Stepwise Telecommunication Software Generation from Service Specifications in State Transition Model

Akira Takura Tadashi Ohta

ATR Communication Systems Research Laboratories
2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

Abstract

A method of stepwise telecommunication software generation through terminal behavior observation is proposed to implement telecommunication services on any architecture. In the proposed method, a two-layered language is used; service specification language STR and supplementary specification language STR/D. STR specifies terminal behaviors which can be recognized from the outside of a telecommunication system. Supplementary specifications are described with STR/D to implement STR rules.

Telecommunication services, such as UPT (Universal Personal Telecommunication), are standardized to be provided on a given functional model consisting of functional entities. Software for each functional entity is semiautomatically obtained from the above two kinds of initially described specifications.

1 Introduction

There are some specification languages, such as SDL [1], LOTOS [2, 3], to describe communications systems or protocols. Using these specification languages to develop a telecommunication software, it is necessary to describe the protocol specification which describes the interactions between protocol entities.

There are some refinement methods of specifications [4, 5, 6]. Cameron et al. [5] uses a rule-based language L.0 to implement a real-life protocol by an incremental development and refinement. Tsai et al. [6] uses a frame-and-rule oriented requirement specification language FRORL. In these methods, specifications are incrementally refined to obtain protocol specifications, however, they can not synthesize protocol specifications from service specifications.

Bochmann and Gotzhein [7], Chu and Liu [8], Saleh and Probert [9] proposed protocol synthesis methods from service specifications. They assume the relationships between protocols and services described in Fig. 1. A communications system provides communications services for service users who access the system through service access points SAP1, $\cdots$, SAPn. Within the communications system protocol entities cooperate to provide the services by exchanging messages between entities. This communications between entities are provided by the service provider of lower layers. In these synthesis approaches, we can obtain protocol specification from service specifications. However, these methods cannot handle real-life protocols completely [10].

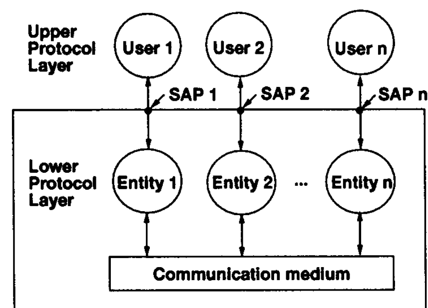

Figure 1: Architecture model for layered protocol

Telecommunication service specifications can be described by specifying terminal behaviors which can be recognized from the outside of telecommunication systems [11]. With this method, we can describe specifications without detailed knowledge of the target telecommunication system. The specification language used in this method is called STR (State Transition Rules). With STR, all specifications consist of a set of rules. Each STR rule specifies a global state transition of terminals for an input event that

has occurred at a terminal. In this service specification description, terminals are considered service access points in a communications system.

The specification described with STR cannot become an implementable specification capable of controlling a terminal and network. To fill this gap, we need a supplementary specification. Such a supplementary specification is described with the language STR/D [12] by experts who have knowledge of the communications system.

Specifications described with both STR and STR/D are automatically transformed into an FSM-based (finite state machine) protocol specification able to control a terminal and network [13]. In this transformation, we assume that the terminal is controlled by a process, which communicates with other processes to decide the STR rule to be applied. This architecture model is the same as that of Fig. 1. Note, however, that some standardized communications services are incompatible with this protocol architecture model. UPT, for instance, uses a functional model [14].

This paper proposes a specification description method that produces functional entity specifications for any functional model. In this method, service specifications are described with STR. The information flow among functional entities is standardized. Therefore, the specifications of the information flow can be described as knowledge with STR/D. Therefore, a service designer who describes service specifications with STR rules, does not have to know the detailed functional model. We show an application example of the proposed method to UPT.

2 Two-layered specification description

A two-layered language is used; one layer includes specification description language STR and the other layer includes supplementary specification description language STR/D. The first layer specifies terminal behaviors observable from the outside of communications systems. The second layer describes supplementary specifications to control terminals and a network.

