マルチメディア通信と分散処理 70−20
（１９９５．５．２６）

# 構造化されたエンティティからのプロトコル合成

ベッド. バハドゥール. ビスタ, 富樫 敦, 白鳥 則郎
東北大学電気通信研究所 / 情報科学研究科
〒980-77 仙台市青葉区片平 2-1-1

通信システムのエンティティは通信規則 (プロトコル) の集合にもとづいて動作し, それらの動作は互いに関係しあっている. 本稿では, この関係を使い, 単一のエンティティからピアエンティティを生成することで, 構造化されたプロトコルモデルの導出アルゴリズムを提案する. アルゴリズムに入力として与えるエンティティは, 通信プロトコルの文脈において自然な, 通信プロセスの形式をとる. さらに本稿では, 合成されたプロトコルモデルが安全である, すなわち全体でデッドロックがないことと, 両方のエンティティによって双模倣されることを示す.

# Protocol Synthesis from a Structured Entity

Bhed Bahadur Bista, Atsushi Togashi and Norio Shiratori
Research Institute of Electrical Communication, Tohoku University
2-1-1, Katahira, Sendai, 980-77 Japan

Entities in communication systems behave under sets of communication rules (protocols) and their behaviours are related to each other. Using this relationship, we present a synthesis algorithm to derive a structured protocol model by generating a peer entity from a given entity automatically. The given entity is in the form of a communicating process, which is natural in the context of communication protocols. We prove that the synthesized protocol model consisting of the generated entity and the peer entity is safe, i.e. logical errors free, collectively represented as *deadlock* free and is bisimulated by both entities.

## 1 Introduction

Communication protocols are the main parts in computer networks and distributed information processing systems and play a major role for their proper operations. Formal Description Techniques (FDTs) such as Estelle [1], LOTOS [2] and SDL [3] have been developed to write such protocols precisely, correctly and unambiguously.

Behaviours of entities in communication systems are highly related to each other as they communicate with each other under certain sets of communication rules. By exploiting this relationship, it is possible and desirable to generate a peer entity (a missing entity) from a given entity. In [4], [5], a peer entity is automatically generated in LOTOS from a given entity which is specified as a communicating

process in LOTOS. However, the entities are specified in monolithic style [6], which is a expanded form of behaviour expressions. Large and complex systems are usually structured into phases and concurrent entities such a way that they are more understandable, manageable and each phase or entity can be designed separately. LOTOS unlike other FDTs has various operators such as *enabling*, *disabling* and *parallel* operators to structure behaviours of a system. See [1] for details. A system which is not structured is difficult to understand, manage and is error prone. In this paper, we propose a synthesis algorithm to derive a protocol, which is structured, by generating a peer entity from a given entity. The given entity is defined as a communicating process, which is structured, in LOTOS. The generated peer entity is structured and is in LOTOS also. Furthermore, we show that the resulting protocol is deadlock free and is bisimulated by both entities to show that the protocol is safe and maintains the behaviours of entities.

This paper is organized as follows. Following the Introduction, section 2 presents structured protocol model. Section 3 discusses the details of the proposed synthesis algorithm and presents a simple application example using the algorithm. Section 4 concludes the paper.

## 2 Structured Communication Model

### 2.1 Protocol Model

In a communication system, protocol entities provide services to their users by interacting among themselves. Protocol entities and service access points are represented in LOTOS by processes and gates respectively. An abstract communication system can be represented as in Figure 1 in which $PM$ represents *protocol model* consisting of interacting processes, $P$ and $Q$, which provide services to their respective users, $U_p$ and $U_q$.

