(Computer Communications: The Int'l Journal for the Comp. and Telecomm. Industry, Special Issue on FDTs in Practice, in press, 1999)

A Success Story of Formal Description Techniques: Estelle Specification and Test Generation for MIL-STD 188-220 ⋆

Mariusz A. Fecko [a] M. Ümit Uyar [b] Paul D. Amer [a]
Adarshpal S. Sethi [a] Ted Dzik [c] Raymond Menell [c]
Mike McMahon [d]

[a] *Computer and Information Sciences Department*
University of Delaware, Newark, DE

[b] *Electrical Engineering Department*
The City College of the City University of New York, NY

[c] *US Army CECOM, Fort Monmouth, NJ*

[d] *ARINC, Inc., Shrewsbury, NJ*

Abstract

This paper presents a success story of specifying a complex real-life protocol (MIL-STD 188-220) in Estelle and generating test sequences from the formal specification. 188-220 is being developed in the US Army, Navy and Marine Corps systems for mobile combat network radios. A key factor in this success story has been the collaboration among the researchers of the University of Delaware and the City College of the City University of New York, the developers of the US Army Communications-Electronics Command (CECOM), and the protocol designers in the Joint Combat Net Radio Working Group. Based on the research results, 188-220 test sequences are realizable without timer interruptions while providing a 200% increase in test coverage. The test cases are being installed at a CECOM test facility.

Key words: conformance testing, test case generation, protocol specification and testing, Estelle, MIL-STD 188-220

⋆ This work was supported by the US Army Research Office (DAAH04-94-G-0093), and prepared through collaborative participation in the Advanced Telecommunications/Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0002.

1 Introduction

Formal methods in conformance testing are becoming widely used in the testing of real-life protocols [7,17,18,25,35,36,74]. OSI conformance testing has been applied to the Internet protocols such as Hypertext Transfer Protocol [25], Simple Mail Transfer Protocol [7], and TCP/IP [36]. Another area of successful application of conformance testing are ATM/B-ISDN protocols, e.g., the testing of ATM Adaptation Layer protocol [74], ATM switching systems [17,35], and B-ISDN signaling [18].

There also have been several reports on successful application of Estelle to real-life systems [9,15,34,48,56,68]. Estelle was used to automatically implement TP4/IP protocol [68], to uncover ISO Remote Operations Service Element [30] protocol errors [34], and to specify, detect, and resolve feature interactions in Intelligent Networks [9]. Most recently, simulation studies [48] of the Estelle specification of Service Specific Connection-Oriented Protocol [33] revealed some specification errors and flow control inefficiences of the protocol definition. Catrina et al. [15] showed that it is possible to use an Estelle specification to automate implementation of a sophisticated transport protocol (XTP 4.0 [73]). Thees [56] compared performance of the XTP 4.0 implementations automatically generated from the Estelle specification with hand-coded implementations. Both studies point out numerous advantages (and potential problems) of using Formal Description Techniques (FDTs) for XTP code generation.

This paper presents a success story of specifying a complex real-life protocol in Estelle, and automatically generating conformance tests from the formal specification. The US Department of Defense (DoD)/Joint protocol, called Military Standard (MIL-STD) 188-220, is being developed in the US Army, Navy and Marine Corps systems for mobile combat network radios [20]. With the understanding of the power of formal description techniques, a key factor in this success story has been the collaboration among the four groups: the researchers of the University of Delaware (UD) and the City College of the City University of New York (CCNY), the developers of the US Army Communications-Electronics Command (CECOM) in New Jersey, and the protocol designers in the Joint Combat Net Radio (CNR) Working Group (WG). As a result of this success story, the synergistic framework to develop C^4I (Command, Control, Communications, Computers, and Intelligence) systems with the help of formal methods serves as a model for future DoD standards development [20].

Since this paper is a case study promoting a successful application of Estelle to a real-life protocol, it includes a cross section of activities over the past few years. Sections 2 provides the background on the collaboration among the MIL-STD 188-220 sponsors, research and development teams, and standards organizations. Sections 3 and 4 briefly overview the formal description technique Estelle and 188-220, respectively. Section 5 presents a part of the Estelle specification of 188-220, and gives examples of errors and ambiguities found thanks to formally specifying the protocol. A general approach adopted at UD and CCNY to test generation from an Estelle formal specification is described in Section 6. This section also summarizes recent research results obtained in minimum-length test generation based on Estelle specifications. Section 7 contains information on practical results in test generation and delivery. Finally, Section 8 presents the authors' view on the improvements to the protocol development process due to using formal methods.