2.1 STR

A service is defined as a set of STR rules. An STR rule has the form:

initial-global-state event: next-global-state.

The "initial-global-state" and the "next-global-state" represent global states of terminals. A global state is represented by a set of local states. A local state is represented by a set of state primitives. A state primitive has one or two arguments as variables expressing terminals. The first argument represents the terminal having the state primitive expressing a terminal state. If the second argument is specified, the terminal designated by the first argument holds a relation of the primitive to the terminal specified by the second argument. Therefore, the local state of a terminal is defined as the set of state primitives whose first argument designates the terminal. A state primitive represents a terminal state which is recognizable from the outside of a communications system.

The "event" also has one or two arguments as variables expressing terminals. It represents a logical input to the terminal designated by its first argument. If the second argument is described in an event, the second argument represents a terminal identifier given by the event.

A rule may be applied to a set of terminals $t_1, \cdots, t_n$ if its event has occurred at one of the terminals, and these terminals have the primitives specified by the initial-global-state of the rule. If there are two rules, `r1`, `r2`, whose state primitives in the initial-global-state are included in the local states of terminals $t_1, \cdots, t_n$, and the initial-global-state of `r1` is included in `r2`, then `r2` is applied. This inclusion relation is not total-order. Thus, there still exists the possibility that multiple rules may be applied. When multiple rules can be applied, we may select a rule arbitrarily.

2.2 An example

Figure 2 shows an example of an STR rule. The "ring-back(A, B)" is a state primitive for terminal A, and B is a terminal to which A has the relation "ring-back". The "dial(A, B)" is an event at terminal A. This rule shows that if a user on dial-tone receiving terminal A dials terminal B, then the states of terminals A and B are changed to ring-back tone receiving state (A) and ringing state (B), respectively.

dial-tone(A), idle(B)
dial(A,B):
ring-back(A,B), ringing(B,A).

Figure 2: An example STR rule

2.3 Graph representation

We use graph representation of an STR rule to generate software from service specifications described with STR. An STR rule can be represented by two graphs: an initial graph corresponding to the initial global state and an event description, and the next graph corresponding to the next global state description. Both an initial graph and a next graph are called rule graphs. Figure 3 shows the graph representation of the rule shown in Fig. 2.

A rule graph consists of a set of vertices and directed edges. Each vertex has its own name and some vertices have labels. A vertex is denoted by a circle. The name of a vertex is written in a circle and labels of the vertex are written near the circle. Each edge may also have labels that are written near the edge. In an initial graph, there is a label which shows an event. This label is written in italics.

A vertex that has a label or an edge incident from it is called "labeled." For each vertex in an initial graph, there must be a path from the vertex at which an event occurred.

Figure 3: Graph representation of an STR rule

A global state for all the processes in the communications system is called a "system state," and the graph denoting the system state is called a "system graph." An STR rule states that the subgraph in the system graph that is isomorphic to the initial graph of the rule should be replaced by the next graph of the rule.

2.4 STR/D

STR/D describes supplementary specifications to implement service specifications described with STR. An STR/D specification consists of a set of STR/D rules. An STR/D rule describes tasks to be executed on the state transition of terminals by STR rules. Each STR/D rule has the form:

position-designation {task-designation}

This rule specifies that "task-designation" is executed at positions where the condition "position-designation" is satisfied on a state transition of a terminal.

There are eight kinds of position designations as follows:

1. Initial position designation: A rule for the initial position designates initial tasks to begin services.

2. Terminal position designation: A rule for the terminal position designates final tasks before returning to the initial state.

3. Primitive designation: A rule for a primitive designation indicates that the tasks in its task-designation are executed just before the states having the designated primitives.

4. State designation: A rule for a state designation indicates that the tasks in its task-designation are executed just before the designated state.

