ are empty we have
$$Q_j = \sum\{r_j!ma_u^1; E_{jua}\} u \in U\}|| r_j!poll_l^1; s_j!poll_l^1; Q_j$$
$$Q_k = \sum\{b_k^1; s_k!ma_k^1; F_{kuv}\} v \in V\}|| s_k!poll_l^1; Q_k$$
$$Q_l = \sum\{r_l!ma_k^1; F_{luv}\} v \in V\}|| r_l!poll_l^1; s_l!poll_l^1; Q_l$$
$$\cdots$$
$$Q_y = \sum\{c_y^1; s_y!ma_y^1; Z_{yvw}\} w \in W\}|| s_y!poll_y^1; Q_y$$
$$Q_z = \sum\{r_z!ma_z^1; Z_{zvw}\} w \in W\}|| r_z!poll_z^1; s_z!poll_z^1; Q_z$$

Let $P$ be the composite protocol then we have
$$P = \textbf{hide} s_i, r_i, s_j, r_j, s_k, r_k, s_l, r_l, \ldots, s_y, r_y, s_z, r_z \textbf{in}$$
$$\{(Q_i|||Q_j)|||\cdots|||(Q_y|||Q_z)\}$$
$$\{(s_i, r_i, s_j, r_j, s_k, r_k, s_l, r_l, \ldots, s_y, r_y, s_z, r_z\}||Medium\}$$
$$= \sum\{a_i^1;i:i: \textbf{hide} s_i, r_i, s_j, r_j \textbf{in}(E_{iua}|||E_{jua})$$
$$\{(s_i, r_i, s_j, r_j, s_k, r_k, s_l, r_l, \ldots, s_y, r_y, s_z, r_z\}||Medium\}$$
$$\ldots$$
$$\textbf{in}$$

According to the fact, if $X$ and $Y$ are processes, and $T = X||Y||i:T$, then $T \approx X||Y$, that is $\approx$ ignores internal loops [14]. Thus we can write that $P \approx P'$ where
$$P' = \sum\{a_i^1;i:i: \textbf{hide} s_i, r_i, s_j, r_j \textbf{in}(E_{iua}|||E_{jua})$$
$$\{(s_i, r_i, s_j, r_j, s_k, r_k, s_l, r_l, \ldots, s_y, r_y, s_z, r_z\}||\text{Medium}\}$$
$$\ldots$$
$$\ldots$$

Now if the expansion is continued, we can see that $S_1[|S_2|] \ldots [|S_n| \approx P$.  

\subsection{4.2 Concurrent Composition}

In distributed and parallel computing, concurrent execution of functions are very important. It is also important in protocol engineering when at a node there are functional behaviours that need to be executed concurrently. For instance, in data transfer between two nodes, each node might send and receive data concurrently using a bi-directional communication channel. When two functions are to be executed concurrently, we combine them concurrently. In this paper we use LOTOS parallel operator to do so. In parallel composition, choosing synchronization actions is important because an undesirable behaviour of the composite specification, such as the non-execution of some events, may result if the synchronization actions are improperly chosen.
In our proposed algorithm, common actions of entities are dynamically chosen as the synchronization actions. The common actions of processes, say $P_1$ and $P_2$, are denoted by $\text{Act}(P_1) \cap \text{Act}(P_2)$.

**Algorithm 4.2** (Parallel):
Let $S_1, S_2, \ldots, S_n$ be $n$ given service specifications and

\[
(E_i \mid \vec{l}_j) \mid [s_i, r_i, s_j, r_j] \mid \text{Medium}, \\
(F_k \mid F_l) \mid [s_k, r_k, s_l, r_l] \mid \text{Medium}, \\
\ldots \ldots \\
(Z_y \mid Z_z) \mid |s_y, r_y, s_z, r_z| \mid \text{Medium},
\]

be corresponding given protocols respectively where

\[
E_i, E_j, F_k, F_l, \ldots, Z_y, Z_z \text{ are entities at nodes } \]

\[
i, j, k, l, \ldots, y, z \text{ respectively, and the service and the protocol specification satisfy the bisimulation relation, i.e.,}
\]

\[
S_1 \approx \text{hide } s_i, r_i, s_j, r_j \text{ in } \\
(E_i \mid \vec{l}_j) \mid [s_i, r_i, s_j, r_j] \mid \text{Medium}, \\
S_2 \approx \text{hide } s_k, r_k, s_l, r_l \text{ in } \\
(F_k \mid F_l) \mid [s_k, r_k, s_l, r_l] \mid \text{Medium}, \\
\ldots \ldots \\
S_n \approx \text{hide } s_y, r_y, s_z, r_z \text{ in } \\
(Z_y \mid Z_z) \mid [s_y, r_y, s_z, r_z] \mid \text{Medium},
\]

