

Formal Methods for Generating Protocol Conformance Test Sequences

ANTON T. DAHBURA, MEMBER, IEEE, KRISHAN K. SABNANI, SENIOR MEMBER, IEEE, AND M. ÜMIT UYAR, MEMBER, IEEE

Invited Paper

The wide range and high complexity of services expected from a communication protocol have made the process of protocol conformance testing a challenging task. Formal methods are required to thoroughly test these complex protocols. This paper reviews the four major methods of conformance test generation reported in the literature: transition tours, distinguishing sequences, characterizing sequences, and unique input/output sequences. These methods are used to test the control portion of a protocol specification. The conformance testing concepts developed in the standards world are also summarized. Their relationship with the four formal methods is discussed.

I. INTRODUCTION

Far-reaching technological advances in communication networks have enabled the interconnection of heterogeneous systems in order to provide services such as data transfer and sharing of distributed resources. A *protocol* defines a set of rules for the communication among these systems. Reliable communication can be ensured among such systems if the protocol implementations used within each system conform to their specifications. As the services expected from a communication network increase, the complexity of the communication protocols also increases, making the task of *conformance testing* more challenging and most essential. Hence, protocol conformance testing has become an integral part of the communication system design.

There are two key issues regarding the conformance testing of complex protocols. The first issue is the preparation of the conformance tests, which needs to be complete in the coverage of all aspects of a protocol to be tested. Secondly, the time required to run such tests in a testing laboratory should not be unacceptably long. The need to combine these two conflicting constraints of conformance testing has led to the development of formal methods to automatically generate the conformance tests. In this paper,

various test generation techniques reported in the literature are discussed. The concepts developed in the standards bodies are also summarized, and their relationship with the formal test generation methods are discussed.

During the conformance testing of a protocol implementation, an external tester applies a sequence of inputs (e.g., messages, packets, etc.) to the implementation and verifies that the behavior of the implementation is as described in its specification. This exercise is complicated by the limitations on the controllability and the observability of the protocol implementation. In most cases, because of the limited controllability, the implementation cannot be directly put into a desired state, usually requiring several additional state transitions. Unless efficient solutions are found, this limitation may result in test sequences with infeasibly large numbers of state transitions. Limited observability prevents the external tester from directly observing the state of the protocol implementation, which is critical for a test to detect errors. Considering that even the simplest protocol may admit an astronomical number of different input sequences, these limitations make the conformance testing problem especially challenging. Test generation in the past has usually used ad hoc techniques. There are two major problems with ad hoc methods: (a) test generation is time consuming; (b) tests are typically very long and do not cover all essential parts of the behavior.

Protocols are typically modeled as extended finite-state machines [1] where the control portion is a finite-state machine and the data portion consists of program segments. In this paper, we will focus on the conformance testing for the control portion of a communications protocol. Some work on generating test sequences for the data portion has used a technique in software testing called *data flow graph testing* [2]. Ural studied the application of data flow techniques to select appropriate tests for data portion of protocol implementations [3]. The control portion of a protocol (henceforth referred to as *the protocol* for simplicity) can be specified as a *deterministic finite-state machine* (FSM) [4], [5], described in the following. The *state* of a protocol is defined as a stable condition in which the protocol rests until a stimulus, called an *input*, is applied.

Manuscript received November 17, 1988; revised June 7, 1990.
A. T. Dahbura and K. K. Sabnani are with AT&T Bell Laboratories, Murray Hill, NJ 07974, USA.
M. Ü. Uyar is with AT&T Bell Laboratories, Holmdel, NJ 07733, USA.
IEEE Log Number 9037866.

The protocol generates a response to the stimulus, called an *output*, (which may be null) when an input is applied, and moves into a new state (which may be the same as the previous state) where it stays until the next input. The purpose of conformance testing is to check whether behavior of the implementation of an FSM is as defined by the specification.

Typically, formal conformance testing techniques generate a set of input sequences that will force the FSM implementation to undergo all specified transitions. These techniques can be classified as the *transition tour method* [6]–[8], the *distinguishing sequence method* [5], [9]–[12], the *characterizing sequences method* (also known as the *W-method*) [5], [13], and the *unique input/output sequences method* [14]–[16]. All of these techniques assume the so-called *black box* approach where only the outputs generated by the implementation (upon receipt of inputs) are observable to the external tester.

The transition tour method generates a state tour that exercises every state transition of the implementation [6], [7] and does not address the observability problem described above. For certain protocols which have a special message to determine the state of the protocol (i.e., the observability problem is solved by the specification) the length of the tour can be minimized by the technique given in [8], which is based on a graph theoretic concept called the *Chinese Postman Problem* [17].

The remaining three methods emphasize the observability problem. In the distinguishing sequence method, an input sequence is found for a protocol such that the outputs generated by the implementation will identify its state. The requirement for this method is a fully specified protocol, which may be too strong for most actual protocols (i.e., many actual protocols do not possess distinguishing sequences). The characterizing sequence method defines a set of input sequences for a subset of states such that the resulting set of output sequences ultimately distinguishes each state from the others. In both of these methods, the current state of the implementation is assumed to be unknown and the sequences generated by them are powerful enough to find the current state. In other words, both the distinguishing and characterizing sequences answer the question of “*what is the current state of the implementation?*”. However, in the UIO sequences method, this question is relaxed as “*is the implementation currently in state x ?*” which results in much shorter sequences than the other two methods. A technique for minimizing the length of the test sequence generated by the UIO method is given in [16] which is based on a more general form of the Chinese Postman problem called the *Rural Chinese Postman problem*. Experience with this method indicates that the test sequences generated are about one-third the size of those generated by ad hoc methods [16].

While researchers have been involved in developing formal methods for conformance test generation, standards organizations such as the International Organization for Standards (ISO) and the International Consultative Committee for Telephones and Telegraphs (CCITT) have formalized concepts regarding the implementation of tests in a laboratory within a testbed. Among the standardized aspects of conformance testing are the testbed architectures and their implementation, test description languages, and the evaluation of the test results. In this paper, these

concepts are briefly reviewed. We also give the relationship between these concepts and the four formal methods discussed earlier.

The rest of the paper is organized as follows. Section II describes the test generation techniques in detail. Section III gives an outline of the basic terms and concepts developed by the standards organizations. Section IV contains concluding remarks and describes some important unresolved issues.