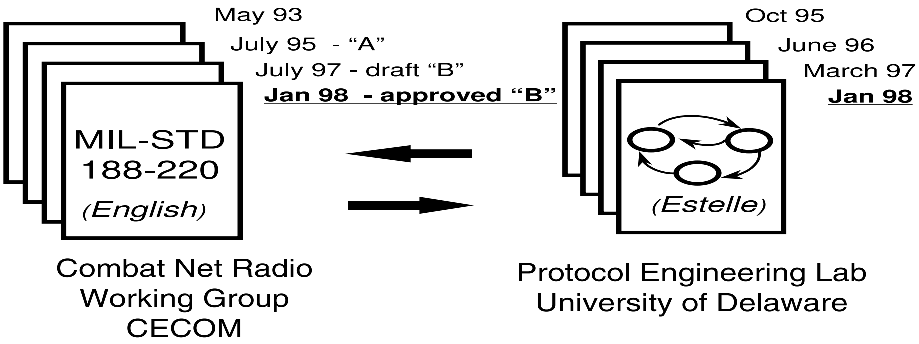

Fig. 1. History of MIL-STD 188-220 Development

2 History of MIL-STD 188-220 Development

In 1994, UD's Protocol Engineering Laboratory began its involvement with the US Army in using Estelle [12,29,55,58] to formally specify a military standard MIL-STD 188-220 [19]. An initial small contract with the Army Research Laboratories supported both simulation and specification of the 1993 version of 188-220 [14,44]. The formal specification research effort received the attention of the CECOM Software Engineering Center, which leads the effort to evolve 188-220 to meet the Army's requirements for battlefield digitization, through the Joint CNR WG, itself responsible for the evolving 188-220 standard.

From 1995 to 1998, at least fifty changes to the English specification of 188-220 resulted from UD's efforts using Estelle to formally specify the standard [2,19] (see Figure 1). While the English text takes precedence in case of disagreement with the formal specification, *the Estelle specification of 188-220 is an official part of the military standard.* To the best of our knowledge, 188-220 [1] represents one of the first major national or international standards officially including an Estelle specification. There are other examples of international standards that include Estelle specifications such as OSI session protocols [47,31], OSI Distributed Transaction Processing [32], and OSI File Transfer, Access and Management protocol [28].

CECOM is developing a Conformance Tester that automatically evaluates a 188-220 implementation identifying where it differs from the standard. The test generation research was initiated as part of the US Army's Advanced Telecommunication and Information Distribution Research Program (ATIRP) in January 1996, when UD's Protocol Engineering Lab began research collaboration with CCNY. Efforts were focused on automatically generating test cases from the Estelle specifications. Generating tests from formal specifications such as pure finite state machines (FSMs) has been extensively studied in the literature. But the inherent complexity of 188-220

[1] In the rest of the paper, 188-220 refers to the version "B" of the standard, completed in January 1998.

is far beyond specifying with pure FSMs, hence the need to use a more powerful specification language such as Estelle, ISO 9074. Unfortunately, generating tests from Estelle specifications presents difficult theoretical and practical problems. UD and CCNY faculty and students continue to investigate these problems with the practical motivation of applying the results towards 188-220 test case generation.

As mentioned above, automatic generation of tests from Estelle specifications presented various theoretical problems:

(1) During testing, if active timers are not taken into account, they can disrupt the test sequences, failing correct implementations (or worse, passing incorrect ones). Therefore, timers have to be incorporated as constraints into the extended FSM (EFSM) model of an Estelle specification.
(2) Test sequence generation is limited by the controllability of an Implementation Under Test (IUT) [8]. Testers may not have direct access to all interface(s) in which the IUT accepts inputs (typically, the interfaces with upper layers, or with timers). In this case, some inputs cannot be directly applied; the interactions involving such interfaces may render some portions of the protocol untestable, and may introduce non-determinism and/or race conditions during testing.
(3) Infeasible test sequences may be generated unless conflicting conditions based on protocol's timers that may be running simultaneously are resolved (the so-called *conflicting timers problem*).

The timing and controllability issues were present in the EFSM model of the Estelle specification of MIL-STD 188-220 [3,21]. Based on the results of investigating problems (1) and (2) by the UD and CCNY joint group [21,23,64], UD has been providing CECOM with automatically generated test sets since 1997. The sizes of the resulting FSMs derived from the Estelle specification range from 48 to 303 states, and from 119 to 925 transitions. The corresponding test sequences range from 145 to 2,803 test steps. These tests are free of interruptions due to unexpected timeouts while their coverage of the number of testable transitions increased from approximately 200 to over 700 by utilizing multiple interfaces without controllability conflicts. Current research focuses on the conflicting timers problem.

3 Estelle

In 1989, Estelle was published as one of two ISO International Standard Formal Description Techniques (FDT) for the specification of computer communication protocols [12,29]. As shown in Figure 2, Estelle specifies a protocol's behavior as a set of communicating extended finite state machines. To avoid ambiguity among different readers of a specification, the Estelle language itself has a formal, mathematical, implementation-independent semantics.

Estelle is an expressive, well-defined, well-structured language that is capable of specifying distributed, concurrent information processing systems in a complete, consistent, concise, and unambiguous manner. An Estelle specification aims at discovering and resolving ambiguities in the

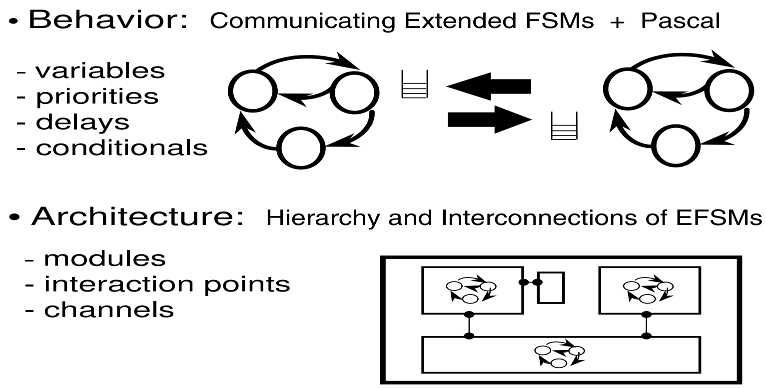

Fig. 2. Estelle: ISO International Standard 9074

original English document that would cause interpretation problems for implementors.

