A LOTOS Based Synthesis Method for Protocol Specification

Bhed Bahadur BISTA, Norio SHIRATORI
Tohoku University
Zixue CHENG
Aizu University

Abstract

This paper purposes a protocol synthesis method for protocol entities whose behaviours are described in LOTOS. In this method, a single entity described in LOTOS is given. The peer entity is generated automatically from the given entity if the given entity posses certain desirable properties. The two entities interact via synchronous communication. One of the main problems in synchronous communication is *deadlock* which arises when synchronously communicating entities are unable to synchronize for communication and wait for each other for synchronization. The synthesis method presented in this paper ensures that the interactions between the given entity and the generated peer entity are deadlock free.

1 Introduction

Protocol synthesis is a process of designing new communication protocols. Since the development of computer communications and networkings, the role of communication protocols in maintaining smooth operation of computer communications and networkings has been recognized.

The objective of developing automatic protocol synthesis method is to provide systematic way of designing protocols such that their correctness can be ensured. Broadly speaking there are two types of protocol synthesis methods[8]. They are *service oriented* and *non-service oriented*. In the service oriented protocol synthesis method, a protocol specification is derived from the service specification. However, the assumption is that the given service specification provides the properties required for the correctness. The service oriented method is based on Finite State Machine(FSM)[1][2] and LOTOS[3]. In non-service oriented protocol synthesis method, incomplete protocols or protocols with errors are given and the synthesis methods generate the correct protocols. It is based on FSM[4]. The brief comparison of them is given in figure 1. Our synthesis method is non-service oriented based on LOTOS. One of the main features and advantages of LOTOS over FSM is the ability of synchronization for communication among entities. One of the main problems in synchronous communication is *deadlock*, which is a situation when synchronously communicating entities are unable to synchronize and enter in a state where no further synchronization is possible. Our synthesis method automatically generates a peer entity from a given single entity by de-

	Service Oriented	Non-service Oriented
Starting points	service specification	incomplete protocol or partial protocol
Assumptions	no logical errors in service specification	logical errors in protocol specification
Aim	derive protocol spec. such that safety properties are satisfied	complete protocol spec. such that safety properties are satisfied
Based on	FSM [1],[2] LOTOS [3]	FSM [4] LOTOS [This paper]

Figure 1: Comparison of Service oriented and Non-service oriented Synthesis Method 4

composing the given entity into *components*. Using the components and predefined rules the components of the peer entity are generated which are then combined in order to obtain the complete specification of the peer entity. The protocol generated is guaranteed to be well behaved and deadlock free if the given entity model posses a certain desirable properties.

2 LOTOS

LOTOS (Language Of Temporal Ordering Specification) is an FDT(Formal Description Technique) developed by ISO (International Organization for Standardization) [5] for the formal description of distributed systems. A distributed system specified in LOTOS consists a number of *processes* interacting with each other. A process in LOTOS is considered as an abstract entity which is capable of performing *internal events* and communicates with other processes via *communication events* by synchronizing at points called *gates*. An event or an *action* is considered as an atomic (i.e. not divisible in time) event. A behaviour of a process is defined by stating the temporal relation of events of the process. There are various LOTOS operators (eg. *prefix(;), choice([])*) to form behaviour expressions and combine behaviour expressions to yield complex behaviour expressions. LOTOS has two parts, process part and data part. In this paper we are concerned with process part whose semantic is modeled by *Labeled Transition System*(LTS).

A LTS has a 4-tuples $< S, Act, T, s_0 >$ where S is a finite state set, Act is a set of actions, T is a set of transition relation such that $T \subseteq S \times S$ and $s_0 \in S$ is the initial state.

3 Component Based Synthesis Methodology

In this section, we will describe our protocol synthesis problem and our approach to solve the problem. Protocol entities (processes in LOTOS) involved in communication are described in LOTOS. In this paper we will use a protocol entity and a process synonymously.

3.1 Synthesis Problem

The number of entities involved in our synthesis method are two and interaction between them is via synchronous communication.

Two protocol entities communicating with each other synchronously and providing services to their users can be depicted as shown in figure 2.