Then combine $n$ service specifications parallelly, i.e.,

\[
[[\cdots][S_1, S_2, \ldots, S_n]].
\]

Combine entities at each node parallelly, i.e.,

\[
([\cdots][B_1])([\cdots][B_2])([\cdots][B_3])([\cdots][B_4]) \\
([\cdots][B_1])([\cdots][B_2])([\cdots][B_3])([\cdots][B_4])
\]

where $B_1, B_2, B_3, \ldots, B_4$ are sets of entities at nodes $i, j, k, l, \ldots, y$ and $z$ respectively.

We use the action lists defined in Definition 4.1 for the proof of Theorem 4.2.

**Definition 4.1:** Let $S$ be a service specification and $E_1$ and $E_2$ be corresponding entity specifications, then

1. $L(S)$ is the set of all actions that $S$ offers to its environment.
2. $\text{Obs}(E_1) / \text{Obs}(E_2)$ is the set of all actions that $E_1 / E_2$ offers to the same environment as $S$, i.e. $\text{Obs}(E_1) \subseteq S$ and $\text{Obs}(E_2) \subseteq S$.

From 1. and 2. of the Definition 4.1, we have

$L(S) = \text{Obs}(E_1) \cup \text{Obs}(E_2)$. $L(E_1)$ is the set of all actions $E_1$ offers to its environment, i.e. $\text{Obs}(E_1) \subset L(E_1)$.

**Theorem 4.2:** Let $S_1, S_2, \ldots, S_n$ be $n$ given service specifications and

\[
(E_i \mid \vec{l}_j) \mid [s_i, r_i, s_j, r_j] \mid \text{Medium}, \\
(F_k \mid F_l) \mid [s_k, r_k, s_l, r_l] \mid \text{Medium}, \\
\ldots \ldots \\
(Z_y \mid Z_z) \mid [s_y, r_y, s_z, r_z] \mid \text{Medium},
\]

be corresponding given protocols respectively where

\[
E_i, E_j, F_k, F_l, \ldots, Z_y, Z_z \text{ are entities at nodes } \]

\[
i, j, k, l, \ldots, y, z \text{ respectively, and the service and the protocol specification satisfy the bisimulation relation, i.e.,}
\]

\[
S_1 \approx \text{hide } s_i, r_i, s_j, r_j \text{ in } \\
(E_i \mid \vec{l}_j) \mid [s_i, r_i, s_j, r_j] \mid \text{Medium}, \\
S_2 \approx \text{hide } s_k, r_k, s_l, r_l \text{ in } \\
(F_k \mid F_l) \mid [s_k, r_k, s_l, r_l] \mid \text{Medium}, \\
\ldots \ldots \\
S_n \approx \text{hide } s_y, r_y, s_z, r_z \text{ in } \\
(Z_y \mid Z_z) \mid [s_y, r_y, s_z, r_z] \mid \text{Medium},
\]

Then

\[
[[\cdots][S_1, S_2, \ldots, S_n]] \approx \\
\text{hide } s_i, r_i, s_j, r_j, \ldots, s_y, r_y, s_z, r_z \text{ in } \\
(E_i \mid \vec{l}_j) \mid [s_i, r_i, s_j, r_j, \ldots, s_y, r_y, s_z, r_z] \mid \text{Medium},
\]

where $B_1, B_2, B_3, \ldots, B_y$ and $B_z$ are set of entities at nodes $i, j, k, l, \ldots, y$ and $z$ respectively.

**Proof:** The proof can be done by expanding of behaviour expressions similar to the Theorem 4.1 or by reshuffling parallel compositions. We will use the latter as the former will result a large behaviour expression. First we will show the proof for a few service and protocol components and then show what the result would be for $n$ components if the proof technique is continuously applied.