A $PM$ is a 5-tuple $\langle P, Q, G, Gp, Gq \rangle$. $P$ and $Q$ are processes. $G$ is a gate at which $P$ and $Q$ interact to exchange messages. $G_p$ and $G_q$ are gates at which $P$ and $Q$ provide services to their users $U_p$ and $U_q$ respectively. Another view of looking at the gates is to treat them as

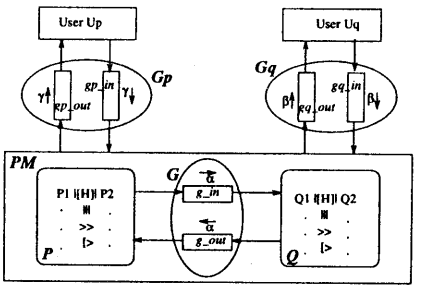


Figure 1: Structured communication model

buffers in which a process puts in a message and the other process takes out the message by synchronizing in an action. For example, $G$ is interpreted in such a way that *g_in* is the gate at which $P$ sends messages ($\overrightarrow{\alpha}$) to $Q$ (and the gate at which $Q$ receives messages ($\overrightarrow{\alpha}$) from $P$) and *g_out* is the gate at which $Q$ sends messages ($\overleftarrow{\alpha}$) to $P$ (and the gate at which $P$ receives messages ($\overleftarrow{\alpha}$) from $Q$).

In this paper, we consider both $P$ and $Q$ as structured processes as shown in Figure 1. The resulting synthesized protocol consisting of $P$ and $Q$ is also structured. We use parallel, enabling and disabling operators which are mainly used for structuring behaviours of a system. The parallel operator is used to express concurrency and constraint between processes whereas the enabling and disabling are mainly used for expressing the phases of the system.

We assume that the given entity is defined as a communicating process which is structured. The synthesis algorithm given in section 3 generates a peer entity which is also structured.

### 2.2 Communicating Processes

In order to define a communicating process, it is assumed that we have denumerable set $X$ of *process variables.*

**Definition 2.1 (communicating process)** A *communicating process* for a given entity is defined inductively by the following BNF notations:

$P ::=$ **stop** | **exit** | $(\sum_{i\in I} S_i) |||(\sum_{j\in J} R_j)$ | $x$
| **rec**$x$.$P$ | $P_1 \gg P_2$ | $P_1[> P_2$ | $P_1[[H]]P_2$ |
$P_1 ||| P_2$
$S ::= \gamma_i\downarrow; \overrightarrow{\alpha_i}; P \mid a; \gamma_i\downarrow; \overrightarrow{\alpha_i}; P \mid \gamma_i\downarrow; \overrightarrow{\alpha_i}; a; P$
$R ::= \overleftarrow{\alpha_j}; \gamma_j\uparrow; P \mid a; \overleftarrow{\alpha_j}; \gamma_j\uparrow; P \mid \overleftarrow{\alpha_j}; \gamma_j\uparrow; a; P$
where $x \in X$ and $a \in H$ □

The recursive behaviour of the given entity is expressed by the communicating process, **rec**$x$.$P$, which is interpreted as a process $x$ whose meaning is defined by the recursive equation, $x \stackrel{def}{=} P$. For simplicity, we treat only communicating processes without mutual recursion as above.

In the communicating processes, $\gamma_i\downarrow; \overrightarrow{\alpha_i}$, $a; \gamma_i\downarrow; \overrightarrow{\alpha_i}$ and $\gamma_i\downarrow; \overrightarrow{\alpha_i}; a$ are considered as *sending primitives* and $\overleftarrow{\alpha_j}; \gamma_j\uparrow$, $a; \overleftarrow{\alpha_j}; \gamma_j\uparrow$ and $\overleftarrow{\alpha_j}; \gamma_j\uparrow; a$ are considered as *receiving primitives*. The temporal orderings of $\gamma_i\downarrow$, $\overrightarrow{\alpha_i}$, $\overleftarrow{\alpha_j}$ and $\gamma_j\uparrow$ in a communicating process is natural in a sense that the communicating process receives a message ($\gamma_i\downarrow$) from its user and sends it ($\overrightarrow{\alpha_i}$) to its peer entity and, similarly receives a message ($\overleftarrow{\alpha_j}$) from the peer entity and delivers it ($\gamma_j$) to the user and then be ready for other actions.