II. FINITE-STATE MACHINE TESTING

Formal methods for test generation for control portion of a communication protocol have been based primarily on the so-called *finite-state machine* (FSM) model [5], in which the machine M to be tested is represented by a directed graph $G = (V, E)$. The set of vertices $V = \{v_1, \dots, v_n\}$ represents the set of specified states $S = \{s_1, \dots, s_n\}$ of M and a directed edge $(v_i, v_j) \in E$ represents a transition from state s_i to state s_j in M .

There is assumed to be a set of input and output operations for every state of M , called the *permissible input set* I and *permissible output set* O , respectively. Thus, when M is in state s_i and a specified input $a_k \in I$ for s_i is applied, M produces a specified output $o_l \in O$ and moves into state s_j . Note that G may contain multiple edges from v_i to v_j corresponding to multiple transitions with distinct inputs, all of which take M from s_i into s_j . Also, an input may cause M to remain in the same state s_i ; this is represented by a self-loop at v_i .

An edge in G is represented by a triple (v_i, v_j, L) , where $L = a_k/o_l$ is the input/output operation corresponding to the transition from s_i to s_j in M . A non-negative, real-valued cost $c(v_i, v_j; L)$ may be associated with edge $(v_i, v_j; L)$, where $c(v_i, v_j; L)$ is the time required to realize the corresponding transition in M . This becomes an important consideration in reducing the total time necessary for executing a test sequence. The formal methods developed thus far assume that M is a *deterministic* FSM: that is, the behavior of M in a given state for a given input is predictable and unambiguous. Also, it is assumed that each state can be reached from any other state via some valid sequence of inputs; that is, it is assumed that G is *strongly connected*.

A specification of an FSM M is said to be *fully specified* if the behavior of M for every given state and input stimulus $a_k \in I$ is defined. In this case, the graph representation G of M contains exactly $|I|$ edges leaving each vertex $v_i \in V$. Otherwise, M is said to be *partially specified*.

The *start state* $s_0 \in S$ of M is a reference state for M , usually the state M is in immediately following power-up. Often, there is a special input which takes M to state s_0 from any other state with a single transition. In this case, M is said to have the *reset capability* and the input which performs the reset is denoted by “*ri*.”

An example of a graph representation of an FSM is shown in Fig. 1. The start state is assumed to be state s_1 , reset edges are not shown for simplicity, and each edge is assumed to be of unit cost. Note that the FSM is deterministic, strongly connected, and partially specified.

The *protocol conformance testing problem* is to decide whether a given black box implementation of an FSM behaves in accordance with, or *conforms to*, a given FSM specification. Equivalently, an implementation of an FSM conforms to a specification if and only if, starting from the

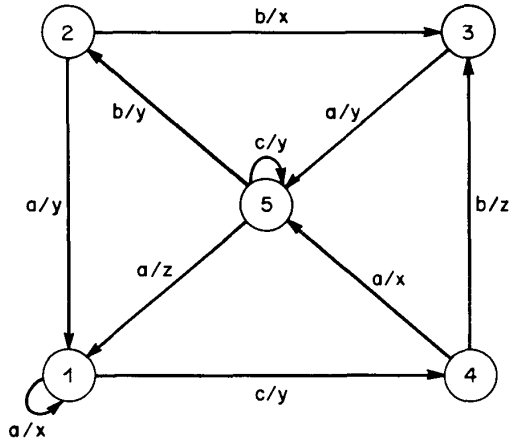


Fig. 1. A graph representation of a finite-state machine.

start state, the implementation produces the same output sequence as its specification for every possible specified sequence of inputs.

The methods to be described here for protocol conformance test sequence generation are based on a *transition-level* approach. Each method produces a test sequence which checks the correctness of each transition of the FSM implementation. The methods are by no means exhaustive in that an implementation which exhibits correct behavior for a test sequence is not *guaranteed* to exhibit correct behavior given every possible input sequence; such a test sequence is not possible to derive [5]. Instead, the intent is to design a test sequence which guarantees "beyond a reasonable doubt" that if the implementation passes a given test then it almost certainly conforms to its specification.

The process of checking a transition from s_i to s_j with input/output a_k/o_l consists of three basic steps:

- Step 1:** The FSM implementation is put into state s_i ;
- Step 2:** Input a_k is applied and the output is checked to verify that it is o_l , as expected;
- Step 3:** The new state of the FSM implementation is checked to verify that it is s_j , as expected.

Henceforth, the sequence of inputs for testing edge $(v_i, v_j; a_k/o_l)$ is denoted as *TEST* $(v_i, v_j; a_k/o_l)$ and consists of input a_k followed by the sequence of inputs necessary to realize Step 3.

The difficulty in realizing *TEST* $(v_i, v_j; a_k/o_l)$ is due to the limited *controllability* of the FSM implementation, so that it is not possible to put the implementation of M into state s_i in Step 1 of the above procedure without realizing several transitions ($n - 1$ transitions in the worst case), and due to the limited *observability* of the FSM implementation, so that it is not possible to directly verify that the implementation of M is in state s_j in Step 3 of the above procedure.

Several methodologies have been designed to overcome these limitations, and are described in the following.

Transition Tour Method

The most straightforward scheme for testing each transition of the FSM is, starting from the start state s_o , to exercise, or traverse, each of its transitions at least once and

then return to s_o . Such a traversal is called a *tour* of the FSM. Of course, the simplicity of the approach is offset by the disadvantage that, after each transition, the new state of the FSM is not checked as specified in Step 3 of the above procedure. In many cases, however, an incorrect new state eventually leads to incorrect output generated later in the test sequence, thereby detecting the presence of an error.

Naito [6] suggested the transition tour approach for FSM representations of sequential circuits and gave a heuristic algorithm for generating a tour of the FSM. Sarikaya and Bochmann [7] were the first to observe that the transition tour method can be applied to protocol testing.

In certain protocol specifications, the FSM M has a desirable feature called a *status message*: for each specified state s_i of M , there exists an input called "status" by convention, which produces a unique output for s_i and causes M to remain in state s_i . In this case, Step 3 of the transition-level testing approach can be easily incorporated into the transition tour method by using the "status" input to verify the state of M after each transition has been realized for the first time. Uyar and Dahbura [8] showed that the problem of generating a minimum-cost test sequence using the transition tour method is equivalent to the so-called *Chinese Postman* problem in graph theory [36], [37] and they gave an efficient algorithm for generating a minimum-cost transition tour of M . They also showed that the status message feature could be used in combination with the transition tour method to realize Step 3 of the transition-level test sequence generation approach described earlier.