Let us assign arbitrary node numbers to entities, say $E_1$ and $E_2$, $F_1$ and $F_2$, and $Z_2$ and $Z_3$ of service specifications $S_1$, $S_2$ and $S_3$ respectively. Combining the service specifications, we obtain

\[
S_1 [[K0] [[S2] [[K1] [[S2]]]],
\]

where

\[
K0 = L(S_1) \cap (L(S_2) \cup L(S_3)) = (\text{Obs}(E_1) \cup \text{Obs}(E_2)) \cap ((\text{Obs}(F_1) \cup \text{Obs}(F_2)) \cup (\text{Obs}(Z_2) \cup \text{Obs}(Z_3)))
\]

and

\[
K1 = L(S_2) \cap L(S_3) = ((\text{Obs}(F_1) \cup \text{Obs}(F_2)) \cap (\text{Obs}(Z_2) \cup \text{Obs}(Z_3)))
\]

Combining the entities we obtain

\[
E_1 [[H0] [[H1] [[F_1] [[F_2] [[H2] [[Z_2])]]]]]]) \cup \text{Obs}(Z_3),
\]

where

\[
H0 = \text{Obs}(E_1) \cap \text{Obs}(F_1), \\
H1 = \text{Obs}(E_2) \cap (\text{Obs}(F_2) \cup \text{Obs}(Z_2)) \text{ and } \\
H2 = \text{Obs}(F_2) \cap \text{Obs}(Z_2).
\]

Since
$L(E_1) \cup H1 = \emptyset, L(F_1) \cup H1 = \emptyset, L(E_2) \cup H0 = \emptyset$ and $L(F_2[H2] Z_2) \cup H2 = \emptyset$, we can reshuffling the parallel composition [12] and obtain

$(E_1 \ || \ E_2) \ || (H0 \cup H1) \ || (F_1 \ || \ (F_2[H2] Z_2)) \ || Z_3$.

Using the property of associativity of parallel composition [12], we can rearrange the parallel composition as

$(E_1 \ || \ E_2) \ || (H0 \cup H1) \ || (F_1 \ || \ (F_2[H2] Z_2)) \ || Z_3$.

Further reshuffling the parallel composition we obtain,

$(E_1 \ || \ E_2) \ || (H0 \cup H1) \ || (F_1 \ || \ (F_2[H2] Z_2)) \ || Z_3$.

Putting back the Medium in the specification we obtain,

$(E_1 \ || \ E_2) \ || \ [s_1, r_1, s_2, r_2] \ || \ Medium \ || (H0 \cup H1) \ || (F_1 \ || \ F_2) \ || \ [s_1, r_1, s_2, r_2] \ || \ Medium \ || (H2) \ || (Z_2 \ || \ Z_3) \ || \ [s_2, r_2, s_3, r_3] \ || \ Medium$.

It is not difficult to show that $K0 = H0 \cup H1$ and $K1 = H2$. Using the laws for hiding operator [11], [12], we obtain

$\text{hide } s_1, r_1, s_2, r_2, s_3, r_3 \ \text{in} \ (E_1 \ || \ E_2) \ || \ [s_1, r_1, s_2, r_2] \ || \ Medium \ || (H0 \cup H1) \ || (F_1 \ || \ F_2) \ || \ [s_1, r_1, s_2, r_2] \ || \ Medium \ || (H2) \ || (Z_2 \ || \ Z_3) \ || \ [s_2, r_2, s_3, r_3] \ || \ Medium = \text{hide } s_1, r_1, s_2, r_2 \ \text{in} \ (E_1 \ || \ E_2) \ || \ [s_1, r_1, s_2, r_2] \ || \ Medium \ || (H0 \cup H1) \ || (F_1 \ || \ F_2) \ || \ [s_1, r_1, s_2, r_2] \ || \ Medium \ || (H2) \ || (Z_2 \ || \ Z_3) \ || \ [s_2, r_2, s_3, r_3] \ || \ Medium$.

For composition of $n$ service specifications, i.e. $[[\cdots]](S_1, S_2, \cdots, S_n)$, and the reshuffling of parallel composition of entities will result

$(E_1 \ || \ E_2) \ || \ [s_1, r_1, s_2, r_2] \ || \ Medium \ || (H0 \cup H1) \ || (F_1 \ || \ F_2) \ || \ [s_1, r_1, s_2, r_2] \ || \ Medium \ || (H2) \ || (Z_2 \ || \ Z_3) \ || \ [s_2, r_2, s_3, r_3] \ || \ Medium.