5. Input designation: A rule for an input designation indicates that tasks are executed just after receiving the designated message.

6. Primitive-input designation: A rule for primitive-input designation indicates that tasks are executed just after receiving the designated message at a state having the designated primitives.

7. State-input designation: A rule for state-input designation indicates that tasks are executed just after receiving the designated message at the designated state.

8. Transition: A rule for transition designation indicates that tasks are executed on a transition satisfying the condition specified by the difference of primitives between two states.

Tasks described in "task-designation" conform to C statements. These tasks are separated by semicolons if multiple tasks are specified in a single "task-designation".

2.5 An example

Figure 4 shows an example of an STR/D description corresponding to Fig. 2. Rule `r1` indicates that tone_on(DLTONE) is executed just before entering a state having the primitive "dial-tone(A)". Rule `r2` indicates that tone_on(RBTONE) is executed when the event dial(A,B) occurs at a state that contains the primitive dial-tone(A). Rule `r3` indicates that tone_on(RGTONE) is executed on a transition where the primitive "idle(A)" is deleted and the primitive "ringing(A,B)" is added at the next state.

3 Application of STR

We show specifications of UPT (universal Personal Telecommunication) described with STR and STR/D.

```
primitive(dial-tone(A))
        {tone_on(DLTONE);}                    (r1)
state_input((dial-tone(A)) & (event dial(A,B)))
        {tone_on(RBTONE);}                    (r2)
transition(-(idle(A)) +(ringing(A,B)))
        {tone_on(RGTONE);}                    (r3)
```

Figure 4: Example STR/D rules.

3.1 Universal personal telecommunication

UPT (Universal Personal Telecommunication) permits access to telecommunication services with personal mobility. Each UPT user has a unique UPT number. When a UPT user initiates or receives a call, the access is verified by checking the UPT number and authentication code. If the authentication is verified, the user can proceed to procedure identification.

UPT services are implemented on the functional model shown in Fig. 5 [14]. In Fig. 5 the Functional Entities (FEs) have the following meanings:

FE1 Originating CCAF
FE2 Originating CCF; associated with SSF
FE3 Transit CCF
FE4 Terminating CCF
FE5 Terminating CCAF
FE6 SCF
FE7 SDF(l) (SDF in the local network)
FE8 SRF
FE9 SDF(h) (SDF in the home network)
where the terms are as follows:
SSF Service Switching Function
SRF Specialized Resource Function
CCF Call Control Function
CCAF Call Control Agent Function
SCF Service Control Function
SDF Service Data Function

This functional model cannot be modeled by the layered architecture described in Fig. 1; however, each functional entity can be modeled by the layered architecture. The functional model cannot be observed from outside a communications system.

The following is an outline of the actions required for a UPT user to access a UPT service and undertake identification and authentication:

1. Access code input by UPT user

2., Recognition of access code, suspension of call processing in CCF, connection of SRF (Establish Temporary Connection)

3. Prompt and response for user identification (input UPT number)

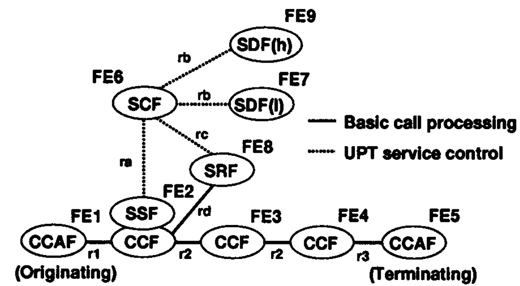

Figure 5: Functional model for UPT service set 1 provision

4. Prompt and response for user authentication (input authentication code)

5. UPT user's service provider undertakes authentication check and sends result

6. Decision:

- if successful, continue on to procedure identification
- if unsuccessful and more attempts allowed, advise user of failure and restart at 3
- if unsuccessful and no more attempts allowed, advise and release call.