For example, a minimum-cost transition tour of the FSM shown in Fig. 1 is (including reset edges), starting from state s_1 : $a/x, ri/null, c/y, a/x, b/y, b/x, ri/null, c/y, ri/null, c/y, a/x, b/y, ri/null, c/y, a/x, c/y, b/y, a/y, c/y, a/x, ri/null, c/y, b/z, a/y, a/z$. The total cost of the tour is 25.

Distinguishing Sequences

Much of the work on formal methods for protocol test sequence generation has its origins in the well-known *checking experiments* of over thirty years ago [5], [12], [39]. The checking experiment problem (also known as the *fault detection experiment problem*) is equivalent to the protocol conformance test sequence generation problem and is a special case of the much more difficult *machine identification* problem. In the machine identification problem, only a black box implementation of an FSM is given; the objective is to derive a graph G (equivalently, a state table) which accurately represents the behavior of the black box. In contrast, the objective of the checking experiment is to decide whether a given black box implementation of an FSM behaves according to a given graph representation of an FSM.

In this respect, several techniques have been developed to assist in designing "experiments" (test sequences) for the checking experiment and machine identification problem. Such procedures make it possible to identify the state of an FSM when a sequence of inputs is first applied (henceforth called the *initial state* of the FSM). One such tool is called a *distinguishing sequence* [5], [9]–[12]. A sequence of inputs $A_{ds} = a_1, \dots, a_p$ is a distinguishing sequence if the output sequence produced by an FSM M in response to A_{ds} is distinct for each initial state s_i . For example, the sequence $A_{ds} = a, c, a$ is a distinguishing sequence for the finite-state

machine shown in Fig. 1. It can be readily verified that the sequence of outputs produced by A_{ds} for each of the five initial states is distinct. Note that if an FSM has the status feature described previously then the input "status" is a distinguishing sequence for the FSM.

A distinguishing sequence is a useful tool for achieving Step 3 of the three-step testing procedure outlined earlier. One straightforward way to use distinguishing sequences as the basis of a test sequence for protocol conformance testing is as follows: For each transition $(s_i, s_j; L)$ in M , the FSM M is taken to state s_i , the appropriate input is applied and corresponding output is observed, and the new state s_j is checked by means of a distinguishing sequence of M .

There are two severe drawbacks with the distinguishing sequence approach: 1) in practice, very few FSMs actually possess a distinguishing sequence, rendering the concept to be primarily of academic interest (see Fig. 2), and 2) even

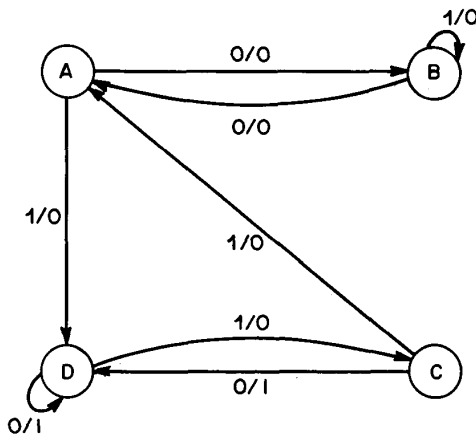


Fig. 2. A finite-state machine which does not possess a distinguishing sequence.

if an FSM does have a distinguishing sequence, the best known upper bound on the length of the distinguishing sequence is $(n - 1)n^n$ [5], where n is the total number of states in M , which is much too large to be useful in general.

Two other concepts which emerged from the research effort on checking experiments and which are worthy of note are *homing sequences* and *synchronizing sequences*. An input $A_{hs} = a_1, \dots, a_q$ is a homing sequence of M if the final state of the machine can be determined uniquely from the FSM's response to A_{hs} , regardless of the initial state [5]. An input sequence $A_{ss} = a_1, \dots, a_r$ is a synchronizing sequence of M if A_{ss} takes M to some final state regardless of the output or initial state of M . Note that a synchronizing sequence is also a homing sequence but not vice versa.

As in the case of distinguishing sequences, not all FSMs actually possess a synchronizing sequence, although the *reset feature* described earlier turns out to be extremely useful since it is a synchronizing sequence and in fact has been studied for use in conjunction with distinguishing sequences to generate test sequences in the following way: in order to test transition $(s_i, s_j; L)$ in M , the synchronizing sequence "ri" takes M to state s_i , a minimum-cost input sequence takes M to s_j , the appropriate input is applied and corresponding output is observed, the distinguishing

sequence for M is applied to verify that the new state of M was s_j as expected, "ri" is applied, and so on. The minimum-cost input sequence taking M from one state s_1 to another state s_2 is called a *transfer sequence* and denoted as $T(s_1, s_2)$.

For the example FSM shown in Fig. 1, let $T(s_1, s_2) = c/y$, $a/x, b/y$, $T(s_1, s_3) = c/y, b/z$, $T(s_1, s_4) = c/y$, and $T(s_1, s_5) = c/y, a/x$. Given $A_{ds} = a, c, a$, the test sequence based on the synchronizing sequence "ri," the given transfer sequences, and the given distinguishing sequence is (tested transition in boldface):

ri/null,
a/x, a/x, c/y, a/x, ri/null,
ri/null, a/x, c/y, a/x, ri/null,
c/y, a/x, c/y, a/z, ri/null,
c/y, a/x, b/y, **a/y**, a/x, c/y, a/x, ri/null,
c/y, a/x, b/y, **b/x**, a/y, c/y, a/z, ri/null,
c/y, a/x, b/y, **ri/null**, a/x, c/y, a/x, ri/null,
c/y, b/z, **a/y**, a/z, c/y, a/x, ri/null,
c/y, b/z, **ri/null**, a/x, c/y, a/x, ri/null,
c/y, **a/x**, a/z, c/y, a/x, ri/null,
c/y, **b/z**, a/y, c/y, a/z, ri/null,
c/y, **ri/null**, a/x, c/y, a/x, ri/null,
c/y, a/x, **a/z**, a/x, c/y, a/x, ri/null,
c/y, a/x, **b/y**, a/y, c/y, a/x, ri/null,
c/y, a/x, c/y, a/z, c/y, a/x, ri/null,
c/y, a/x, **ri/null**, a/x, c/y, a/x, ri/null.

The total cost of the test sequence is 100.