Each entity has two sets of interaction

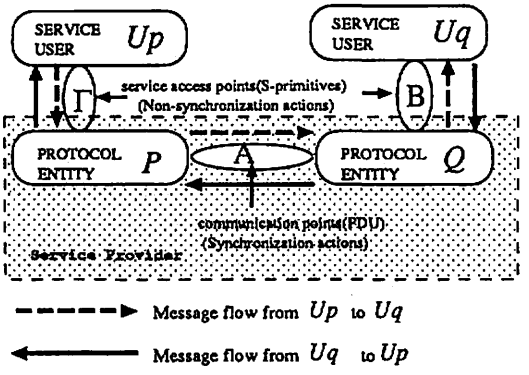


Figure 2: Synchronous Communication Model

points (gates in LOTOS):

(1) a set of interaction points at which an entity and its service user interact for communication (i.e. Γ and B).

(2) a set of interaction points at which two entities interact for communication (i.e. A).

We define actions occurring at (1) and (2) as *non-synchronization actions* and *synchronization actions* of a entity respectively as defined in definitions 2 and 3.

[Definition 1] (Actions of process P):
Actions of process P, $Act(P)$, is defined as a set of all possible actions executed by P. □

[Definition 2] (Synchronization actions of process P):
Synchronization actions of process P, $Sync(P)$, is defined as a set of actions at which P synchronizes with a process (peer entity) Q. □

Example: If $P|[a,c]|Q$ then $Sync(P) = Sync(Q) = \{a,c\}$ □

[Definition 3] (Non-synchronization actions of process P):
Non-synchronization actions of process P, $Nonsync(P)$ is defined as $Nonsync(P) = Act(P) - Sync(P)$ (i.e. $Act(P) = Nonsync(P) \cup Sync(P)$). □

Observable actions for a user are non-synchronization actions. Synchronization actions are hidden from the user. But we are concerned with the specification of a protocol entity. From the protocol entity's point of view both non-synchronization and synchronization actions are observable actions. The communication model in figure 2 is closely related to OSI layer systems. So we will consider actions occurring at points (1) and (2) as service primitives and protocol data units(PDUs) respectively.

[Definition 4] (Definition of System Deadlock)
Let a system, $S \stackrel{def}{=} P|[sync\ actions]|Q$, where P and Q are processes then S is said to be deadlock whenever;
$\forall \alpha \in Act(S), \exists S', \exists t \in (\mathrm{Act(S)} - \{\delta\})^*$ such that $S \stackrel{t}{\Longrightarrow} S' \stackrel{\alpha}{\not\rightarrow}$. □
Here
(1) $S' \stackrel{\alpha}{\not\rightarrow}$ means that $\neg\exists S''$ such that $S' \stackrel{\alpha}{\rightarrow} S''$.
(2) $S \stackrel{t}{\Longrightarrow} S'$ means that $\exists S_i (1 \leq i \leq n), t_j \in t$, such that $S = S_0 \stackrel{t_1}{\rightarrow} S_1 \stackrel{t_2}{\rightarrow} S_2 \stackrel{t_3}{\rightarrow} .... \stackrel{t_j}{\rightarrow} S'$.

The deadlock defined in definition 4 is different from the definition of deadlock defined in FSM based protocol synthesis methods. In definition 4, system S after executing sequences of actions t enters a intermediate state S' from which it cannot execute any actions.

[Example](System Deadlock)
Let us assume that the system consist of two processes P and Q and synchronization actions are a and b as shown in figure 3. It is easy to note that P and Q cannot synchronize for communication and enter into deadlock state.

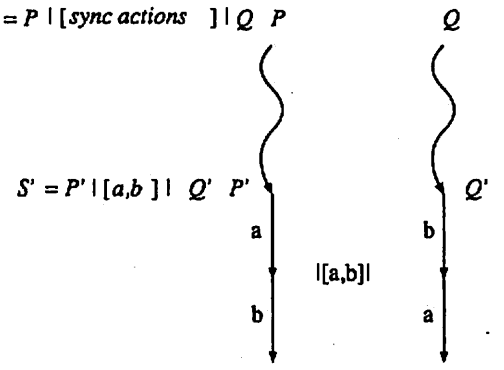


Figure 3: Example of Deadlock

Now we are ready to define our synthesis problem.

[Definition 5](Protocol Synthesis Problem)
Given a single entity P, whose specification is described in Basic LOTOS, derive a specification of the peer entity Q, which is synchronously communicating with P, in Basic LOTOS such that their interactions are deadlock-free and complete. □

Deadlock-freeness guarantees that no communicating entities are waiting for each other forever. Completeness ensures that each send message in an entity appears as receive message in its peer entity.

[Notations]

We use the following notations in the rest of the discussion.