The most difficult part in describing the communicating processes is to find out which actions belong to $H$ in generalized parallel operator, $P_1[[H]]P_2$. We realize that actions $\gamma_i\downarrow$, $\overrightarrow{\alpha_i}$, $\overleftarrow{\alpha_j}$ and $\gamma_j\uparrow$, cannot belong to $H$ as it will be unnecessary because it will mean that $P_1$ and $P_2$ will receive the same message either from user $U_p$ and send it $Q$ or vice versa. So we assume that $H$ has some actions which are local to $P_1$ and $P_2$ only, i.e. hidden from their environments $U_p$ and $Q$, and denote them by $a$. $a$ can appear in every possible places in $P$ as shown by the LTSs of $P$ in Figure 2. For simplicity, we consider Figure 2(a) only. Similarly, a communicating process for the peer entity can be defined.

### 2.3 A Safe Protocol Specification

A safe protocol specification is the one which is free from logical errors. In general, there are four types of logical errors, (1) *deadlock*, (2) *unspecified reception*, (3) *non-executable transition* and (4) *buffer overflow*, which are well

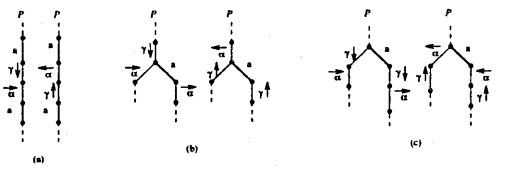

Figure 2: The temporal ordering of $a$ in $P$

discussed in FSMs model [7]. Since we do not have a real buffer (though indirectly implied), we do not consider the buffer overflow. Other three errors are collectively called as deadlock from actions execution point of view in our model, since the deadlock occurs in LOTOS when processes cannot synchronize in an action [4],[5], [8]. We formally define the deadlock in definition 2.2.

**Definition 2.2 (Deadlock)** Let $PM = P[[G]]Q$ be a protocol model. $PM$ is *deadlock* if there exist processes $P'(Q')$ and $a \in Act$ such that

(1) $P \stackrel{t}{\rightarrow} P' \stackrel{a}{\rightarrow} (Q \stackrel{t}{\rightarrow} Q' \stackrel{a}{\rightarrow})$ for some $t \in Act^*$

(2) For any $Q''(P'')$ such that $PM \stackrel{t'}{\rightarrow} P'[[G]]Q''$ $(PM \stackrel{t'}{\rightarrow} P''[[G]]Q')$, for some $t' \in Act^*$ we have $P'[[G]]Q'' \not\stackrel{a}{\rightarrow}$ $(P''[[G]]Q' \not\stackrel{a}{\rightarrow})$.

In this case, we say that deadlock *occurs* at $P'$ $(Q')$ and $a$ is the *non-executable* action[1]. □

Now, we can formalize our protocol synthesis problem as follows:
*Given*: Specification of an entity $P$, which is expressed as a communicating process.
*Problem*: Generate the specification of the peer entity $Q$, which is expressed as a communicating process also, such that

(1) $PM = P[[G]]Q$ is *deadlock* free.

(2) $P$ with an action of the form $\overleftarrow{\alpha}$ replaced by **i**; $\overleftarrow{\alpha}$ is bisimilar to **hide** $\tilde{\beta}$ **in** $P[[G]]Q$, where $\tilde{\beta}$ denotes the sequence of actions in $Q$ of the form $\beta\uparrow$ and $\beta\downarrow$.

[1]Note that $a$ must be a synchronization action.

(3) $Q$ with an action of the form $\overrightarrow{\alpha}$ replaced by **i**; $\overrightarrow{\alpha}$ is bisimilar to **hide** $\tilde{\gamma}$ **in** $P|[G]|Q$, where $\tilde{\gamma}$ denotes the sequence of actions in $P$ of the form $\gamma\uparrow$ and $\gamma\downarrow$.