An Estelle specification consists of two parts: an architecture and its behavior. The architecture specifies a collection of systems of nested modules. Each module's behavior is described by an extended FSM. These EFSMs interact via the sending of interactions over a set of channels. The interactions are conceptually stored in infinite FIFO queues enabling transitions in the receiving module which are fired when all enabling conditions are satisfied. A complex set of rules define either a parallel or synchronous firing of transitions within each EFSM. Overall, the many features of Estelle allow a user to formally specify a wide variety of network protocol behaviors. Further information about Estelle can be found in [55,58].

One major benefit of an Estelle specification as a model of a communication protocol is that it can be used as input to a conformance test generation tool. Since Estelle makes it possible to create a complete and unambiguous protocol model, the test cases generated from it can potentially achieve higher fault coverage than hand-generated ones, and are reproducible with far less effort as 188-220 evolves in the future. These advantages are the primary motivations for using Estelle to specify 188-220.

4 MIL-STD 188-220

The Protocol Engineering Lab researchers at UD used Estelle to specify parts of the 188-220 protocol suite. This suite was developed to meet the requirements for horizontal integration, seamless Internet communications and increased mobility using combat network radios [20]. This protocol, a critical piece of the new Joint Technical Architecture, is now mandated for CNR communications. It is being implemented in US Army, Navy and Marine Corps systems, and has been demonstrated initially during the Army's Advanced Warfighting Experiment in 1997. 188-220 is now receiving allied/international attention, while portions of its protocol architecture have been promulgated in the Internet Engineering Task Force. Expected outcomes from its use

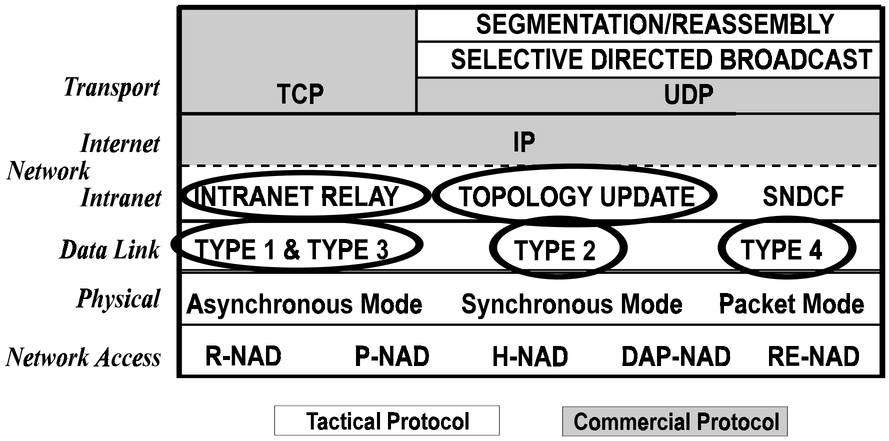

Fig. 3. MIL-STD 188-220 Protocol Architecture. Parts of the protocol where FDTs were used during the development are circled.

are: seamless connectivity of C^4I systems (discussed briefly in Section 8), horizontally integrated information networks, and joint interoperable C^4I systems for the warfighter.

188-220, originally developed in 1993, evolved to 188-220A with substantial new functionality, including support for new radio technology and integration with Internet protocols (commercial IP, TCP, and UDP at the network and transport layers). Version 188-220B, whose architecture is depicted in Figure 3, describes the protocols needed to exchange messages using CNR as the transmission media. These protocols include the physical, data link and part of the network layer of the OSI model. The protocols apply to the interface between host systems and radio systems. Hosts usually include communications processors or modems that implement these lower layer protocols. The unshaded portions of Figure 3 indicate those protocols and extensions that were developed specifically for use with CNR.

MIL-STD-188-220 Datalink layer specifies several service types, each intended to handle different types of traffic with different quality of service (QoS) demands. A 188-220 station can actually process several different types of traffic simultaneously (and almost orthogonally). MIL-STD-188-220 Network Layer consists of Internet (IP) Layer, Subnetwork Dependent Convergence Function (SNDCF), and Intranet Layer. The Intranet Layer has been dedicated to routing intranet packets between a source and possibly multiple destinations within the same radio network. The Intranet Layer also accommodates the rapid exchange of topology and connectivity information—each node on the radio network needs to determine which nodes are on the network and how many hops away they are currently located.

5 188-220 Estelle Specification

To help a reader realize the magnitude of formally specifying a protocol of 188-220 size and complexity, let us give some numbers. The Datalink (Network) layer specification consists of

