Sindh Univ. Res. Jour. (Sci.Ser.), Vol: 36 (1) 41 - 48 (2004)

AN ALGORITHM FOR THE PROTOCOL DERIVATION PROBLEMS

M. Nawaz Brohi

Institute of Mathematics and Computer Science, University of Sindh, Jamshoro, Pakistan

Abstract

A protocol synthesis technique for developing the protocol derivation algorithm, by using a state-transition model is presented. A state-transition model, which can easily describe a service containing an infinite number of possible execution paths by using transition loops in finite state machines (FSM) and which seems to be a more natural and better understood model.
An error-recovery transformation procedure to fix the sink-state and duplicate acceptance problems is deviced.

Keywords: Sequential Processes; Finite State; Protocol Synthesis; Service Transition .

Introduction

There are two approaches to ensure correctness of computer communication protocols called analysis and synthesis. In an analysis approach, a protocol is first examined to reveal some properties, desirable or undesirable and then modified to get rid of the undesirable ones. By the synthesis approach, rules ensuring some desirable properties are enforced during the protocol design process.

The synthesis approach has the advantage over the analysis approach in that it can assist the protocol designer to reduce the possibilities of making errors, if not to prevent it totally during the protocol design process. In this paper we will briefly survey the work done previously by researchers in the area of protocol synthesis, discuss the limitations of current protocol synthesis techniques and present our protocol synthesis techniques.

The protocol synthesis techniques in service specification required category do consider service specifications in a formal manner. Merlin and Bochman's(1983) work, however, additionally requires the existence of specifications for (n-1) communicating entities, where n is the total number of communicating entities in the protocol layer of interest. Prinoth's (1982) work and Bochmann and Gotzhein's (1986) work is more ambitious since only the service specification of the interest layer is needed at the outset. Nevertheless, in Prinoth's (1982) work, some auxiliary actions are in some cases, needed to be added into the service specification prior to the application if his protocol construction algorithm; yet the algorithm dose not provide a method to perform the refinement of the service specification by including such auxiliary actions. In Bechmann and Gotzhein's (1986) work, the required sychronization messages are derived automatically; however, their service specification language is not able to express a service containing an infinite number of possible execution paths.

The protocol synthesis techniques in no service specification required category provide some rules or methods for obtaining the complete protocol specification, starting from a partial protocol specification, either interactively or fully automatically. However, they don't have a service specification initially given as a reference. The protocol designer is responsible for the semantics of the synthesized protocol specification thus he or she must resort to his or her intuitive understanding of the intended service, a very informal task in

current protocol design. As a result, more burden is placed on the protocol designer in the stage of protocol verification.

In our protocol derivation algorithm, we follow the same approach of Bochmann and Gotzhein's (1986) work and thus inherit the advantages of this approach. But we use a state-transition model, which can easily describe a service containing an infinite number of possible execution paths by using transition loops in FSMs and which seems to be a more natural and better understood model.

There are two sections of this paper; In section-I we discusses work done previously by researchers in the area of protocol synthesis. In section-II we have presented synthesis technique for developing the protocol derivation algorithm.

Section- I: Previous Work

Previous work on protocol synthesis can be classified into two categories, depending on whether a service specification is required or not.

Service Specification Required

Protocol synthesis techniques in this category require the initial provision of a service specification to which synthesized protocol specification has to conform. The goal of these techniques is not only to construct protocol free from protocol logical errors, but also to mandate the synthesized protocol specification to conform to the given service specification. In the following, we briefly describe three such techniques.

Merlin's Submodule Construction Method

Merlin and Bochmann (1983) proposed a method of determining the specification for the missing entity from a given service specification and the specification for the remaining entities. Unfortunately the technique does not guarantee the deadlock-freedom for the synthesized protocol specification and thus must be supplemented by an analysis procedure to delete the deadlock.

Prinoth's Protocol Construction Algorithm

The input to Prinoth's (1982) protocol conversion algorithm is actually a specification by adding some auxiliary action transitions and the output from the algorithm is a protocol specification. Therefore, the protocol designer has to refine the service specification to produce the input to the algorithm. The algorithm itself does not include a method to perform the refinement of the service specification.