Characterizing Sequences

For FSMs which do not possess a distinguishing sequence, it still may be possible to determine the initial state of an FSM. The states of the FSM are first partitioned into *blocks* which can be distinguished by observing the sequence of outputs produced by a sequence of inputs. Each block is subsequently partitioned into distinguishable sub-blocks, and so on, until each block consists of exactly one state. As a consequence, the procedure for identifying a state s_i consists of applying an input sequence, returning to s_i via a transfer sequence, applying a second input sequence, and so on. The complete set of such input sequences for an FSM M is called the *characterizing set* W of M and the procedure is sometimes referred to as the *W-method* [5], [13].

As an example, consider the finite-state machine shown in Fig. 2. It was shown in [5] that this machine does not possess a distinguishing sequence. For the input sequence $A_{cs1} = 0, 1, 0$, the response is identical for initial states C and D (101), which is distinct from that for initial states A (000) and B (001). Note, however, that the response to the input sequence $A_{cs2} = 10$ is distinct for initial states C (00) and D (01). Therefore, one input sequence, A_{cs1} , is required to identify states A and B, while two input sequences, A_{cs1} and A_{cs2} , along with appropriate transfer sequences, are required to identify states C and D.

Unique Input/Output Sequences

In the distinguishing sequence and characterizing sequence approaches, when an initial state is not that which is expected, the resulting sequence of outputs uniquely identifies the actual initial state. For the purpose of testing, however, this is too strong a requirement; indeed, if the

initial state is not that which is expected, it is sufficient that an error has been detected.

One way to determine either that the initial state was that which is expected or that an error has occurred is to make use of the concept of *Unique Input/Output (UIO) sequences* [14], [15] (also called *Simple Input/Output sequences* in [38]). Formally, a UIO sequence for state s_i , denoted UIO_i , is a specified input sequence of minimum length $UIO_i = a_1, \dots, a_s$ with initial state s_i such that there is no $s_j \neq s_i$ for which the sequence of outputs produced by UIO_i for initial state s_j is identical to the sequence of outputs produced by UIO_i for initial state s_i .

If the initial state s_i was that which is expected, the UIO sequence for s_i produces an output sequence that could have only been produced by initial state s_i . If the initial state was not s_i , the resulting output sequence will be distinct from that for initial state s_i , but will not necessarily yield any information about the identity of the actual initial state.

For example, for the finite-state machine shown in Fig. 1, the input sequence $UIO_3 = a, a$ produces an output sequence of y, z for initial state 3. No other initial state can produce a response of y, z to UIO_3 . Furthermore, there is no input sequence of length one for initial state 3 with this property; therefore, UIO_3 is a UIO sequence for state 3. In addition, $UIO_1 = c, a$, $UIO_2 = b$, $UIO_4 = b$, and $UIO_5 = b$.

The advantages of UIO sequences over distinguishing sequences and other related techniques are twofold. First, the cost of a UIO sequence is never more than that of a distinguishing sequence and in practice is usually much less, since a distinguishing sequence is a UIO sequence but not necessarily vice versa; our own experience with several complex protocols shows that UIOs usually consist of less than five inputs, and a similar experience was reported in [39] with the PROWAY protocol. Second, nearly all FSMs have UIO sequences for each state while few have a distinguishing sequence [5], [14], [39]. Therefore, the UIO sequence approach is that of choice for executing Step 3 of the testing approach described above, and the procedure for realizing $TEST(v_i, v_j; a_k/o_i)$ for $(v_i, v_j; a_k/o_i) \in E$ is the following:

- Step 1:** The FSM implementation is put into state s_i ;
- Step 2:** Input a_k is applied and the output is checked to verify that it is o_i , as expected;
- Step 3:** The new state of the FSM implementation is checked to verify that it is s_j , as expected, by applying input sequence UIO_j and checking that the resulting output sequence is that which is expected.

In [16] it was shown that the problem of finding the minimum-cost test sequence which includes tests for each transition of the FSM is equivalent to a more general form of the Chinese postman problem, known as the *Rural Postman Problem* [40]. An efficient algorithm was given for finding the optimal test sequence for an FSM using UIO sequences under the assumption that 1) the FSM specification has the reset feature and/or 2) each state in the FSM specification has at least one self-loop. Recent experiences have shown typically a 3-to-1 reduction in the time required for testing actual communication protocols using the Chinese postman approach compared to the ad hoc methods [16].

For example, a minimum-cost test sequence for the FSM shown in Fig. 1 is as follows, where the transition being tested is in boldface:

ri/null,
a/x, c/y, a/x,
a/z, c/y, a/x,
ri/null, c/y, a/x,
c/y, b/y,
a/y, c/y, a/x,
b/y, b/x,
a/y, b/y,
b/x, a/y, a/z,
ri/null, c/y, a/x,
a/z,
c/y, b/z,
ri/null, c/y, a/x,
a/z,
c/y,
ri/null, c/y, a/x,
a/z,
c/y,
a/x, b/y,
ri/null, c/y, a/x,
a/z,
c/y,
b/z, a/y, a/z.

The total cost of the test sequence is 48.

III. OSI CONFORMANCE TESTING CONCEPTS

This section reviews the basic terms and concepts developed by the ISO and the CCITT for the conformance testing of the Open Systems Interconnection (OSI) protocol implementations. For details, the reader is referred to [18], [19]. The main focus of the OSI conformance principles is the execution of the conformance tests in a test laboratory: the definition of various abstract testbed architectures and their implementation, classification of conformance test types, interpretation of test results, and formal representation of the tests [18], [19]. The aspects that are being standardized do not include the formal methods for test generation, possibly because of the lack of a formal method that is accepted by all of the major testing organizations. Section III-B discusses the relationship between the OSI conformance testing concepts and the formal techniques discussed in Section II.

A. OSI Conformance Testing

ISO has defined the OSI reference model to represent the interconnection of heterogeneous systems [20]. The OSI reference model consists of seven hierarchical layer entities, each built upon its predecessor layer (Fig. 3). Although the number of layers may differ from network to network, the purpose of an entity at layer N is to provide certain services, called (N) -Services, to its upper layer entity using the services provided by $(N - 1)$ layer while isolating the implementation details of the lower layer entities from the upper layers. Peer (N) -Entities communicate with each other through an $(N - 1)$ -Service Provider by exchanging (N) -Protocol Data Units ((N) -PDUs) (Fig. 4). An (N) -protocol defines the rules and conventions for the communication between two (N) -Entities.