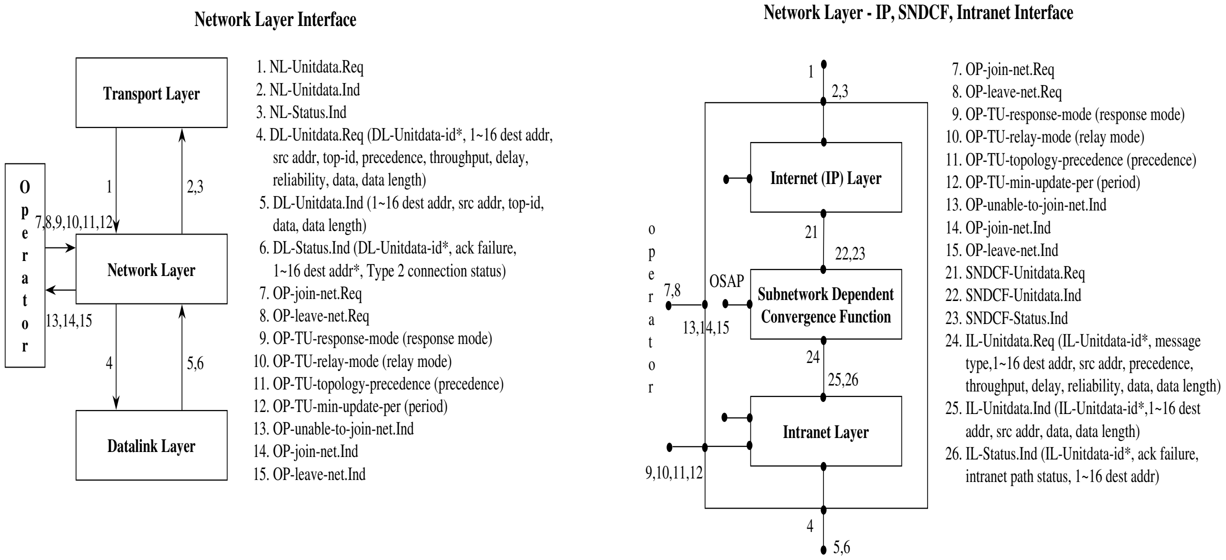

Fig. 4. Network Layer Interface and Architecture

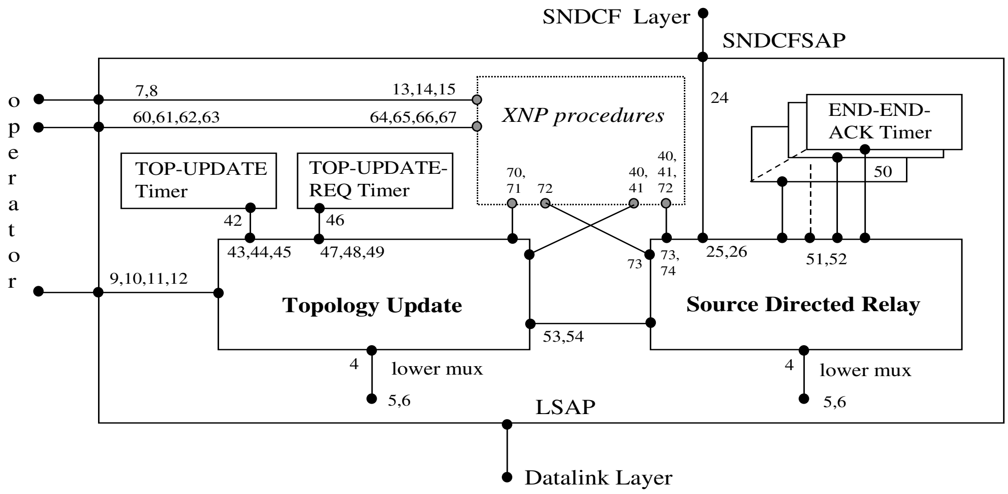

Fig. 5. Intranet Layer Architecture

69 (19) documents describing architecture, interfaces, EFSM, and state table of each module. The Datalink layer specification is accompanied by three (for Datalink classes A, B, and C) Estelle source code files with approximately 1,600, 8,700, and 2,400 lines of code, respectively. The Estelle source code for the Network layer has 7,150 lines of code, defining 34 states and 370 transitions in 7 EFSMs.

Due to its large size, it is not possible to include the actual Estelle specifications in this paper. For a more detailed description of the semantics of Estelle specification components (communication channels, interactions, etc.), the reader may consult our paper on Datalink Layer specification [2], or may visit the Web site at `http://www.cis.udel.edu/~amer/CECOM`. In the next section, we present an overview of the Network Layer architecture with a focus on the Topology Update and Source Directed Relay functions of the Intranet sublayer.

5.1 Intranet Layer Architecture

Figure 4 shows the interface and general architecture of the Network layer. The architecture represents the protocol stack at a single station, as well as an interface with "operator module" which can interact with several different layers in the stack. The operator module abstracts the link layer's interactions with both a human operator and a system management process.[2]

Figure 5 shows the internal structure of the Intranet Layer. The two main Intranet Layer functionalities, Source Directed Relay (SDR) and Topology Update exchange (TU), were encapsulated in separate component modules of the Intranet Layer module. This simplifies the design of the FSMs that model the entire layer, and also allows for generating test cases for each functionality separately.

The SDR module receives *IL_Unitdata_Req* messages through *SNDCFSAP* interaction point. It starts/stops a varying number of *END_END_ACK* timers, one for each IP packet that has been sent but not yet acknowledged. The TU module interacts with the SDR module by notifying it of any topology changes that take place dynamically. The TU module communicates with two timers: *Topology_Update Timer* and *Topology_Update_Request Timer*. The former is started after a topology update message is sent by the station. According to 188-220A, a station is not allowed to send another topology update message until the timer expires. The latter performs the same role for topology update request messages.

Both SDR and TU modules can send and receive messages from the datalink layer through their *lower_mux* interaction points—the messages from the two modules are multiplexed by the parent Intranet Layer module. A peer operator or management component is connected directly to the Topology Update module and can set parameters that are relevant in topology update mechanism. Part of the diagram inside the dash-lined rectangular contains modules that handle XNP procedures: joining and leaving the net with either centralized or distributed control, and parameter update requests.