In order to realize (2) and (3) we have to consider the semantical meaning of entities' behaviours. For instance, the behaviour of $P$ in (2) may be expressed in different form (i.e. $\overleftarrow{\alpha}$ is replaced by **i**; $\overleftarrow{\alpha}$) because for $P$, any message sent by user $U_q$ to $Q$ seems invisible thus non-deterministic, i.e. it is not known to $P$ which message is sent to $Q$ by $U_q$ and hence to $P$. In order to realize this non-determinism, we replace receiving actions $\overleftarrow{\alpha}$ of $P$ by $\mathbf{i};\overleftarrow{\alpha}$.

## 3 Synthesis Algorithm and Results

In this section, we present the details of our proposed algorithm and show some results.

### 3.1 Synthesis Algorithm

The basic idea of the synthesis algorithm is as follows. For each sending primitive of the given entity, we generate an appropriate receiving primitive for the peer entity. Similarly, for each receiving primitive of the given entity, we generate an appropriate sending primitive for the peer entity. In other words, when $P$ receives a message ($\gamma\downarrow$) from its user and sends it ($\overrightarrow{\alpha}$) to the peer entity, i.e. $P = \gamma\downarrow;\overrightarrow{\alpha};P'$, we generate the peer entity $Q$ such a way that it receives the message ($\overrightarrow{\alpha}$) from $P$ and delivers the message ($\beta\uparrow$) to its user, i.e. $Q = \overrightarrow{\alpha};\beta\uparrow;Q'$. Similarly, we consider other primitives of $P$ to generate appropriate primitives of $Q$. The concept is natural for communication protocols. For instance, in a simple connection establishment protocol, if given entity is defined as; $P = ConReq\downarrow;\overrightarrow{CR};$**exit**, then the generated peer entity is; $Q = \overrightarrow{CR};ConInd\uparrow;$**exit**, where $ConReq\downarrow$ is *Connection Request* service primitive, $\overrightarrow{CR}$ is the corresponding *Protocol Data Unit* and $ConInd\uparrow$ is *Connection Indication* service primitive.

In the Algorithm 3.1, *peer* is the function (procedure) whose input is a given entity $P$. *peer* checks the communication pattern of primitives of $P$ recursively and returns the corresponding primitives of $Q$. *peer* has auxiliary procedure, $aux(S)$ and $aux(R)$, which generate receiving primitives and sending primitives of $Q$ respectively. The peer entity is structurally generated if the given entity is structured. In case of the given entity consisting of sub-entities composed by enabling and disabling operators, $P_1 \gg P_2$ and $P_1[> P_2$, we transform them into $P_1 \gg nc;P_2$ and $P_1[> nc;P_2$ respectively. In the protocol model $PM$, $P$ and $Q$ synchronize in $nc$ (next phase) also. The reason for this transformation is that enabling and disabling operators are mainly used for expressing phases in a system. Here in this paper, we would like the phases of $P$ and $Q$ progress synchronously by synchronizing in $nc$ before executing the actions of the next phases, $P_2$ and $Q_2$. The advantage of this transformation is that phases of $P$ and $Q$ do not lag or advance while communicating to each other. We state the algorithm as shown below.

**Algorithm 3.1 (Synthesis Algorithm)**
**procedure** $peer(P)$
**begin**
**case** $P$ **of**
**stop: return stop**; (1)
**exit: return exit**; (2)
$x$: **return** $x$; (3)
$(\sum_{i\in I} S_i) \;|||\; (\sum_{j\in J} R_j)$ :
**return** $(\sum_{i\in I} aux(S_i)) \;|||\; (\sum_{j\in J} aux(R_j))$; (4)
**rec** $x.P$: **return rec** $x.peer(P)$; (5)
$P_1 \gg nc;P_2$:
**return** $peer(P_1) \gg nc;peer(P_2)$; (6)
$P_1[> nc;P_2$:
**return** $peer(P_1)[> nc;peer(P_2)$; (7)
$P_1|[H]|P_2$:
**return** $peer(P_1)|[H']|peer(P_2)$; (8)
$P_1 \;|||\; P_2$:
**return** $peer(P_1) \;|||\; peer(P_2)$; (9)
**end**
**procedure** $aux(S)$
**begin**
**case** $S$ **of**
$\gamma_i\downarrow;\overrightarrow{\alpha_i};P$ :
**return** $\overrightarrow{\alpha_i};\beta_i\uparrow;peer(P)$; (S1)
$a;\gamma_i\downarrow;\overrightarrow{\alpha_i};P$ :