PDUs are divided into three categories: *valid*, *inopportune*, and *invalid*. A valid PDU is a peer-level message which the receiver (N) -Entity knows how to handle in its current

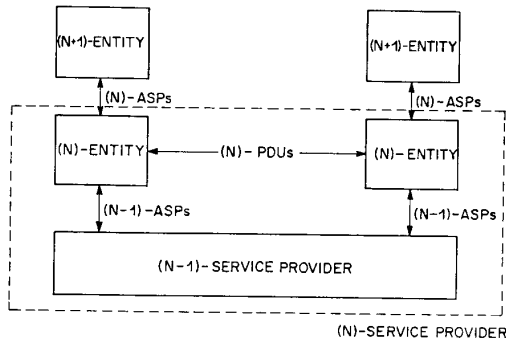


Fig. 3. Abstraction of an (N) -entity in a multi-entity OSI reference model.

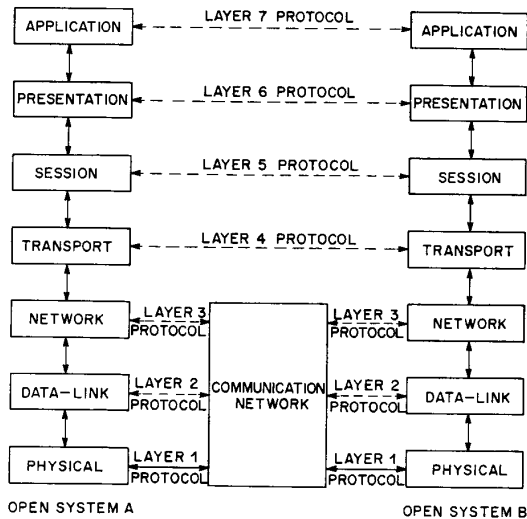


Fig. 4. Open Systems Interconnection (OSI) reference model.

state. Valid PDUs are defined for the "normal" (or expected) behavior of an (N) -Entity. Inopportune PDUs are syntactically correct, however, they arrive unexpectedly corresponding to the "unexpected" behavior of the sender (N) -entity. The receiver protocol entity does not know how to handle the inopportune PDUs in its current state. An invalid PDU does not meet the syntax requirements of the (N) -Entity and represents the "erroneous" behavior of the sender (N) -Entity.

Interactions of an (N) -Entity with its upper and lower entities are defined by (N) - and $(N - 1)$ -Abstract Service Primitives ((N) -ASPs and $(N - 1)$ -ASPs, respectively). These two sets of ASPs define the *externally controllable* and *observable* behavior of the (N) -Entity. The objective of the *conformance testing* of an (N) -Entity implementation is to determine whether the behavior of the implementation is the same as the (N) -protocol specification. During conformance testing, the ASPs and PDUs that are defined as inputs are sent to the (N) -entity implementation and the ASPs and PDUs generated by the implementation are observed. This approach in conformance testing, which considers only the externally visible behavior of an (N) -entity implementation, is called *black box testing*. Note that some implementations, due to design restrictions, may cause some ASPs to become inaccessible when the (N) -entity is installed within

a multi-layer product where the upper and lower entities surround the (N) -entity. In such cases, the observability and controllability of the (N) -entity will be partially lost. The (N) -ASPs that are implemented as an exposed interface within a multi-layer product can be controlled and observed. The points at which the occurrence of test events (i.e., the exchange of $(N - 1)$ - and (N) -ASPs and (N) -PDUs) can be controlled and observed are called the *points of control and observation* of the (N) -entity implementation.

Various abstract testbed architectures are defined by the ISO and CCITT based on the points of control and observation (i.e., availability of the ASPs) in an implementation under test (IUT). The system in which an IUT resides is called *the system under test (SUT)*. Each method is briefly described below for single-layer testing of an IUT:

Local Testing Method: In this method, points of control and observation are defined at the lower and upper interfaces of the IUT (Fig. 5). In other words, (N) -PDUs, (N) - and

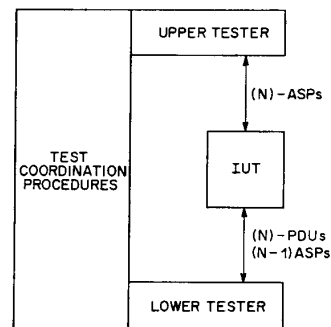


Fig. 5. Local testing method.

$(N - 1)$ -ASPs are assumed to be available such that the upper and lower testers can control and observe them. In Figure 5, the *upper tester* refers to the part of the architecture that can control and observe (N) -ASPs. It acts as the (N) -service user of the IUT. Similarly, the *lower tester* controls and observes $(N - 1)$ -ASPs and (N) -PDUs. It exchanges (N) -PDUs with the IUT, acting as a peer entity of the IUT. The test cases are interpreted and evaluated by the lower tester. During the testing, the coordination between the actions to be taken by the upper and lower testers are provided by *test coordination procedures*.

Distributed Testing Method: The lower interface of an IUT may not be an exposed interface and, therefore, $(N - 1)$ -ASPs and (N) -PDUs cannot be directly controlled and observed. In this method $(N - 1)$ -ASPs and (N) -PDUs are controlled and observed indirectly (remotely) at the opposite side of the $(N - 1)$ -Service Provider (Fig. 6). In this archi-

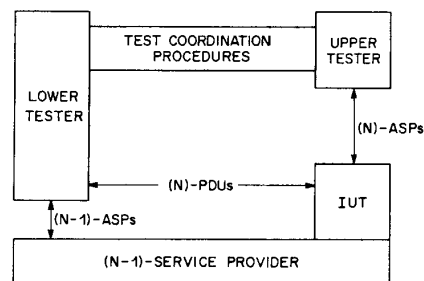


Fig. 6. Distributed testing method.

texture, lower and upper testers can reside in remote locations.

Coordinated Testing Method: In this method, which is an enhanced version of the distributed method, the control and observation of (N) -ASPs are performed by the test coordination procedures. The lower tester is considered to be the master of the upper tester. The actions of the upper tester are controlled by the lower tester through a protocol, called the *test management protocol*. The test management protocol uses Test Management PDUs (TM-PDUs) which are specifically defined to coordinate the actions between the upper and lower testers. Note that the coordinated testing method does not require an exposed interface at the upper service boundary of an IUT.

Remote Testing Method: This method is applicable to IUTs that do not have an exposed upper interface (Fig. 7).

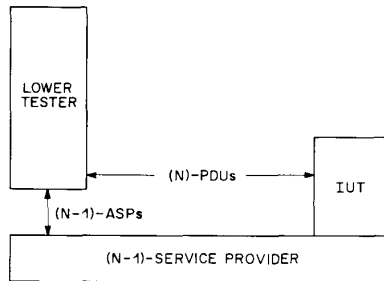


Fig. 7. Remote testing method.