5.2 Problems and Ambiguities Found in 188-220 through Formal Specification

Primary goals in developing an Estelle formal specification are:

(1) discover and document problems and ambiguities that are commonly seen in a standard written in natural language,
(2) verify the protocol,

[2] Note that the numbers in Figures 4 through 5 refer to interactions, and are consistent throughout the figures (e.g., number 12 refers to *OP-min-update-per* in all three figures).

(3) simulate the protocol,
(4) automate code implementation,
(5) automate test generation process.

MIL-STD 188-220 project focuses on goals (1), (3), and (5), with simulation studies done by the US Army as reported in [20]. Although the formal verification of 188-220 is not part of the project, some of the errors found during the formal specification can also be classified as part of goal (2). Achieving goal (4) is an open issue; manufacturers, which were already developing implementations before the Estelle specification was created, now have an option to use the Estelle specifications for automated code generation.

In the process of developing the Estelle specifications of the Data Link layer and, most recently, the Intranet Layer, more than fifty problems in the original English specification have been documented. Here we present a representative cross-section of examples of ambiguities found and corrected, demonstrating the difficulty of defining protocol operations in a natural language.

Examples range from ambiguities such as:

- "... a station shall wait for some period of time *bounded by the probability* of the remote ack time expiration."
- The Intranet Layer allows a station to enter *Quiet mode* whereas the Data Link layer refers to a station being in *response mode off*. It was unclear how these two terms differ, if at all.

to more serious examples of correctness/completeness such as:

- Intranet routing was originally defined based on spanning trees of the Intranet topology. However the draft standard's examples did not comply with the mathematical definition of a spanning tree.
- The phrase "may report to the higher layer protocol, and may initiate appropriate error recovery action" was added in several locations when the datalink layer identified an error condition such as a lack of acknowledgment after the maximum allowed number of retransmissions.

A few more selected examples are presented in Appendix A to illustrate the nature of errors found in the protocol. All these problems were reported back to the CNR Working Group and subsequently removed from the standard.

6 Test Case Generation

Test scripts (test cases) specify a logical sequence of test steps that are performed by a Conformance Tester to individually test a given protocol entity. The test scripts are input to the Conformance Tester which in turn stimulates an IUT, and assesses the IUT's responses to determine if the IUT correctly implements the protocols. Since it is impossible to exhaustively test an implementation in practice, a good set of test scripts should at least check those events that affect state/transition, boundary conditions, and stress points. The test scripts themselves

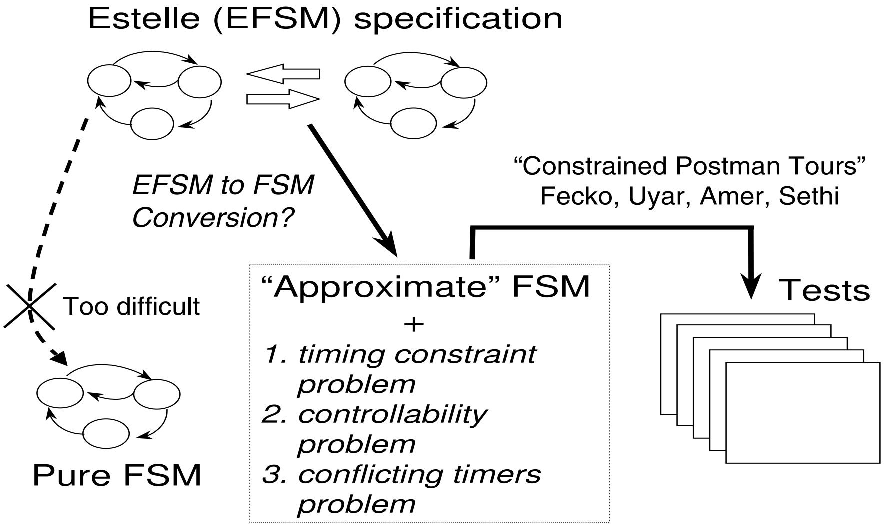

Fig. 6. Test Generation from Extended FSMs

should be structured as independent modular components to facilitate modifying and adding to the scripts in response to 188-220's continuing evolution.

A number of techniques have been proposed to generate test sequences from Estelle specifications [41,42,59,60,72]. However, full Estelle specifications of large systems may prove to be too complex for direct test case generation. As shown in Figure 6, there are several ways of generating test sequences from Estelle specifications. One approach would be to *expand* Estelle's EFSMs thereby converting them to pure FSMs. This would be useful since methods exist for generating tests directly from pure FSMs (e.g., [1]). Unfortunately, converting even simple EFSMs can result in the state explosion problem, that is, the converted FSM may have so many states and/or transitions that either it takes too long to generate tests, or the number of tests generated is too large for practical use.

As an alternative, UD and CCNY joint group uses an intermediate approach, where an Estelle EFSM is partially expanded (hence resulting in some more states and transitions), but not expanded completely to a pure FSM. The EFSM is expanded partially just enough to generate a set of tests that is feasible and practical in size. Determining which features to expand in the general case is the difficult aspect of this research.

Test Case Generation Research:

Conformance test generation techniques reported in literature [1,8,38,45,53,60], using a deterministic finite-state machine (FSM) model of a protocol specification, focus on the optimization of the test sequence length. However, an IUT may have timing constraints imposed by active timers. If these constraints are not considered during test sequence generation, the sequence may not be realizable in a test laboratory. As a result, valid implementations may incorrectly fail the conformance tests (or, nonconformant IUTs may incorrectly pass the tests).