Figure 6 shows the information flow for the procedure of "access, identification and authentication". There are two other information flows involved in the above actions: "authentication rejection and retry" and "maximum retries reached".

3.2 STR description of UPT

Figure 7 shows STR description of the information flows of access, identification and authentication; retry; and, maximum retries reached. This description introduces new variables to denote upt users. The variables declared by "Terminal" denote terminals as before; variables declared by "User" are used for upt users. In Fig. 7 the UPT user gets access through the terminal "A", "U" denotes the user's UPT number, and "V" denotes the other users' UPT numbers.

STR rules r1, r2, r3 express the information flow in Fig. 6. Rule r4 expresses the sequence of authentication retries performed because of a wrong authentication code. Rule r5 expresses the sequence when the retry limit is exceeded.

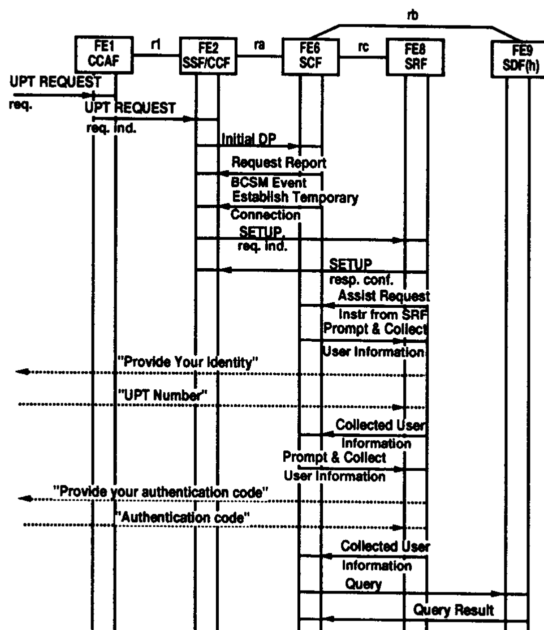

Figure 6: Access, identification and authentication

We give the meaning of each of the state primitives and events in Fig. 7. "dial-tone(A)" represents a state where a UPT service initiation request can be received. "ident(A)" represents a state where a UPT number can be received. "auth(A,U)" represents a state where the authentication code for UPT number "U" at terminal "A" can be received. "success(A)" represents the authentication success state. "fail(A)" represents the authentication failed state which results when the authentication retry limit is exceeded. "m_limit(A)" holds when the authentication retry through terminal "A" exceeds the limit.

Terminal A;
User U, V;
r1) dial-tone(A) uptreq(A): ident(A).
r2) ident(A) idnumber(A,U): auth(A,U).
r3) auth(A,U) acode(A,U): success(A,U).
r4) auth(A,U) acode(A,V): ident(A).
r5) auth(A,U), m_limit(A) acode(A,V): fail(A).

Figure 7: STR description of authentication in UPT.

"uptreq(A)" represents the UPT service initiation request. "idnumber(A,U)" represents an event for which a UPT identification number is received. "U" denotes the received UPT users's identification number. "acode(A,U)" represents a reception of the authentication code for user "U" at terminal "A".

3.3 STR/D description

Figure 8 shows the supplementary specification needed to implement the STR rules in Fig. 6 on the functional model in Fig. 5. In addition to the declaration of STR description, the functional entities defined in Fig. 5 are declared by "Entity". Tasks described in the "task-designation" part are provided as routines. The actions of these routines are standardized. Therefore these routines can be given as knowledge.

If there are two or more entities described as parameters of a task, then the task includes communications between the described entities. Tasks are carried out in the described order when more than one task is described in a single "task-designation". Therefore, the original order of communications is preserved.

4 Stepwise refinement

We show two refinement methods to derive softwares. The first method automatically generates software which assume the architecture in Fig. 1. The second method semiautomatically generates software which conforms to any functional model including the functional model for UPT in Fig. 5.