Γ_P: a set of non-synchronization actions of P.
B_Q : a set of non-synchronization actions of Q.
A_{PQ} : a set of synchronization actions(of P and Q).
Let
$\gamma_a\downarrow, \gamma_b\downarrow, .. \in \Gamma_P$ be service primitives(interactions points) from the user to P representing send actions.
$\gamma_1\uparrow, \gamma_2\uparrow, ... \in \Gamma_P$ be service primitives from P to the user representing receive actions.
$\beta_a\uparrow, \beta_b\uparrow, ... \in B_Q$ be service primitives from Q to the user representing receive actions.
$\beta_1\downarrow, \beta_2\downarrow, ... \in B_Q$ be service primitives from the user to Q representing send actions.
$\vec{\alpha}_a, \vec{\alpha}_b, ... \in A_{PQ}$ be Protocol Data Units(PDUs) from P to Q.
$\overleftarrow{\alpha}_1, \overleftarrow{\alpha}_2, ... \in A_{QP}$ be Protocol Data Units(PDUs) from Q to P.

3.2 Outline of Synthesis Method

The outline of our synthesis method can be summarized by figure 4. The specification of the given entity P, which is the input to our algorithm, is described in basic LOTOS and satisfies the following four assumptions.

[Assumption 1] Whenever P synchronizes with its user to receive a message in order to send to Q then P must immediately synchronize with Q to send the message,(i.e.if $P \xrightarrow{\gamma\downarrow} P'$ then $P' \xrightarrow{\vec{\alpha}}$).

[Assumption 2] Whenever P synchronizes with Q to receive a message from Q then P must immediately synchronize with its usr to deliver the message,(i.e. if

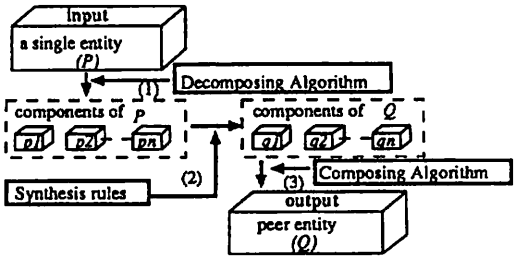


Figure 4: Protocol Synthesis Method

$P \xrightarrow{\bar{\alpha}} P'$then $P' \xrightarrow{\gamma\downarrow}$).

[**Assumption 3**] There are no internal actions i in the specification of P.

[**Assumption 4**] P does not contain any choice between $\gamma\downarrow$ and $\bar{\alpha}$.

It is easy to note that the given assumptions are not special assumptions but very natural in protocol design systems. If the specification of P satisfies these assumptions then it is guaranteed that the communication between P and its peer entity (generated by our synthesis algorithm) progresses without deadlock.

Our synthesis method can be described informally in three steps as shown below.

step 1: Using the decomposing algorithm, the given specification is decomposed into components $p_1, p_2, ..., p_n$ in such way that each components has a pair/s of synchronization and non-synchronization actions.

step 2: Using the pre-defined rules and the components $p_1, p_2, ..., p_n$, the components $q_1, q_2, ..., q_n$ of the peer entity Q are generated.

step 3: Using the composing algorithm which uses the information relating components of P, the components of Q are combined in order to obtain the complete specification of Q.
The peer entity Q is unique to P.

4 Synthesis Method

In this section, we will describe decomposing algorithm, synthesis rules, composing algorithm and finally an application example.

4.1 Decomposition of Given Entity

The specification of a given entity P is decomposed at certain states of P into components $p_1, p_2, p_3, ...$ such a way that each component has $n(n = \{1, 2\})$ pairs of synchronization and non-synchronization actions. If a component has only prefix operator then $n = 1$ and if it has choice operator then $n = 2$. In the initial decomposition of P, n can be greater that 2 for a component. In such cases the component is further decomposed.

It is important to maintain the inter-component information (i.e. information of states of components) for future use such as in composing algorithm to combine the components of Q and to know how the specification of P was decomposed. So we define the *state relation* between components of P as in definition 6.

[**Definition 6**](State Relation)
Let $p_1, p_2, ..., p_n$ be the components of P obtained sequentially as a result of applying decomposing algorithm on P and let S_1 be the set of states of p_1, S_2 be the set of states of p_2 and so on, then the *state relation* $\mathbf{R}$, is defined as $\mathbf{R}_{12} \subset S_1 \times S_2$.
$\mathbf{R} = \{\mathbf{R}_{ij}\}$ where $\mathbf{R}_{ij} \subset S_i \times S_j$.