Another problem in test sequence generation is due to the limited controllability of an IUT. Typically, the inputs defined for the interfaces with upper layers or with timers cannot be directly applied by the tester. In this case, the testability of an IUT may severely be reduced; in addition, non-determinism and/or race conditions may occur during testing.

In Sections 6.1 and 6.2, the research results to eliminate the timing constraints and controllability problems which appear in the EFSM model of the 188-220 are outlined. The current research is focusing on the so-called *conflicting timers problem*, where infeasible test sequences may be generated unless conflicting conditions based on timers are resolved, as described in Section 6.3.

6.1 Timing Constraint Problem

During testing, traversing each state transition of an IUT requires a certain amount of time. A test sequence that traverses too many *self-loops* (a *self-loop* is a state transition that starts and ends at the same state) in a given state will not be realizable in a test laboratory if the time to traverse the self-loops exceeds a timer limit as defined by another transition originating in this state. In this case, a timeout will inadvertently trigger forcing the IUT into a different state, and thereby disrupting the test sequence before all of the self-loops are traversed. If this unrealizable test sequence is not avoided during test generation, most IUTs will fail the test even when they meet the specification. Clearly, this is not the goal of testing. Therefore, a properly generated test sequence must take timer constraints into account.

The presented methodology [64,65] optimizes the test sequence length and cost, under the constraint that an IUT can remain only a limited amount of time in some states during testing, before a timer's expiration forces a state change. The solution first augments an original graph representation of the protocol FSM model. Then it formulates a Rural Chinese Postman Problem solution [43] to generate a minimum-length tour. In the final test sequence generated, the number of consecutive self-loops never exceeds any state's specified limit. In most cases, this test sequence will be longer than one without the constraint since limiting the number of self-loop traversals likely requires additional visits to a state which otherwise would have been unnecessary.

The methodology uses UIO sequences for state verification. However, the results presented also are applicable to test generation that uses distinguishing or characterizing sequences. Earlier results of this study, limited to verification sequences that are self-loops, are presented in [64]. The later paper [65] generalizes these earlier results to both self-loop and non-self-loop verification sequences.

6.1.1 Practical Motivation

Examples of protocols that contain many self-loop transitions in their FSM models include ISDN Q.931 for supplementary voice services, MIL-STD 188-220 [19] for Combat Net Radio communication, and LAPD [66], the data link protocol for the ISDN's D channel. For example, in ISDN Q.931 protocol (Basic voice services, for the user side), each state has an average of 9 inopportune transitions, which requires the traversal of 18 self-loop transitions during testing. A Q.931 implementation has several active timers that are running in certain states, e.g., timer *T304* running in state *Overlap sending*, and timer *T310* in state *Outgoing call proceeding*. An EFSM modeling the Topology Update (TU) functionality of the Intranet Layer has three active states in which one or two timers are running [64].

It is not always possible to delay the timeout at a tester's convenience. In real protocols, there may be timers whose timeouts are difficult to set by the tester, e.g., acknowledgment timers' timeout values often are computed by the implementation. Moreover, a tester may want to test an IUT's behavior for different settings of the IUT's internal timers, to be able to test the IUT's correctness for various configurations of the timers.

In addition to the original self-loops of a specification model, additional self-loops are typically created when generated test sequences use state verification techniques such as unique input/output (UIO) sequences [52], distinguishing sequences [6,37], or characterizing sequences [6,37].

6.1.2 Optimizing Tests under Timing Constraints

Let E_{self} and E_{vnsl} be the sets of self-loop and non-self-loop edges to be tested, respectively. Let $d_{self}(v_i)$, the number of self-loops of vertex v_i, be defined as the number of edges in E_{self} incident on v_i. Let $d_{min\_self}(v_i)$ be the minimum number of times any tour covering all edges of $E_{vnsl} \cup E_{self}$ must include vertex $v_i \in V$.

Let $d_{state\_ver}(v_i)$ be the number of self-loop transitions used to verify whether an IUT is in state v_i. Suppose that during testing, a given vertex $v_i \in V$ can tolerate at most $max\_self(v_i)$ self-loops executed at one visit to vertex v_i. Attempting to remain in state v_i to execute $1 + max\_self(v_i)$ self-loops would result in disruption of a test sequence. Testing a self-loop transition involves traversing the self-loop transition followed by applying the state verification self-loop sequence, which contains $d_{state\_ver}(v_i)$ transitions.

Due to space limitations, we are unable to include the detailed derivation of $d_{min\_self}(v_i)$. In [64], we prove that the minimum number of times vertex v_i must be visited in a test sequence is as follows:

$$d_{min\_self}(v_i) = \begin{cases} d_{in}(v_i) \text{ if } d_{self}(v_i) \leq (d_{in}(v_i) * \Delta_1(v_i)) \\ \Gamma(v_i) \quad \text{if } d_{self}(v_i) > (d_{in}(v_i) * \Delta_1(v_i)) \end{cases} \tag{1}$$

where $d_{out}(v_i)$ and $d_{in}(v_i)$ are respectively the out-degree and the in-degree of vertex v_i in E_{vnsl}, and where

$$\Gamma(v_i) = d_{in}(v_i) + \lceil \frac{d_{self}(v_i) - (d_{in}(v_i) * \Delta_1(v_i))}{\Delta_2(v_i)} \rceil \tag{2}$$

$$\Delta_1(v_i) = \lfloor \frac{max\_self(v_i) - d_{state\_ver}(v_i)}{1 + d_{state\_ver}(v_i)} \rfloor \tag{3}$$