**return** $b; \overrightarrow{\alpha_i}; \beta_i\uparrow; peer(P);$ (S2)

$\gamma_i\downarrow; \overrightarrow{\alpha_i}; a; P$ :

**return** $\overrightarrow{\alpha_i}; \beta_i\uparrow; b; peer(P);$ (S3)

**end**

**procedure** $aux(R)$

**begin**

**case** $R$ **of**

$\overleftarrow{\alpha_j}; \gamma_j\uparrow; P$ :

**return** $\beta_j\downarrow; \overleftarrow{\alpha_j}; peer(P);$ (R1)

$a; \overleftarrow{\alpha_j}; \gamma_j\uparrow; P$ :

**return** $b; \beta_j\downarrow; \overleftarrow{\alpha_j}; peer(P);$ (R2)

$\overleftarrow{\alpha_j}; \gamma_j\uparrow; a; P$ :

**return** $\beta_j\downarrow; \overleftarrow{\alpha_j}; b; peer(P);$ (R3)

**end** □

Note that $a \in H$ and $b \in H'$

### 3.2 Results

In this section, we prove that the synthesized protocol is safe, i.e. deadlock free (Theorem 3.1) and is bisimulated by both entities $P$ and $Q$ (Theorem 3.2). We omit the proofs of the theorems due to the lack of space.

**Lemma 3.1** *If $P_i|[G]|Q_i$ are deadlock free for all $i \in I$, then the protocol model, $PM = (\sum_{i\in I} \gamma_i\downarrow; \overrightarrow{\alpha_i}; P_i)|[G]|(\sum_{i\in I} \overrightarrow{\alpha_i}; \beta_i\uparrow; Q_i)$ is deadlock free.* □

**Lemma 3.2** *Let $P_1$ and $P_2$ be processes having no synchronized actions in common with processes $Q_2$ and $Q_1$, respectively. If protocol models $P_1|[G]|Q_1$ and $P_2|[G]|Q_2$ are deadlock free, then $PM = (P_1 \,|||\, P_2)|[G]|(Q_1 \,|||\, Q_2)$ is deadlock free.* □

**Lemma 3.3** *Let $P_1$ and $P_2$ be processes having no synchronized actions in common and let $a \in H$ and $b \in H'$. If $(P_1 \,|||\, P_2)|[G]|(Q_1 \,|||\, Q_2)$ is deadlock free, then $PM = (a; P_1|[H]|a; P_2)|[G]|(b; Q_1|[H']|b; Q_2)$ is deadlock free.* □

**Theorem 3.1 (syntactic property)** *Let $Q = peer(P)$ be the peer entity of a given entity $P$ obtained by the synthesis algorithm. Then $PM = P|[G]|Q$ is deadlock free.* □

**Theorem 3.2 (Semantic Property)** *Let $Q = peer(P)$ be the peer entity of a given entity $P$ obtained by the synthesis algorithm. Then the following semantic properties are satisfied, where $\tilde{\beta}$ denotes the sequence $\beta_1\uparrow, \ldots, \beta_n\uparrow, \beta_1\downarrow, \ldots, \beta_m\downarrow$ and $\tilde{\gamma}$ denotes the sequence $\gamma_1\uparrow, \ldots, \gamma_n\uparrow, \gamma_1\downarrow, \ldots, \gamma_m\downarrow$ of non-synchronized actions appearing in $P$ and $Q$ respectively.*

(1) $\hat{P} \approx$ **hide** $\tilde{\beta}$ **in** $P|[G]|Q$, *where $\hat{P}$ is the resulting $P$ with an action of the form $\overleftarrow{\alpha}$ replaced by $\mathbf{i}; \overleftarrow{\alpha}$, i.e. $\hat{P} = P\{\mathbf{i}; \overleftarrow{\alpha} / \overleftarrow{\alpha}\}$*