In other words, (N) -ASPs cannot be controlled and observed by an upper tester. In this case, there are no test coordination procedures.

Ferry Method: In this method [21], an $(N + 1)$ -Entity, called the *ferry control protocol*, replaces the upper tester (Fig. 8).

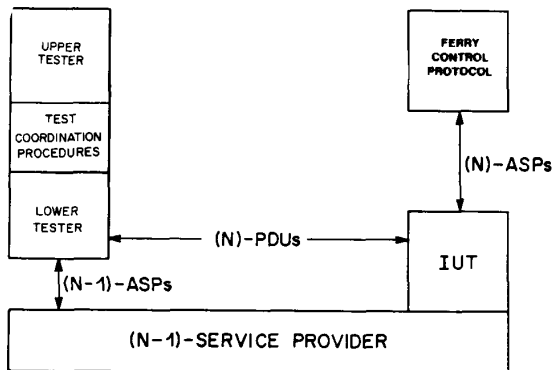


Fig. 8. Ferry testing method.

Both the upper and lower testers reside in the tester side, which simplifies the test coordination procedures. The ferry protocol provides a logical loop-back between the upper and lower testers. The (N) -ASPs are routed by the ferry protocol between the upper service boundary of the IUT and the upper tester.

The testing methods described above are abstract definitions for testbed implementations. Depending on the availability of points of control and observation in an IUT, testers can implement one or several methods in their testbeds.

The local testing method assumes that the upper and lower service boundaries of an IUT are observable and controllable. Therefore, the local testing method can test an IUT more effectively than the other methods. However, once an implementation is incorporated in a product, it may not be possible to use the local testing method, which restricts this method to in-house testing performed by implementors.

In terms of error detection capability, the distributed and coordinated methods are less effective than the local method, since the $(N - 1)$ -ASPs are controlled and observed at the remote end of an IUT (as opposed to the site where IUT resides). These methods are highly suitable for testing end-to-end protocols, such as protocols for the network layer and higher. The distributed method requires an exposed upper service boundary at the IUT. In the coordinated method, interchange of TM-PDUs needs additional requirements on the testbed, such as *out-of-band* services.

The remote testing method is the most restrictive of all methods since it assumes that (N) -ASPs cannot be controlled and observed. However, it is the easiest method to implement since it does not place any restrictions on an IUT. The remote testing method is mostly applied to interface protocols (protocols that are not end-to-end), such as X.25.

The ferry method is a distributed version of the local testing method where the synchronization between the upper and lower testers is virtually eliminated (they reside at the same site). However, it requires an exposed upper service boundary, and *out-of-band* services for the ferry protocol.

Distributed, coordinated, and remote testing methods can be also used for testing multiple layers or an embedded layer (single layer within a multi-layer system-under-test (SUT)) of an SUT. Various issues related to the implementation of these methods, such as the implementation of the upper and lower testers, test coordination procedures, and $(N - 1)$ -Service Providers, are discussed in [22]-[30].

For the conformance of an IUT to the relevant standard specification, the standards organizations have defined four test types:

Basic Interconnection Tests: provide a limited testing to assure that the IUT can establish a basic interconnection before thorough testing is performed.

Capability Tests: check that the IUT can provide the observable capabilities based on the *static conformance requirements* which are the requirements describing the options, ranges of values for parameters, timers, etc.

Behavior Tests: test the *dynamic conformance requirements* of an IUT, which are the requirements (and options) defining the observable behavior of a protocol. A large part of behavior tests, which constitute the major portion of conformance tests, can be generated by the formal methods discussed in this paper.

Conformance Resolution Tests: used to provide definite diagnostic answers to specific requirements, such as previously identified situations that may cause incorrect behavior of an IUT. For example, they provide a yes/no answer for whether a particular feature, such as reset, is implemented in an IUT.

Other Test Types: In addition to conformance testing, the following types of tests can be performed on an IUT depending on the application: *interoperability tests*, to check whether two or more implementations that pass the conformance tests can operate together, *performance tests*,

to measure the maximum throughput that can be obtained, and stress tests, to examine the behavior of an IUT under heavy load conditions.

The *test events*, atomic interactions during the conformance testing between a particular IUT and an upper or lower tester, are described in an *abstract conformance test suite*. The tests are represented in a hierarchical structure in an abstract conformance test suite. The key level is called the *test case* which has a narrowly defined purpose (e.g., the expected behavior of an IUT upon receipt of a PDU in a certain state). *Test groups* consist of several test cases according to a logical ordering of execution. A test case is divided into *test steps* each of which consists of several test events (e.g., sending a PDU).

An informal abstract test notation, called the Tree and Tabular Combined Notation (TTCN) (Annex E, Part 3 of [19]), is defined to represent the test cases in a standard format. In TTCN, the behavior and actions of an IUT during the execution of the test cases (i.e., sending and receiving various PDUs defined by the test cases) are described in a tree format.

B. Relationship between Formal Methods and the OSI Framework

The standard bodies focus on the issues related to the implementation aspects of conformance testing as outlined in the previous section. This section intends to fill the gap between the concepts used within the OSI framework and the terms used by the researchers to describe the formal test generation techniques.

The test generation methods described in Section II are typically used to generate the behavior tests for single-layer testing. Behavior tests constitute the bulk of a conformance test suite. Basic interconnection and capability tests can be easily generated by ad hoc methods. These tests are basically a starting point for the thorough testing of the protocol implementation. There are no formal methods for generating conformance resolution tests the tests listed under other test types. These tests must also be generated by ad hoc methods. The formal methods can be applied to embedded or multi-layer testing if the behavior of the IUT can be formally modeled for such cases. The test architecture in which the tests will be run can be local, distributed, coordinated, or remote. Sarikaya and Bochmann [31] have considered the problem of testing an implementation in various test architectures. They have named this task a synchronization problem, and its discussion is beyond the scope of this paper.

The test sequences generated by the formal methods can be viewed as an ordered set of test events, the smallest units of an abstract test suite. This paper uses the term *input/output operation* to represent the interaction with the upper or lower testers required for a state transition. An input/output operation corresponds to sending/receiving a single PDU or ASP to/from the IUT (i.e., a test event). PDUs and ASPs can be sent/received by an upper or a lower tester; they can be controlled/observed locally or remotely depending on the testing architecture.

It is assumed that a model representing the dynamic conformance requirements of an IUT is a finite state machine, described in the previous section. Such a model can be prepared manually or automatically from the protocol specification written in English or in one of the formal specifi-

cation languages, such as Estelle [32], LOTOS [33], or SDL [34]. In [35], a tool to generate tests from Estelle specification is reported.