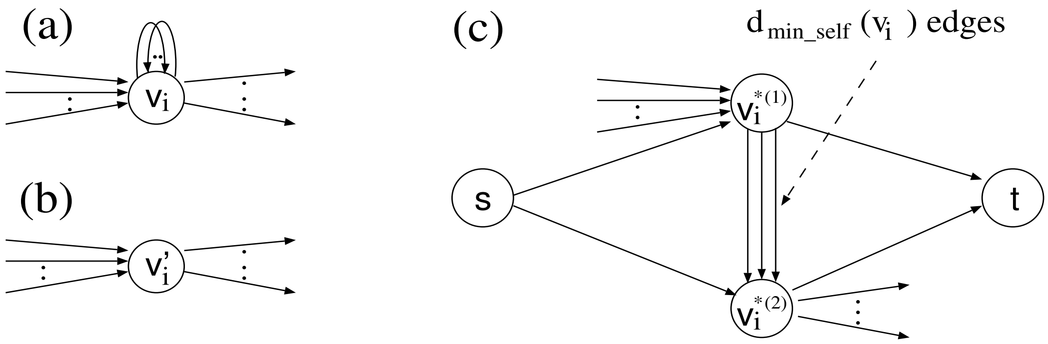

Fig. 7. Conversion of v_i in G (part (a)), to $v_i^{'}$ in $G^{'}$ (part (b)) and to $v_i^{*(1)}, v_i^{*(2)}$ in G^* (part (c)).

$$\Delta_2(v_i) = \lfloor \frac{max\_self(v_i)}{1 + d_{state\_ver}(v_i)} \rfloor \qquad (4)$$

$G^{'}(V^{'}, E^{'})$ ($G^{'}$ is obtained from G by removing self-loop edges) is converted to $G^*(V^*, E^*)$ by splitting each vertex $v_i^{'} \in V^{'}$ satisfying

$$d_{min\_self}(v_i) > max(d_{in}(v_i), d_{out}(v_i)) \qquad (5)$$

into the two vertices $v_i^{*(1)}$, $v_i^{*(2)} \in V^*$ (Figure 7).

Then, $v_i^{*(1)}$ is connected to $v_i^{*(2)}$ with a set of edges with cardinality of $d_{min\_self}(v_i)$: $E_1^* \stackrel{def}{=} \bigcup_{v_i^{'} \in V^{'}} g((v_i^{*(1)}, v_i^{*(2)}), d_{min\_self}(v_i))$. Each edge in E_1^* is assigned infinite capacity β and a zero cost ψ. These fake edges will force additional visits to v_i in a minimum-cost tour of G.

We then use network flow techniques (similar to Aho et al. [1]) to maximize the flow on graph G^* with minimum cost. This flow defines a minimum-cost tour of G under timing constraints.

Example : *Consider the FSM (represented by the graph $G(V, E)$) with self-loop transitions shown in Figure 8. Suppose that vertices v_0, v_2, and v_3 of the FSM can tolerate at most three, and v_1 at most two self-loop transitions during each visit. Let transitions $e10$ and $e11$ correspond to timeouts. After either $e10$ or $e11$ is triggered, the FSM is brought into state v_3.*

UIO sequences and the values of max_self, d_{state_ver} and d_{min_self} for vertices v_0, v_1, v_2, and v_3 are as follows:

Vertex	UIO	max_self	d_{state_ver}	d_{min_self}
v_0	*e0*	*3*	*1*	*2*
v_1	*e2*	*2*	*1*	*3*
v_2	*e6,e7*	*3*	*2*	*4*
v_3	*e9*	*3*	*1*	*2*

The Chinese postman method [61] when applied to the graph without any self-loop repetition constraint

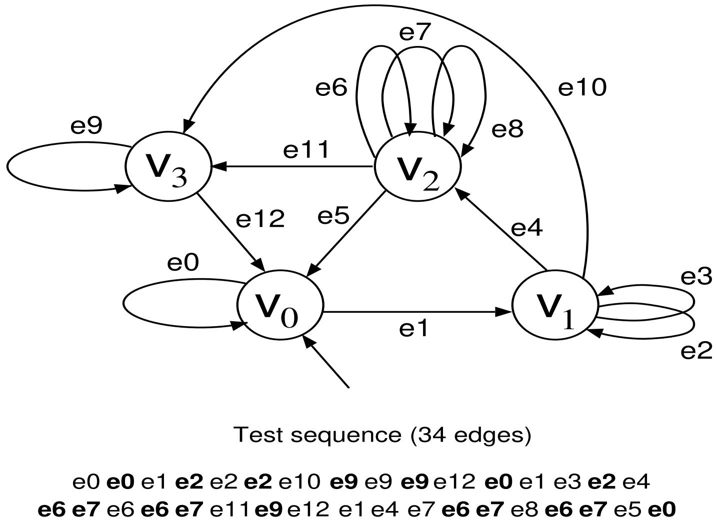

Fig. 8. Minimum-cost test sequence without self-loop repetition constraint.