(2) $\hat{Q} \approx$ **hide** $\tilde{\gamma}$ **in** $P|[G]|Q$, *where $\hat{Q}$ is the resulting $Q$ with an action of the form $\overrightarrow{\alpha}$ replaced by $\mathbf{i}; \overrightarrow{\alpha}$, i.e. $\hat{Q} = Q\{\mathbf{i}; \overrightarrow{\alpha} / \overrightarrow{\alpha}\}$.* □

Note that for enabling and disabling operators we have,
$\hat{P} \approx$ **hide** $\tilde{\beta}$ **in** $(P_1 \gg nc; P_2)|[G \cup \{nc\}]|(Q_1 \gg nc; Q_2)$ and
$\hat{P} \approx$ **hide** $\tilde{\beta}$ **in** $(P_1[> nc; P_2)|[G \cup \{nc\}]|(Q_1[> nc; Q_2)$. Similarly for $\hat{Q}$

### 3.3 Example of Application of Synthesis Algorithm

We briefly present an application example of the synthesis algorithm to the small portion of simplified version of the protocol handler of the Transport Layer [9]. The details can be found in [4],[5], [9]. The protocol handler has two entities, a *calling* entity and a *called* entity. Entities exchange messages in three phases, *connection phase*, *data phase* and *disconnect phase*. Suppose the given entity, $P$, is a calling entity which defined as:
$P$ = Connect_Phase_$P \gg$ (Data_Phase_$P$ [> Disconnect_Phase_$P$) . The given entity is transformed as shown below.
$P$ = Connect_Phase_$P \gg nc$; (Data_Phase_$P$ [> $nc$;Disconnect_Phase_$P$).

The peer entity $Q$ is obtained by applying the synthesis algorithm as follows:
$Q = peer(\text{Connect\_Phase\_}P) \gg nc;$
$(peer(\text{Data\_Phase\_}P)\,[> nc; peer(\text{Disconnect\_Phase\_}P))$

We use the following abbreviations:

For SPs, $ConReq\downarrow : \gamma_1\downarrow, ConCnf\uparrow : \gamma_2\uparrow, DisIn\uparrow : \gamma_3\uparrow, DatReq\downarrow : \gamma_4\downarrow, DatIn\uparrow : \gamma_5\uparrow, DisReq\downarrow : \gamma_6\downarrow.$

For PDUs, $\overrightarrow{CR} : \overrightarrow{\alpha_1}, \overleftarrow{CC} : \overleftarrow{\alpha_2}, \overleftarrow{DR} : \overleftarrow{\alpha_3}, \overrightarrow{DT} : \overrightarrow{\alpha_4}, \overleftarrow{DT} : \overleftarrow{\alpha_5}, \overrightarrow{DR} : \overrightarrow{\alpha_6}$.

The phases of the given entity are as follows

Connect_Phase_$P$ =
$\gamma_1\downarrow;\overrightarrow{\alpha_1};(\overleftarrow{\alpha_2};\gamma_2\uparrow;\mathbf{exit}[]\overleftarrow{\alpha_3};\gamma_3\uparrow;$Connect_Phase_$P$)
Data_Phase_$P$ =
$\gamma_4\downarrow;\overrightarrow{\alpha_4};$Data_Phase_$P$ $|||$ $\overleftarrow{\alpha_5};\gamma_5\uparrow;$Data_Phase_$P$
Disconnect_Phase_$P$ =
$\gamma_6\downarrow;\overrightarrow{\alpha_6};\mathbf{exit}$ $|||$ $\overleftarrow{\alpha_3};\gamma_3\uparrow;\mathbf{exit}$

Here, we apply the algorithm to the Connect_Phase_$P$ of the given entity only. The application of the algorithm is shown as follows;