Discussion of a number of issues such as obtaining adaptive test sequences (i.e., a different test sequence for different implementation options of the same protocol), modification of test sequences, and synchronization mechanisms for a particular testbed architecture is beyond the scope of this paper.

IV. CONCLUSIONS

Considering the complexity of recent communication protocols, generating conformance tests by means of manual or ad hoc techniques is virtually impossible. In this paper, four major formal test generation techniques were reviewed. Among them, the transition tours deal with the controllability aspect of the testing whereas the distinguishing and characterizing sequences emphasize the observability aspect. The UIO sequences method, combined with the Rural Chinese Postman approach, is the only one that addresses both of these issues simultaneously.

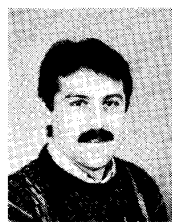
Formal test generation methods have recently been an active research area. Much work is needed to resolve various issues, a few of which are listed in the following:

- In the distributed and coordinated testbeds, formal mechanisms for synchronization between the upper and lower testers are needed. These mechanisms should be easily inserted into the abstract test suites generated by the formal methodologies. This problem has been studied by Sarikaya and Bochmann in [31].
- While work has been reported on the effectiveness of the various formal test sequence methods [41], [42], the types of discrepancies detected by each of the methods need to be characterized and quantified.
- Currently, testing typically consists of two stages: conformance testing and interoperability testing. Research is needed to find out how conformance tests can be changed so that the interoperability testing is not necessary.
- Formal descriptions of protocols by using the formal languages such as LOTOS, Estelle, or SDL, should be enhanced to allow well-defined interfaces for the test generation tools [28], [43], [44].
- Testing of complete inopportune and illegal behavior is virtually intractable. Future research should focus on how to choose a good subset of inopportune and illegal behavior for testing a protocol implementation.
- Sound methodologies are needed to automatically generate useful high-level models of protocols for the purpose of generating conformance test sequences.

REFERENCES

- [1] G. v. Bochmann and C. A. Sunshine, "A survey of formal methods," in *Computer Networks and Protocols*, P. E. Green, Ed., New York: Plenum Press, 1983, pp. 561-578.
- [2] B. Sarikaya, Test Design for Computer Network Protocols, Ph.D. thesis, McGill University, Montreal, Canada, 1984.
- [3] H. Ural, "Test sequence selection based on static data flow analysis," *Computer Communications*, vol. 10, no. 5, pp. 234-242, 1987.
- [4] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