Algorithm for Decomposing Given Entity

We assume that the given entity is in the form as shown bellow.
$P = \sum\{a_i; b_i; P_i \mid i \in I\}$ for finite index set I, where P_i is either process identifier or is in the form of P again.
Please note that if $a_i \in \gamma\downarrow$ then $b_i \in \vec{\alpha}$ and if $a_i \in \overleftarrow{\alpha}$ then $b_i \in \gamma\uparrow$ (from assumptions 1 and 2).

(* *Repeat* *)

[step 1]
(Decomposition of P and Generation of State Relation R*)*

(1) $Decomp(P) = \sum\{p_i; Decomp(P_i) | i \in I\}$
(2)$\mathbf{R}$=$\{(finalstate(p_i)) = (initialstate(P_i))\}$

where $p_i \equiv a_i; b_i$

[step 2]
(Generation of components)

If $I = \{1\}$ then $p_1 = a_1; b_1$

[step 3]
(Generation of components)

If $I = \{1, 2, 3, ..., n\}$ and n is *even* then
$p_1 = p_{11}[]p_{12} = a_1; b_1[]a_2; b_2$
$p_2 = p_{21}[]p_{22} = a_3; b_3[]a_4; b_4$
$\vdots$
$p_{n/2} = p_{(n/2)1}[]p_{(n/2)2} = a_{n-1}; b_{n-1}[]a_n; b_n$
$\mathbf{R}_{12} = \{(initialstate(p_1)) = (initialstate(p_2))\}$
$\vdots$
$\mathbf{R}_{((n-2)/2)(n/2)} = \{(initialstate(p_{(n-2)/2})) = (initialstate(p_{n/2}))\}$

[step 4]
(Generation of components)

If $I = \{1, 2, 3, ..., n\}$ and n is *odd* then
$p_1 = p_{11}[]p_{12} = a_1; b_1[]a_2; b_2$
$p_2 = p_{21}[]p_{22} = a_3; b_3[]a_4; b_4$
$\vdots$
$p_{(n+1)/2} = a_n; b_n$
$\mathbf{R}_{12} = \{(initialstate(p_1)) = (initialstate(p_2))\}$
$\mathbf{R}_{((n-1)/2(n+1)/2)} = \{(initialstate(p_{(n-1)/2})) = (initialstate(p_{(n+1)/2}))\}$

(* *Until P cannot be decomposed* *)

4.2 Synthesis Rules

These synthesis rules are applied on the components of given entity P to generate the components of peer entity Q. Prefix rules are applied if the given component of P is in prefix form and choice rules are applied if it is in choice form. Here $\dot{p}(h)$ denotes the component p and the state h of P and $q(k)$ denotes the component q and the state k of Q and other notations are same as in section 3.1 .

Prefix Rules

Rule_1 *If* $p(h) \xrightarrow{\vec{\alpha}_1} p(h+1) \xrightarrow{\gamma_1\uparrow} p(h+2)$
then $q(k) \xrightarrow{\beta_1\downarrow} q(k+1) \xrightarrow{\overleftarrow{\alpha}_1} q(k+2)$

Rule_2 *If* $p(h) \xrightarrow{\gamma_a\downarrow} p(h+1) \xrightarrow{\vec{\alpha}_2} p(h+2)$
then $q(k) \xrightarrow{\overleftarrow{\alpha}_2} q(k+1) \xrightarrow{\beta_a\uparrow} q(k+2)$

Choice Rules

Rule_3 *If* $p(h) \xrightarrow{\vec{\alpha}_1} p(h+1) \xrightarrow{\gamma_1\uparrow} p(h+2)$
[] $p(h) \xrightarrow{\vec{\alpha}_2} p(h+3) \xrightarrow{\gamma_2\uparrow} p(h+4)$
then $q(k) \xrightarrow{\beta_1\downarrow} q(k+1) \xrightarrow{\overleftarrow{\alpha}_1} q(k+2)$

$$
\begin{array}{lll}
 & [] & q(k) \overset{\beta_2\downarrow}{\rightarrow} q(k+3) \overset{\leftarrow\alpha_2}{\rightarrow} q(k+4) \\
\textit{Rule\_4} \quad \textit{If} & & p(h) \overset{\gamma_a\downarrow}{\rightarrow} p(h+1) \overset{\vec{\alpha}_a}{\rightarrow} p(h+2) \\
 & [] & p(h) \overset{\gamma_b\downarrow}{\rightarrow} p(h+3) \overset{\vec{\alpha}_b}{\rightarrow} p(h+4) \\
\textit{then} & & q(k) \overset{\vec{\alpha}_a}{\rightarrow} q(k+1) \overset{\beta_a\uparrow}{\rightarrow} q(k+2) \\
 & [] & q(k) \overset{\vec{\alpha}_b}{\rightarrow} q(k+3) \overset{\beta_b\uparrow}{\rightarrow} q(k+4)
\end{array}
$$