Bochmann's Protocol Derivation Algorithm

Bochmann and Gotzhein (1986) proposed an algorithm to derive a protocol specification from a given service specification. A service in their model is described by an expression of service primitives connected by sequence, parallelism and alternative operators.

No Service Specification Required

Protocol synthesis in this category do not require the initial existence of a service specification to which the synthesized protocol specification has to conform. Therefore, the protocol designer is responsible for the semantics of the synthesized protocol specification. The goal of these techniques is to construct protocol specification free from the following logical errors: nonspecified reception, nonexecutable interaction, deadlock, unboundedness and improper termination. Each technique has achieved either a portion or the whole of the goal. Generally speaking the techniques achieving just a portion if the goal have higher flexibility than those achieving the whole of the goal. Seven techniques are

included in this category, each of which is discussed in the following.

Zafiropulo's Reception Production Rules

It proposed three reception production rules, which were used in an interactive protocol synthesis system. As long as these rules are obeyed, two protocol logical errors; unspecified reception and nonexecutable interaction, can be prevented for any synthesized protocol specification. These rules, however, are only applicable to two-entity protocols.

Sidhu's Protocol Design Rule

Sidhu (1982) proposed four protocol design rule that can be used to monitor all kinds of protocol logical errors. However, the protocol designer has to specify all the interactions (message transmission and receptions) between communicating entities. Thus the technique is just an algorithm to validate a protocol in the process of designing and is not a real synthesis technique. The internal representation of protocol behavior in the technique is a global state-transition graph.

Zhang's Protocol Synthesis Algorithm

The protocol synthesis algorithm proposed consists of three production rules and two deadlock avoidance rule. Like Sidhu's protocol design rule, the internal representation of protocol behavior in his algorithm is a global state-transition graph. Their technique can be considered as an improvement over Sidhu's technique in that they enhanced Sidhu's technique by automatically generating the specifications of all receptions that can occur and by adding deadlock avoidance rules to prevent possible occurrence of deadlock. Their technique is restricted to two-entity protocols and it is suspected that the deadlock avoidance rules are not general enough to cover all deadlock-free-two-entity protocols.

Choi's Sequence Method

Choi (1986) presented a method for constructing protocol specifications in the Finite State Machine (FSM) model by first synthesizing a pair of regular expressions of star height zero or one and than converting the regular expressions to equivalent FSMs. His method can prevent all kinds of protocol logical errors mentioned above. However, his technique is limited to two-entity protocols whose entity FSMs correspond to regular expressions of star height al most one.

Gouda's Synthesis Algorithm

Given a partial specification of a communicating entity, the algorithm proposed by Gouda and Yu (1984) enforces a fixed communication pattern between two communicating entities in order to construct the complete protocol specification in which all kinds of design errors are not existent. One advantage of their algorithm is that all generated specification for the peer entity is just one of the possible correct specifications and may not be the one intended by the protocol designer. Furthermore, the algorithm is applicable only to two-entity protocols.

Ramamoorthy's Automated Protocol Synthesizer

The automated protocol synthesizer was developed by Ramamoorthy and Dong (1982) Ramamoorthy *et al.* (1985) makes use of six transformation rules to build up the specification for the peer entity from a given specification for the local entity. All kinds of design errors can be prevented by this synthesizer, if the specification for the local entity processes have some desirable properties. The synthesizer, suffers the same drawbacks as Gouda's and Yu's (1984) algorithm.

Section II: Our Synthesis Technique For Developing The Protocol Derivation Algorithm.