4.1 Architecture-independent refinement

We show a method of generating software from STR and STR/D rules. The STR compiler [13] generates an abstract process specification from STR rules. Protocol specifications are described in the generated abstract process specification. The STR/D compiler [12] generates an implementable process specification from a generated abstract process specification and STR/D specification. Finally, a process program written in C is obtained from the process specification by a code generator.

In this transformation we assume that the STR rules are restricted to those with a directed path going through all the vertices appearing in each initial-global-state of STR rules. This directed path is called a trunk. This restriction not only provides an efficient process specification but also permits description of various communications services [11].

```
Terminal A;
User U;
Entity CCAF, SSF/CCF, SCF, SRF;

transition(-(dial-tone(A)) +(ident(A)))
{ UptReq(CCAF, SSF/CCF);
  InitialDP(SSF/CCF, SCF);
  ReqReport(SCF, SSF/CCF);
  TempConnect(SCF, SSF/SCF);
  SetupReqInd(SSF/CCF, SRF);
  SetupRespConf(SRF, SCF);
  AssistReq(SRF, SCF);
  PromptCollect(SCF, SRF, "Provide your
      identity"); }
input(event number(A, U))
{ CollectedUserInf(SRF, SCF);
  PromptCollect(SCF, SRF, "Provide your
      authentication code"); }
transition(-(auth(A, U)) +(success(A)))
{ CollectedUserInf(SRF, SCF); }
transition(-(auth(A, U)) +(ident(A)))
{ PromptCollect("Wrong authentication,
      please retry.
      Provide your authentication code"); }
transition(-(auth(A, U)) +(fail(A)))
{ PromptCollect("Retry limit exceeded.
      Your line is now blocked.
      Please hang up."); }
```

Figure 8: STR/D rules for authentication.

The generation process consists of four steps.

Step 1: First, we classify STR rules for each event described in the rules. Next we hierarchically classify vertices in the trunks of initial graphs. The vertices with the same local state are represented as the same vertices in the classification tree. A vertex in the classification tree has a set of corresponding STR rules, although the leaf vertex has only one rule; otherwise, there exist multiple rules that have the same initial graph, and thus these rules are non-deterministic. This tree is also used for deciding messages to communicate between processes and communication routes to traverse the necessary processes for deciding the rule to be applied. An edge in the classification tree generates a message.

Figure 9 shows example STR rules and their classification tree.

Step 2: Next, we generate process behavior patterns called PBPs. Each vertex in a classification tree

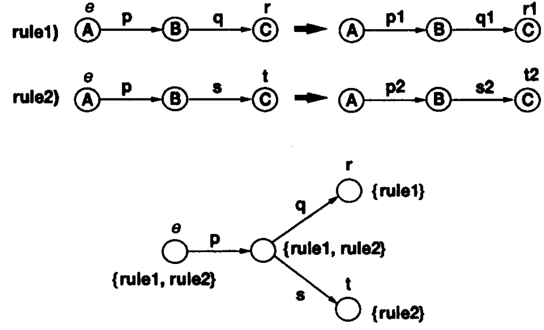

Figure 9: An example classfication tree.

generates a PBP. A PBP consists of an initial state, a transition and a subsequent state. Communications between processes are inserted into transitions. In this step we generate all message communications in order to decide the rule to be applied when an event occurs. In a message, a set of candidate rules is coded to a unique identifier.

Figure 10 shows PBPs of the rule 1 in Fig. 9. Generated messages **n1** and **n2** are used in these PBPs.

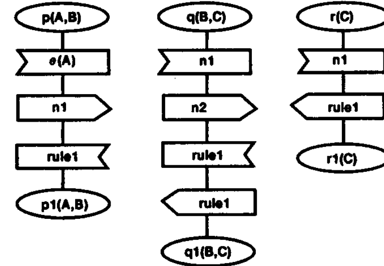

Figure 10: Example PBPs.