4.3 Composition of Peer Entity

Two components of Q are combined at a time starting from the first component i.e. q_1 and q_2, q_2 and q_3, and so on, until no components are left for combinations. In case of recursion in a given entity a higher number component is combined with a lower number component. In order to combine the components of Q we need to know the intercomponent information of P; i.e. state relation **R** otherwise correct combinations of components is not possible. Besides, it is used for combining two components by LOTOS operators (prefix or choice). How **R** is used to combine components by LOTOS operators will be clear in the following algorithm.

Algorithm for Composing Q

The composing function takes two arguments.
Suppose q_1 is obtained by applying synthesis rules on p_1 and q_2 is obtained by applying synthesis rules on p_2 then

(* *Repeat* *)

[step 1]
(**R** *is converted into prefix operator*)

If $\mathbf{R}_{12} =$
$\{(finalstate(p_1)) = (initialstate(p_2))\}$
then
$Comp(q_1, q_2) = q_1; q_2$

[step 2]
(**R** *is converted into choice operator*)

If $\mathbf{R}_{12} =$
$\{(initialstate(p_1)) = (initialstate(p_2))\}$
then
$Comp(q_1, q_2) = q_1 [] q_2$

. (* *Until there are no components of* Q *to be combined* *)

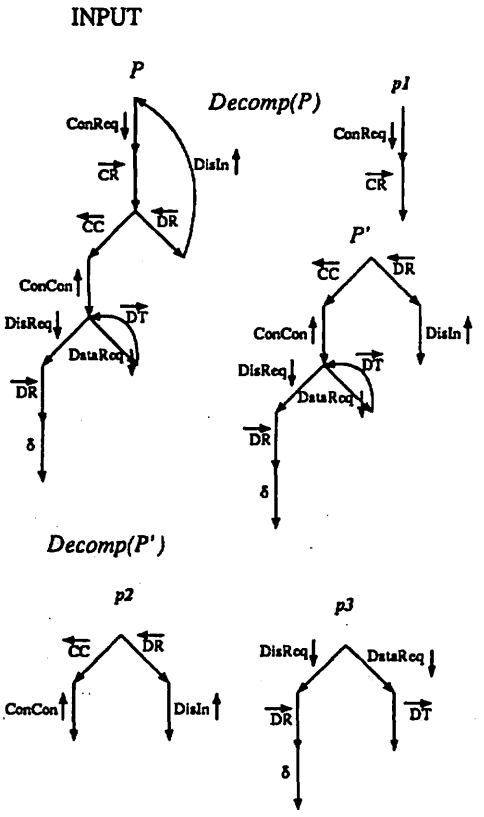
4.4 An Application Example

The example shown in figure 5 and figure 6 is a three phases (i.e. connection establishment, data transfer and disconnect phases) protocol. For simplicity, the specification of the given entity P, is shown in the LTS form of LOTOS specification and the generated peer entity is also in LTS form.

4.5 Theorems and Proof of Theorem

[Theorem]
System S consisting of processes P and Q defined as,
$S \overset{def}{=} P[[synchronization\ actions]]Q$ is deadlock free.

[Proof]
The proof directly follows from *Decomposing* algorithm, *Synthesis Rules* and *Composing* algorithm. The decomposing algorithm decomposes the specification of P into finite number of components $p_1, p_2, ..$ such that the synthesis rules



State Relations of Components:

R_{12} ={(final state(*p1*)) = (initial state(*p2*))}

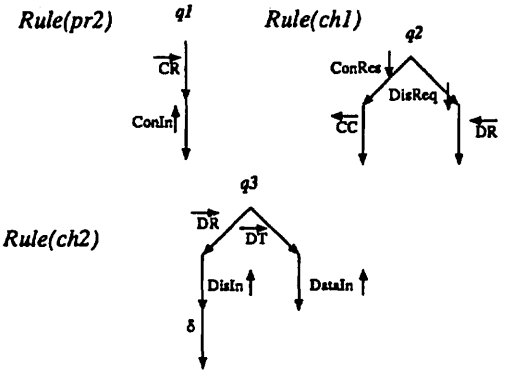
R_{21} ={(final state(*p22*))= (initial state*p1*))}