We believe the right approach to protocol design should be one that treats the service concept formally. In particular, we feel that one should start formal specifications of the (N)- service and the (N-1) – service to construct the desired formal specification of the (N)-protocol, as depicted in Fig.1. Within this architectural view, we are interested in automating the procedure of deriving a protocol specification from given service specifications. That is, we want to find an algorithm for the protocol derivation problem. However this protocol derivation procedure for an arbitrary communication service appears to be formidably difficult. As a result, we concentrate on a class of communication service whose behavior can be described by a set of directly coupled FSMs. This state– transition model allows the specification of both terminating and nonterminating communication service. For a service specified in the state-transition model, we provide a protocol derivation algorithm that produces the protocol specification automatically once some further information about decision options and initiation options is given by the protocol designer. The provision of the above information is to make sure that the derived protocol specification is desired by the protocol designer.

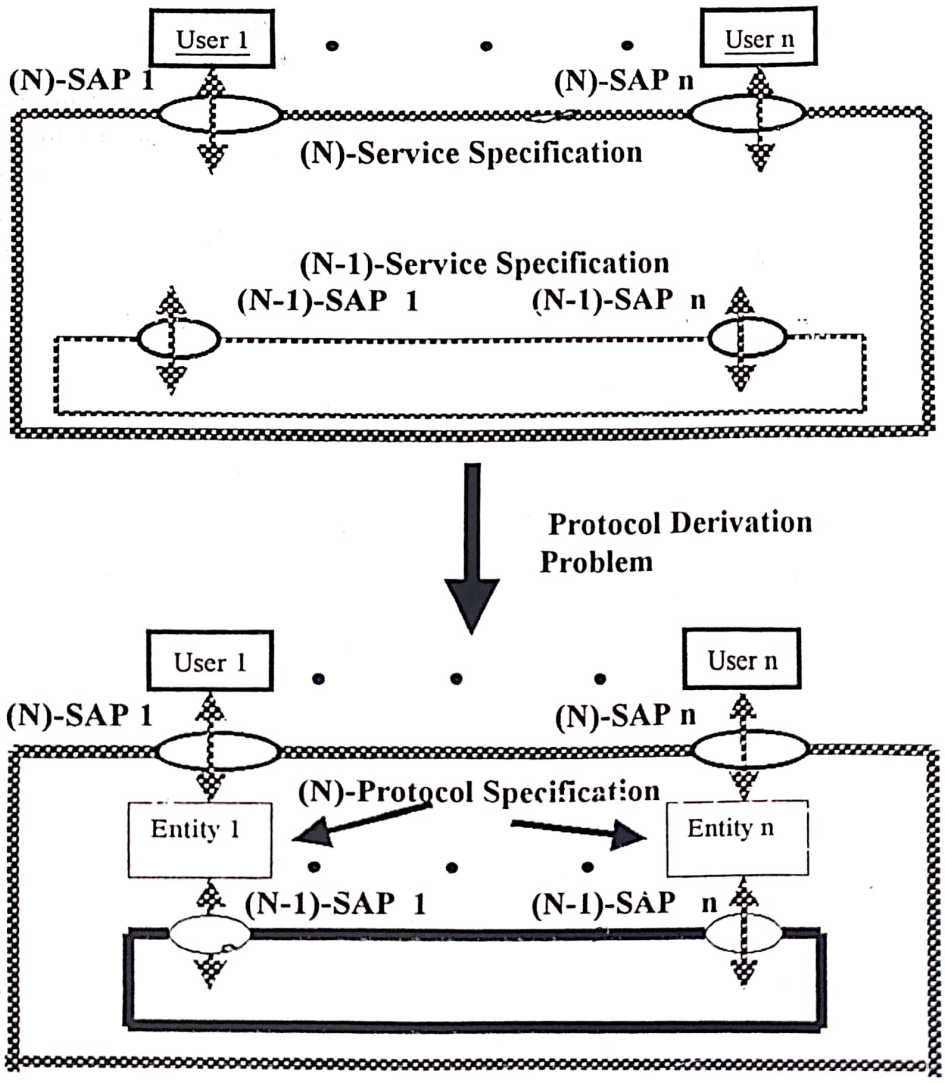


Fig. 1. From the (N) – service specification and the (N -1) – service specification to the (N) – protocol specification.