Step 3: Then, an abstract process specification is generated by synthesizing PBPs . The synthesis starts from the idle state. If a local state is generated, then we synthesize every PBP whose initial state is included in the generated local state. In this synthesis all the state primitives that are included in the generated local state and that coincide with the initial state of the PBP to be synthesized are replaced with the next state primitives of the PBP. This synthesis is repeated until no new local state is generated. The generated abstract process specification satisfies all of

the described STR rules. Therefore, the process specification can behave in all situations described by the STR rules.

Step 4: Finally, tasks described by the STR/D rules are inserted into the abstract process specification. The location where tasks are inserted into the abstract process specification is determined by "position-description" in an STR/D rule. When all tasks in the STR/D rules are inserted into the abstract process specification, we can get an implementable process specification satisfying all STR and STR/D rules.

4.2 Architecture-dependent refinement

We show a method of generating software conforming to any functional model from STR and STR/D rules. First we obtain "service specifications" of each functional entity from STR rules which specify terminal behaviors, and STR/D rules. Then we synthesize process specifications from the obtained service specifications of functional entities. In this method, we use intermediate language STR(L) and STR/D(L). They have the same syntax as STR and STR/D, but they specify local specifications of one functional entity.

Specification generation of functional entities consists of the following three steps:

Step 1: Event assignments to functional entities:

The events described in STR rules are manually assigned to functional entities where they truly occur.

Step 2: STR(L) and STR/D(L) rule generation from the original STR and STR/D rules, and event assignments:

First we compare conditions described by the initial-global-state, event and next-global-state in an STR rule and position-designations in STR/D rules. If the conditions of the STR rule match those of the STR/D rules, we combine these rules to obtain new global state transition rules. The obtained global state transition rules consist of four elements: the revised initial-global-state, event, task-designation, and the revised next-global-state.

Next we divide each obtained global state transition rule into local state transition rules for each functional entity used in executing the tasks in the task-designation; however, the states of the generated local state transition rules specify global states originated in STR and STR/D rules. In this division, communication actions are divided into two types: send action and receive action. The event assigned to one of the functional entities in Step 1 is specified in the assigned functional entity.

Finally, we can generate STR(L) and STR/D(L) rules from the obtained local state transition rules. In this generation, the send action generates a new STR/D(L) rule for the appropriate entity, and the receive action generates the event of a new STR(L) rule for the appropriate entity.

Step 3: Entity specification generation:

From the generated STR(L) and STR/D(L) rules for each entity, an FSM based entity specification is generated by using the same method of **Step 3** in the architecture-independent refinement.

The above functional entities specification generation can be applied when primitive send and receive actions can be extracted from the tasks to be executed in functional entities.

4.3 An example

We apply the above architecture-dependent refinement procedure to generate entity specifications from the specifications in Fig. 6 and Fig. 8.

Step 1: The events described in the rules in Fig. 6 are manually assigned to functional entities as follows:

```
uptreq(A)       :  CCAF
idnumber(A,U)   :  SRF
acode(A,C)      :  SRF
```

Step 2: We now generate new STR(L) and STR/D(L) rules according to the assignment obtained in Step 1. "s1(A)", "s2(A)" are newly generated state primitives. The following rules are the STR and STR/D rules for the entity SCF. Note that each generated STR rule specifies a local state transition of one functional entity.

```
dial-tone(A) InitialDP(A): s1(A).
s1(A) AssistReq(A): s2(A).
s2(A) CollectedUserInf(A): ident(A).

transition(-(dial-tone(A)) +(s1(A)))
{ send(SSF/CCF,ReqReport);
  send(SSF/CCF,TempConnect); }
transition(-(s1(A)) +(s2(A)))
{ send(SRF,PromptCollect,
      "Provide your identity"); }
transition(-(s2(A)) +(ident(A)))
{ send(SRF,PromptCollect,
      "Provide your authentication code"); }
```

Step 3: Finally, we generate entity specifications from that generated STR(L) and STR/D(L) rules.