$\approx S_1[[K0]](S_2[K1]|S_2)$.

For composition of $n$ service specifications, i.e. $[[\cdots]](S_1, S_2, \cdots, S_n)$, and the reshuffling of parallel composition of entities will result

$(E_1 \ || \ E_2) \ || \ [s_1, r_1, s_2, r_2] \ || \ Medium \ || (H0 \cup H1) \ || (F_1 \ || \ F_2) \ || \ [s_1, r_1, s_2, r_2] \ || \ Medium \ || (H2) \ || (Z_2 \ || \ Z_3) \ || \ [s_2, r_2, s_3, r_3] \ || \ Medium.$

From the resulting expression, it can be seen that the $\approx$ is satisfied.

5. Related Works

We briefly describe some closely related works in this section. In [1], [2], [15], specifications of $n$ protocol entities are derived from a given service specification. The approach is a specification transformation approach. The specification of a service is fixed. Any addition of a new service to an existing service specification cannot be handled and if added, though not mentioned, the transformation has to be performed all over again resulting new specifications of entities in some cases. In [16], a protocol with $n$ entities is validated and synthesized from protocol components. The approach does not consider the service specification. In [6], $n$ number of service components are combined and the resulting service specification is transformed into two protocol entities only. Though $n$ number of services are considered, the derivation results only two entities.

We measure our proposed algorithms in terms of following criterions:

(i) formalism: our proposed composition method is based on LOTOS whereas other algorithms for composition of protocols are mainly based on CFSMs. In CF-SMs, the problem of state explosion can arise whereas in LOTOS such a problem does not occur as the states are not explicitly stated.

(ii) composition techniques: our technique combines both $n$ service and corresponding $n$ protocol specifications simultaneously to obtain $n$ entities communication protocols and services, whereas other compositions techniques, mentioned in Sect. 1 and the above, combine only protocol or only service specifications. Unlike [1], [2], [15], our approach is compositional approach to obtain $n$ entities protocol and the approach re-uses specifications.

(iii) behaviour preservation: in our method, original specifications are not modified so much compared to the other methods.

6. Conclusion

In our proposed method for designing service and protocols with $n$-communicating entities, we have presented two algorithms to combine $n$ component service and protocol specifications alternatively and concurrently. We have also presented that the composition method satisfies the properties of correctness of service and protocol specifications (i.e. weak bisimulation equivalence). Our future work is to investigate how $n$ communicating entities and service components can be combined sequentially and interruptly without modifying their original behaviours significantly and also satisfy the above mentioned properties of correctness.
References


Bhed Bahadur Bista received the B.Eng. degree in Electronics from University of York, England in 1991 and the M.S. and Ph.D. degrees in Information Science from Tohoku University, Japan in 1994 and 1997 respectively. He worked as a research associate at Miyagi University from April 1997 to March 1998. At present, he is an associate professor in Faculty of Software and Information Science, Iwate Prefectural University, Japan.

His current research interests are protocol specification and synthesis, and formal description techniques. He is a member of IPSJ and IEEE.

Kaoru Takahashi joined the Research Institute of Electrical Communication, Tohoku University in 1974. He received Ph.D. degree from Tohoku University in 1992. From 1993 to 1995, he was a senior visiting researcher in the Advanced Intelligent Communication Systems Laboratories. He is now a professor in the Department of Information and Communication Engineering, Sendai National College of Technology. Currently he is doing research on formal description techniques which cover the whole process of distributed systems design. Dr. Kaoru Takahashi received the 25th anniversary memorial prize-winning paper award of IPSJ (Information Processing Society of Japan) in 1985, the 6th telecommunication advancement foundation incorporation award in 1991, and the information network research award of IEICE in 1997. He is a member of IPSJ.

Norio Shiratori received his doctorate degree at Tohoku University, joined the Research Institute of Electrical Communication (RIEC), and is now a professor. He has been engaged in research on distributed processing systems, and flexible intelligent networks. He received IPSJ Memorial Prize-Winning Paper Award in 1985, Telecommunications Advancement Foundation Incorporation Award in 1991, the Best Paper Award of ICOIN-9 in 1994, the IPSJ Best Paper Award in 1997, etc. He has been named a Fellow of the IEEE for his contributions to the field of computer communication networks. He is a member of IPSJ and IEEE.