The Model

A service specification in our model is composed of local constraint FSMs and global constraint FSMs, directly coupled with one another. One example is the connection establishment and release phases of the simplified ISO transport service, as specified using the modified Communicating Sequential Processes (CSP) of Liu and Liu (1984) without the provider – initiated disconnections. In this service specification (see Fig. 2) there are five service FSMs: two local constraint FSMs, M1 and M2; and three global constraint FSMs, N1 N2 and N3.

Using a set of directly coupled service FSMs to specify a service may result in an inconsistent description; therefore we provide an "inconsistency checking'' algorithm to delete any inconsistency. An inconsistent nonterminating service specification is one that may deadlock, whereas an inconsistent terminating service specification is one that may reach a global state from which no final global state can be reached (called improper termination). The inconsistency checking algorithm actually constructs the reach ability graph in which the deadlock (or improper termination) is checked.

A protocol specification consists of two entity specification, each of which, similar to a service specification , contains local protocol FSMs and synchronizing protocol FSMs, directly coupled with one another.

Protocol Derivation Algorithm

In deriving a protocol specification from a given specification, the local constraint FSMs of the service specifications can be embedded directly into entity specifications as the local protocol FSMs since local constraint FSMs perform decision locally without requiring any communication between entities. On the other hand global constraint FSMs enforce the relative execution order of service primitives associated with different Service Access Points (SAPs), requiring protocol entities serving different SAPs to communicate with each other to synchronize the execution order of service primitives. The algorithm to derive the synchronizing protocol FSM pair (two synchronizing protocol FSMs, one for Entity 1 and other for Entity 2) from a global constraint FSM has three major steps:

1. Insert some intermediate transitions between service primitive transitions according to the specified decision option of a service state in a global constraint FSM.

2. Adjust the initial state pointer according to the given initiation option.

3. Project the resultant refined FSM onto Entity 1 and Entity 2 independently produce the desired synchronizing protocol FSM pair.

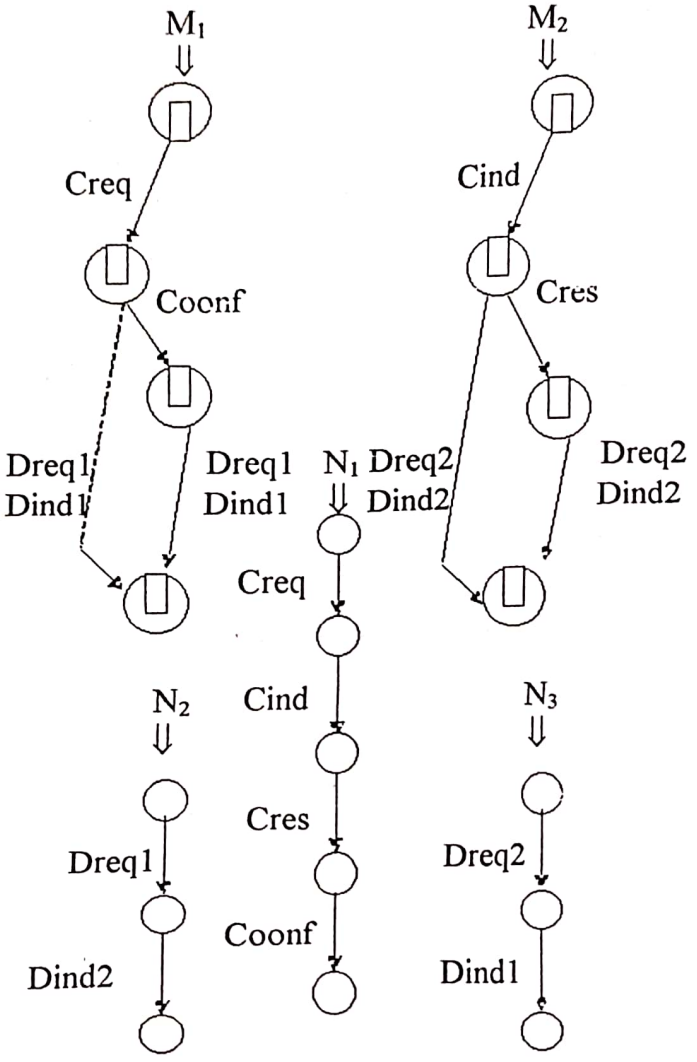
Error – Recovery Transformation

To enable our algorithm to deal with erroneous underlying communication service, we further devise an error-recovery transformation procedure. The error-recovery transformation procedure consists of three transformation rules applicable to three different patterns of transformations in the synchronizing protocol FSM produced by the protocol derivation algorithm from a service specification.