Connect_Phase_$Q$ = $peer$(Connect_Phase_$P$)
from (5) =
$(\mathbf{rec}\,x.peer(\gamma_1\downarrow;\overrightarrow{\alpha_1};(\overleftarrow{\alpha_2};\gamma_2\uparrow;\mathbf{exit}[]\overleftarrow{\alpha_3};\gamma_3\uparrow;x)))$
from (S1) =
$\mathbf{rec}\,x.\overrightarrow{\alpha_1};\beta_1\uparrow;peer((\overleftarrow{\alpha_2};\gamma_2\uparrow;\mathbf{exit}[]\overleftarrow{\alpha_3};\gamma_3\uparrow;x))$
from (R1) =
$\mathbf{rec}\,x.\overrightarrow{\alpha_1};\beta_1\uparrow;(\beta_2\downarrow;\overleftarrow{\alpha_2};peer(\mathbf{exit})[]$
$\beta_3\downarrow;\overleftarrow{\alpha_3};peer(x))$
from (2) =
$\mathbf{rec}\,x.\overrightarrow{\alpha_1};\beta_1\uparrow;(\beta_2\downarrow;\overleftarrow{\alpha_2};\mathbf{exit}[]\beta_3\downarrow;\overleftarrow{\alpha_3};peer(x))$
from (3) =
$\mathbf{rec}\,x.\overrightarrow{\alpha_1};\beta_1\uparrow;(\beta_2\downarrow;\overleftarrow{\alpha_2};\mathbf{exit}[]\beta_3\downarrow;\overleftarrow{\alpha_3};x)$

The result is:
Connect_Phase_$Q$ = $\overrightarrow{\alpha_1};\beta_1\uparrow;(\beta_2\downarrow;\overleftarrow{\alpha_2};\mathbf{exit}[]$
$\beta_3\downarrow;\overleftarrow{\alpha_3};$ Connect_Phase_$Q$)

## 4 Conclusions

We proposed a protocol synthesis algorithm for the process-algebraic language LOTOS to derive a structured protocol model by generating a peer entity from a given entity. Synthesized protocol is proved to be deadlock free and is bisimulated by both entities if the given entity is expressed as a communicating process, which is natural for communicating entities. We also proposed the definition of *deadlock* which is different from the deadlock defined in the protocol synthesis based on FSMs. The protocol synthesis procedure we have presented is applicable to every structure of protocol layer which provides services to its upper layer.

## References


[1] ISO. Estelle: A formal Description Technique based on an Extended State Transition Model. ISO 9074, 1989.

[2] ISO. Information Processing Systems — Open Systems Interconnection — LOTOS — A formal description technique based on the temporal ordering of observational behaviour. IS 8807, 1989.

[3] CCITT. SDL: Specification and Description Language. CCITT Z.100, 1988.

[4] B.B. Bista, Zixue Cheng, Atsushi Togashi, and Norio Shiratori. A new approach for protocol synthesis based on LOTOS. *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, E77-A(10):1646–1655, Oct 1994.

[5] B.B. Bista, Zixue Cheng, Atsushi Togashi, and Norio Shiratori. A synthesis algorithm of a protocol model from a single entity. In Dieter Hogrefe and Stefan Leue, editors, *Seventh Int. Conference on Formal Description Techniques, FORTE'94*, pages 467–482, Berne, Switzerland, 4-7 October 1994.

[6] C. A. Vissers, G. Scollo, M. van Sinderen, and E. Brinksma. Specification styles in distributed systems design and verification. *Theoretical Computer Science*, 89:179–206, 1991.

[7] Yao-Xue Zhang, Kaoru Takahashi, Norio Shiratori, and Shoichi Noguchi. An interactive protocol synthesis algorithm using a global state transition graph. *IEEE Transactions on Software Engineering*, 14(3), March 1988.

[8] B.B. Bista, Zixue Cheng, and Norio Shiratori. A LOTOS based synthesis method for protocol specification. Technical Report 2, Multimedia Communication and Distributed Processing Workshop, IPS of Japan, Nov 1993.

[9] A. Tanenbaum. *Computer Networks.* Prentice-Hall International Editions, 1989.