R_{23} ={(final state(*p21*)) = (initial state*p3*))}

R_{33} ={(final state(*p32*)) = (initial state*p3*))}

Figure 5: Decomposition of Given Entity P

Applying Synthesis Rules



Applying Composing Algorithm

R_{12} ={(final state(*p1*)) = (initial state(*p2*))} therefore

Comp(q1 ,q2) =q1 ; q2

R_{21} ={(final state(*p22*))= (initial state*p1*))} therefore

Comp(q22, q1) =q22 ; q1

R_{23} ={(final state(*p21*)) = (initial state*p3*))} therefore

Comp(q21,q3) = q21 ; q3

R_{33} ={(final state(*p32*)) = (initial state*p3*))} therefore

Comp(q32,q3) =q32 ; q3

OUTPUT

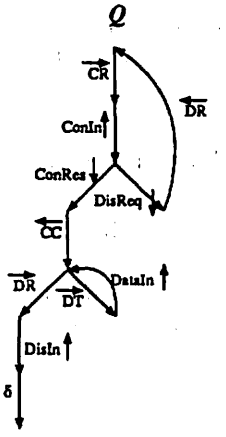


Figure 6: Generation of Peer Entity Q

can be directly applied on them to generate components q_1, q_2.. of Q. The synthesis rules generate one and only one component of Q from a given component of P (i.e. one to one component mapping) and p_i ||[*synchronization actions*]|| q_i is always deadlock free. The composing algorithm combines the components of Q to generate the complete specification of Q using the knowledge of **R** such that interaction among the specifications of P and Q occurs components by components (i.e. p_1 interacts with q_1, p_2 interacts with q_2, ...) as shown bellow;

p_1 ||[*sync actions*]|| q_1

||

Traces of p_1 ||[*sync actions*]|| q_1

⇓

p_2 ||[*sync actions*]|| q_2

||

Traces of p_2 ||[*sync actions*]|| q_2

⇓

⋮

||

Traces of p_{n-1} ||[*sync actions*]|| q_{n-1}

⇓

p_n ||[*sync actions*]|| q_n

||

Traces of p_n ||[*sync actions*]|| q_n

⇓

⋮

Since the interactions of P and Q proceed from corresponding components to components and the interactions between components to components is deadlock free, the system is deadlock free.

5 Conclusion

In this paper we have developed a systematic way of generating a peer entity from the specification of a given entity in basic LOTOS for the synchronous communication model. One main logical error in synchronous communication is deadlock. Our synthesis method guarantees that the communicating entities progress without deadlock. Our future work is to eliminate assumptions on given entity.

References

[1] Kassem Saleh, Robert Probert, *"A Service-Based Method For The Synthesis of Communication Protocols"*, International Journal of Mini and Microcomputers. Vol. 12, No. 3, 1990.

[2] B.V. Bochmann and R. Gotzhein, *"Deriving Protocol Specification From Service Specification"*, Proc. SIGCOMM'86, 1986, pp 144-156.

[3] R. Langerak, *"Decomposition of Functionality: A Correctness-preserving LOTOS Transformation"*, in Proc. Tenth IFIP Int. Symp. Protocol Specification, Testing and Verification, June 1990.

[4] Y. Kakuda and Y. Wakahara, *"Component-Based Synthesis of Protocols for unlimited number of processes"*, in Proc. COMPSAC'87, 1988.

[5] ISO, *"LOTOS- a formal description technique based on the temporal ordering of observational behaviour-"*, ISO 8807(Feb. 1989)

[6] T. Bolognesi and E. Brinksma, *"Introduction to the ISO Specification Language LOTOS"*, Computer Networks and ISDN Systems,14(1987), pp. 25-59.

[7] C. V. Ramamoorthy, S. T. Dong and Y. Usuda,*An Implementation of an Automated Protocol Synthesizer(APS) and Its Application to the X.21 Protocol"*, IEEE Transactions on Software Engineering VOL. SE-11, NO 9, Sept. 1985.

[8] Robert L. Probert and Kassem Saleh, *" Synthesis of Communication Protocols: Survey and Assessment"*, IEEE Transaction on Computers, vol. 40, April 1991.

[9] Gregor v. Bochmann, *Protocol Specification for OSI"*, Computer Networks and ISDN Systems 18(1989/90) pp. 167-184.