A problem called the sink– state problem, has been created by sink states of synchronizing protocol FSMs in the error–recoverable protocol produced by applying the error- recovery transformation to a protocol derived from the protocol derivation algorithm. The problem can be fixed by forcing an entity to send a "sink command'' to the other entity once it reaches a sink state. This repair corresponds to another transformation working on the portions of an error-recoverable protocol

specification that cause the sink-state problem.

The duplicate acceptance problem would result from applying the error-recovery transformation to the protocol produced by the protocol derivation algorithm. This problem can be resolved by performing another transformation on any error-recoverable protocol produced by the protocol derivation algorithm and the error-recovery transformation procedure.



Service primitive P	Sap (P)
Creq - Connection reques (from User 1)	SAP – 1
Cind - Connection indication (to User 2)	SAP – 2
Cres - Connection response (from User 2)	SAP – 2
Coonf – Connection confirmation (to User 1)	SAP – 1
Dreq 1 – Disconnect request from User 1	SAP – 1
Dind 2 - Disconnect indication to User 2	SAP – 2
Dreq 2 – Disconnect request from User 2	SAP – 2
Dind 1 – Disconnect indication to User 1	SAP – 1

⇒ : initial state pointer

○ : final state

Fig. 2. The specification of the simplified ISO transport service.