5 Conclusion

A stepwise telecommunication software generation method from observable terminal behaviors is proposed to implement service specifications on any functional model. The specifications are automatically refined only if a service designer decides where the events are to occur in the functional model, if primitives send and receive can be extracted from the tasks described in STR/D rules. The refined specifications are automatically transformed into functional entity specifications. This stepwise software generation method enables us to describe service specifications and communications system specifications independently. Therefore, a service designer who describes service specifications by STR rules does not have to have the detailed knowledge of communications systems.

Acknowledgments

We sincerely thank Dr. Kohei Habara, Chairman of the Board of ATR Communication Systems Research Laboratories, for his guidance and encouragement in this research. We also wish to thank Dr. Nobuyoshi Terashima, President of ATR Telecommunication System Research Laboratories, and our colleagues for their helpful discussions.

References

[1] CCITT, revised Recommendation Z.100, "CCITT Specification and Description Language (SDL)," May 1992.

[2] M. Faci, L. Logrippo and B. Stepien, "Formal Specification of Telephone Systems in LOTOS: the Constraint-Oriented Style Approach," Comput. Networks and ISDN Syst., vol. 21, pp. 53 - 67, 1991.

[3] L. Drayton, A. Chetwynd, and G. Blair, "Introduction to LOTOS through a worked example," Computer Communications, vol. 15, no. 2, pp. 70 - 85, Mar. 1992.

[4] S. Aggarwal, R. P. Kurshan, and K. K. Sabnani, "A calculus for protocol specification and validation," in Protocol Specification, Testing, and Verification, 3. Amsterdam, The Netherlands: North-Holland, 1983, pp. 19-34.

[5] E. J. Cameron, D. M. Cohen, T. M. Guithner, W. M. Keese, Jr., L. A. Ness, C. Norman, and H. N. Srinidhi, "The L.0 Language and Environment for Protocol Simulation and Prototyping," IEEE Trans. on Compu., vol 40, no. 4, Apr. 1991.

[6] J. J. P. Tsai, T. Weigert, H.-C. Jang, "A hybrid Knowledge Representation as a Basis of Requirement Specification and Specification Analysis," IEEE Trans. on Software eng., vol. 18, no. 12, pp. 1076 - 1100, Dec. 1992.

[7] G. v. Bochmann and R. Gotzhein, "Deriving protocol specifications from service specifications," in Proc. SIGCOMM'86, 1986, pp. 136 - 145.

[8] P. M. Chu and M. T. Liu, "Synthesizing Protocol Specifications from service specification in the FSM model," in Proc. Comput. Networking Symp., Apr. 1988, pp. 505 - 512.

[9] K. Saleh and R. Probert, "A Service-Based Method for the Synthesis of communications protocols," Int. J. Mini and Microcomput. Special Issue on Distributed Systems, vol. 12, no. 3, pp. 97 - 103, 1990.

[10] R. L. Probert, and K. Saleh, "Synthesis of Communication Protocols: Survey and Assessment," IEEE Trans. Comput., vol. 40, no. 4, pp. 468 - 476, Apr. 1990.

[11] Y. Hirakawa, T. Takenaka "Telecommunication Service Description Using State Transition Rules,"in Proc. Sixth Int. Workshop on Software Specification and Design, Oct. 1991, pp. 140 - 147.

[12] A. Takura, T. Ohta and K. Kawata, "Task Generation Mechanisms in a Communication Software Generation," in Proc. Asia-Pacific Conference on Communications, Aug. 1993, pp. 206 - 209.

[13] K. Kawata, A. Takura, T. Ohta, "On a Communication Software Generation Method from Communication Service Specifications Described by a Declarative Language," in Proc. Fifth International Conference on Computing and Information, May 1993, pp. 116 - 122.

[14] CCITT "Draft Recommendation F.851, Universal Personal Telecommunication (UPT) - Service Description," Oct. 1992.