- [5] Z. Kohavi, *Switching and Finite Automata Theory*. New York, NY: McGraw-Hill, 1978.
- [6] S. Naito and M. Tsunoyama, "Fault detection for sequential machines by transition tours," *Proc. 11th IEEE Fault Tolerant Comput. Symp.*, IEEE Computer Soc. Press, pp. 238-243, 1981.
- [7] B. Sarikaya and G. v. Bochmann, "Some experience with test sequence generation," *Proc. of Second Int'l. Workshop on Protocol Specification, Testing, and Verification*, C. Sunshine, Ed. Amsterdam, The Netherlands: North Holland, 1982, pp. 555-567.
- [8] M. U. Uyar and A. T. Dahbura, "Optimal test sequence generation for protocols: the Chinese postman algorithm applied to Q.931," *Proc. 1986 IEEE Global Telecommunications Conf.*, 1986, pp. 68-72.
- [9] E. F. Moore, "Gedanken-experiments on sequential machines," *Automata Studies, Annals of Mathematical Studies*, no. 34, Princeton, NJ: Princeton Univ. Press, pp. 129-153, 1956.
- [10] A. Gill, "State-identification experiments in finite automata," *Information and Control*, vol. 4, pp. 132-154, 1961.
- [11] G. Gonenc, "A method for the design of fault detection experiments," *IEEE Trans. on Computers*, vol. 19, no. 7, pp. 551-558, June 1970.
- [12] F. C. Hennie, "Fault-detecting experiments for sequential circuits," *Proc. 5th Ann. Symp. on Switching Circuit Theory and Logical Design*, pp. 95-110, Nov. 1964.
- [13] T. S. Chow, "Testing software designs modeled by finite-state machines," *IEEE Trans. on Software Engineering*, vol. SE-4, no. 3, pp. 178-187, May 1978.
- [14] K. K. Sabnani and A. T. Dahbura, "A new technique for generating protocol tests," *Proc. 9th Data Communications Symp.*, IEEE Computer Soc. Press, pp.36-43, Sept. 1985.
- [15] —, "A protocol testing procedure," *Computer Networks*, vol. 15, no. 4, pp. 285-297, 1988.
- [16] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and Chinese postman tours," *Proc. of 8th. Int'l Symp. on Protocol Specification, Testing, and Verification*, S. Aggarwal and K. Sabnani, Eds. Amsterdam, The Netherlands: North Holland, 1988, pp. 75-86.
- [17] M.-K. Kuan, "Graphic programming using odd or even points," *Chinese Math.*, vol. 1, pp. 273-277, 1962.
- [18] D. Rayner, "OSI conformance testing," *Computer Networks and ISDN Systems*, vol. 14, no. 1, pp. 79-98, 1988.
- [19] ISO/TC97/SC21, "OSI conformance testing methodology and framework," DP 9646, Parts 1-5, July 1988.
- [20] ISO/TC97/SC21, "Information processing systems—open systems interconnection—basic reference model," IS 7498, 1984.
- [21] H. X. Zeng, X. F. Du and C. S. He, "Promoting the 'local' test method with the new concept ferry clip," *Proc. of 8th. Int'l. Symp. on Protocol Specification, Testing, and Verification*, S. Aggarwal and K. Sabnani, Eds. Amsterdam, The Netherlands: North Holland, 1988, pp. 231-241.
- [22] J. Bremer, G. Mondaai, K. Tarnay, and T. Tibor, "Some experience with test sequence generation in application layer," *Proc. of Fourth Int'l. Workshop on Protocol Specification, Testing, and Verification*, Y. Yemini, R. Strom and S. Yemini Eds. Amsterdam, The Netherlands: North Holland, 1984, pp. 623-636.
- [23] D. Rayner, "Towards standardized OSI tests," *Proc. of Fifth Int'l. Workshop on Protocol Specification, Testing, and Verification*, M. Diaz, Ed. Amsterdam, The Netherlands: North Holland, 1985, pp. 441-460.
- [24] R. J. Linn and J. S. Nightingale, "Testing OSI protocols at the NBS," *Proc. IEEE*, vol. 71, no. 12, pp. 1431-1434, Dec. 1983.
- [25] R. J. Linn and J. S. Nightingale, "Some Experience with testing tool for OSI implementations," *Proc. of Third Int'l. Workshop on Protocol Specification, Testing, and Verification*, H. Rudin and C. H. West, Eds. Amsterdam, The Netherlands: North Holland, 1983, pp. 505-520.
- [26] G. W. Cowin, R. W. S. Hale, and D. Rayner, "Protocol product testing—some comparisons and lessons," *Proc. of Third Int'l. Workshop on Protocol Specification, Testing, and Verification*, H. Rudin and C. H. West, Eds. Amsterdam, The Netherlands: North Holland, 1983, pp. 477-491.
- [27] J. R. Pavel and D. J. Dwyer, "Some experiences of testing protocol implementations," *Proc. of Fourth Int'l. Workshop on Protocol Specification, Testing, and Verification*, Y. Yemini, R. Strom, and S. Yemini, Eds. Amsterdam, The Netherlands: North Holland, 1984, pp. 657-677.
- [28] J. P. Favreau and R. J. Linn, "Automatic generation of test scenario skeletons from protocol specifications written in Estelle," *Proc. of Sixth Int'l. Workshop on Protocol Specification, Testing, and Verification*, B. Sarikaya and G. V. Bochmann Eds. Amsterdam, The Netherlands: North Holland, 1986, pp. 191-202.
- [29] H. Ural and R. L. Probert, "User-guided test sequence generation," *Proc. of Third Int'l. Workshop on Protocol Specification, Testing, and Verification*, H. Rudin and C. H. West, Eds. Amsterdam, The Netherlands: North Holland, 1983, pp. 421-436.
- [30] J. P. Ansart, "GENEPI/A, a protocol independent system for testing protocol implementations," *Proc. of Second Int'l. Workshop on Protocol Specification, Testing, and Verification*, C. Sunshine, Ed. Amsterdam, The Netherlands: North Holland, 1982, pp. 523-528.
- [31] B. Sarikaya and G. v. Bochmann, "Synchronization and specification issues in protocol testing," *IEEE Trans. on Commun.*, vol. COM-32, no. 4, pp. 389-395, Apr. 1984.
- [32] ISO, "Estelle: A formal description technique based on an extended state transition model," DIS 9074, July 1987.
- [33] ISO, "LOTOS: A formal description technique based on temporal ordering observational behavior," DIS 8807, July 1987.
- [34] CCITT Recommendations Z.101-Z.104, SDL, 1984.
- [35] M. Barbeau and B. Sarikaya, "A computer-aided design tool for protocol testing," *Proc. IEEE INFOCOM '88*, pp. 86-95, Mar. 1988.
- [36] J. Edmonds and E. L. Johnson, "Matching, Euler tours and the Chinese postman," *Mathematical Programming*, vol. 5, pp. 88-124, 1973.
- [37] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. Assoc. Comput. Mach.*, vol. 24, pp. 1-13, 1977.
- [38] E. P. Hsieh, "Checking experiments for sequential machines," *IEEE Trans. Comput.*, vol. C-20, no. 10, pp. 1152-1166, Oct. 1971.
- [39] B. Wang and D. Hutchinson, "Protocol testing techniques," *Computer Communications*, vol. 10, no. 2, pp. 79-87, Apr. 1987.
- [40] J. K. Lenstra and A. H. G. Rinnooy Kan, "On general routing problems," *Networks*, vol. 6, pp. 273-280, 1976.
- [41] D. Sidhu and T. Leung, "Fault coverage of protocol test methods," *Proc. IEEE INFOCOM '88*, pp. 80-85, Mar. 1988.
- [42] A. T. Dahbura and K. K. Sabnani, "An experience in estimating the fault coverage of a protocol test," *Proc. IEEE INFOCOM '88*, pp. 71-79, Mar. 1988.
- [43] L. Bromstrup and D. Mogrefe, "TESDL: Experience with generating test cases from SDL specifications," in *SDL '89, The Language at Work*, Amsterdam, The Netherlands: North Holland, pp. 267-279, 1989.
- [44] J. Tretmans, "Test case derivation from LOTOS specifications," in *Proc. 2nd Int. Conf. on FDT's for Distributed Systems and Communication Protocols*, pp. 469-487, 1989.



Anton T. Dahbura (Member, IEEE) received the B.S.E.E., M.S.E.E., and Ph.D. degrees in electrical engineering from The Johns Hopkins University in 1981, 1982, and, 1983, respectively.

In 1983 he joined the Digital Systems Research Department of the Computer Technology Research Laboratory, AT&T Bell Laboratories, Murray Hill, NJ, as a Member of Technical Staff. Since 1986 he has been with the Distributed Systems Research Department, also within the Computer Technology Research Laboratory. His interests include fault-tolerant computing, computer architectures, protocol conformance testing, algorithmic graph theory, interconnection theory, and complexity theory.

Dr. Dahbura is a member of the IEEE Computer Society and the ACM.



Krishan Sabnani (Senior Member, IEEE) received a B.S.E.E. degree from Indian Institute of Technology, New Delhi, India, and a Ph.D. degree from Columbia University, New York, NY.

In 1981, he joined AT&T Bell Laboratories after graduating from Columbia University. He is currently working in the Distributed Systems Research Department of AT&T Bell Laboratories. He was awarded the Bell Laboratories Distinguished Technical Staff Award in 1990. His major area of interest is communication protocols. He was a co-chairman of the Eighth International Symposium on Protocol Specification, Testing, and Verification in Atlantic City, NJ, held in June 1988. He is currently an editor of the IEEE TRANSACTIONS ON COMMUNICATIONS and of the IEEE TRANSACTIONS ON COMPUTERS. He has served on the program committees of several

conferences. He has served as a guest editor for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC) and the *Computer Network Journal*.



M. Ümit Uyar (Member, IEEE) received the B.S. Degree from Istanbul Teknik Üniversitesi, Turkey, in 1978, and the M.S. and Ph.D. degrees from Cornell University, Ithaca, New York, in 1981 and 1986, all in electrical engineering.

He is currently employed at AT&T Bell Laboratories, Holmdel, New Jersey, where he is working on formal description techniques for communication protocols. His research interests include fault-tolerance, parallel processing, interconnection networks, and protocol testing and verification.