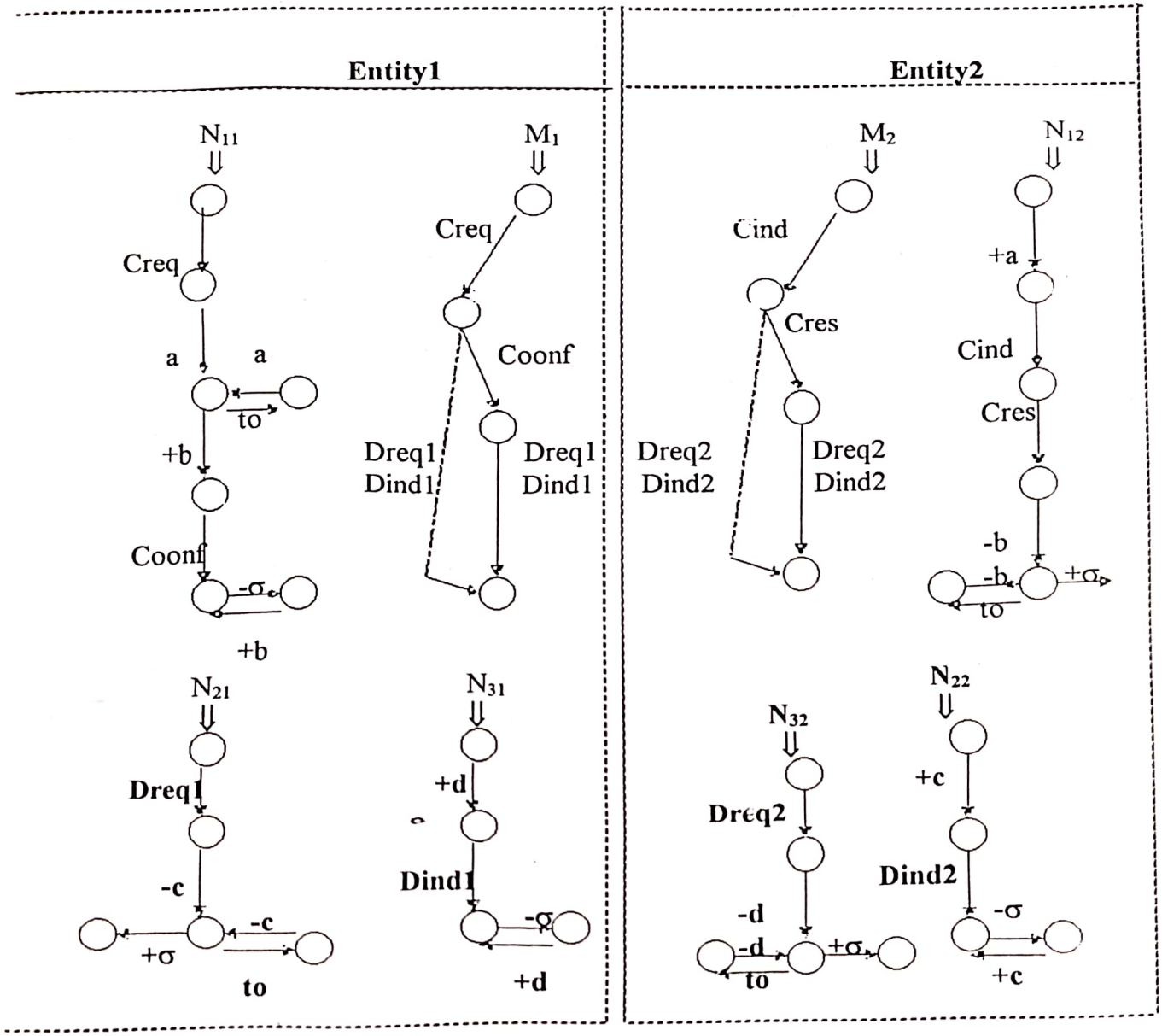


Fig. 3. Error-recoverable protocol for the simplified ISO transport service.

Conclusions

We have developed a protocol derivation algorithm, an error-recovery transformation procedure and transformations to fix the sink-state problem and the duplicate acceptance problem, all of which are based on the state-transition model.

We have applied the protocol derivation algorithm and the error-recovery transformation procedure to the simplified ISO Transport Service as shown in Fig.2.We obtain the protocol specification shown in Fig. 3, where M1 and M2 are local protocol FSMs for Entity 2, respectively and the rest are synchronizing protocol FSMs for Entity 1 or Entity 2.

References

Bachmann, G. V., and R. Gotzhein (1986) Deriving protocol specifications from service specifications. Proc. ACM SIGCOMM 86 Symp., pp. 148 -156.

Choi. T. Y. (1986) A sequence method for protocol construction Proc. IFIP Int. workshop on Protocol Specification, Testing Verification. 6th. pp. 307-321.

Chu. P. M. (1989) Towards automating protocol synthesis and analysis Ph D. Dissertation. Ohio. state University, Columbus, Ohio.

Gouda, M. G. and Y. T. YU (1984) Synthesis of communicating finite-state machines with guarantees progress. IEEE Trans. Comm. COM-**32** (7): 779 -788.

Liu, N. C. and M. T. Liu (1984) CSP-based specification for network protocols and service. Proc. IEEE Computer Networking Symp. pp. 95 -102.

Merlin, P. M. and G. V. Bochmann (1983) On the construction of submodule P specification, P and P communication protocol, A CM TOPLAS, **5** (1): 1-25.

Prinoth, R. (1982) An algorithm to construct distributed systems from state-machines. Proc IFIP Int. workshop on Protocol Specification, Testing and Verification, 2nd, pp. 261-282.

Ramamoorthy, C. V. and S. T. Dong (1982) Communication protocol synthesis. Proc. IEEE COMPSAC, pp. 217-225.

Ramamoorthy, C. V., S. T. Dong, and Y. Usuda, (1985) An implementation of an automated protocol synthesizer (APS) and its application to the X. 21 protocol. IEEE. Trans. Software Engineering Se-**11** (9): 886 -908.

Sidhu, D. P. (1982) Protocol design rules. Proc. Int. Workshop on Protocol Specification, Testing and Verification. 2nd. pp. 283-300.