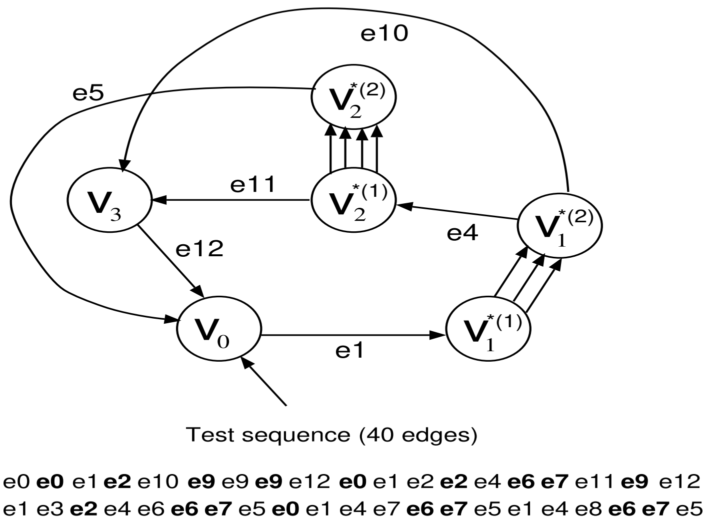

Fig. 9. Minimum-cost test sequence with self-loop repetition constraint.

results in the test sequence

$$\begin{aligned}&\underline{e0, \mathbf{e0}, e1, \mathbf{e2}, e2, \mathbf{e2}, e10}, \mathbf{e9}, e9, \mathbf{e9}, e12, \mathbf{e0}, e1, e3, \mathbf{e2}, e4, \mathbf{e6}, \mathbf{e7},\\ &\quad e6, \mathbf{e6}, \mathbf{e7}, e11, \mathbf{e9}, e12, e1, e4, e7, \mathbf{e6}, \mathbf{e7}, e8, \mathbf{e6}, \mathbf{e7}, e5, \mathbf{e0}\end{aligned} \tag{6}$$

containing 34 edges. Edges used for the purpose of state verification appear in bold.

As can be seen from the underlined part of the above test sequence, after $e1$ is traversed, the IUT should stay in state v_1 for a time that allows at least three self-loop traversals. However, this part of the test sequence is not realizable in a test laboratory because the timeout edge $e10$ will be triggered after the second consecutive self-loop traversal (i.e., $max\_self(v_1) = 2$). The IUT will prematurely move into v_3 and the test sequence will be disrupted.

To address the problem of test sequence disruption due to timeouts, the graph of Figure 8 is converted

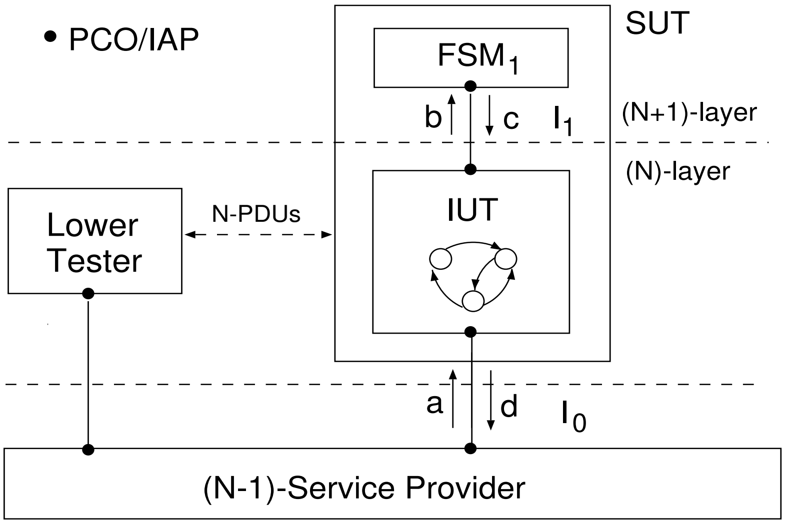

Fig. 10. Testing (N)-layer IUT with an (N+1)-layer semicontrollable interface.

to the graph shown in Figure 9. Since in this example all UIO sequences are self-loops, the simplified conversion presented in [64] is sufficient. The vertices for which a premature timeout may disrupt a test sequence, which are v_1 and v_2, are split and then connected by $d_{min\_self}(v_1) = 3$ and $d_{min\_self}(v_2) = 4$ edges, respectively.

Considering the constrained self-loop problem, the test sequence for the graph of Figure 9 is obtained as

$$\begin{aligned} &e0, \mathbf{e0}, e1, \mathbf{e2}, e10, \mathbf{e9}, e9, \mathbf{e9}, e12, \mathbf{e0}, e1, e2, \mathbf{e2}, e4, \mathbf{e6}, \mathbf{e7}, e11, \mathbf{e9}, e12, e1, e3, \\ &\quad \mathbf{e2}, e4, e6, \mathbf{e6}, \mathbf{e7}, e5, \mathbf{e0}, e1, e4, e7, \mathbf{e6}, \mathbf{e7}, e5, e1, e4, e8, \mathbf{e6}, \mathbf{e7}, e5 \end{aligned} \tag{7}$$

containing 40 *edges.*

Although longer than that of Figure 8, the test sequence in Figure 9 is minimum-length with the introduced self-loop constraint. During each visit to vertices v_0, v_1, v_2 and v_3, the number of consecutive self-loop edges traversed is less than or equal to the maximum allowed number of self-loop traversals. Therefore, this test sequence is realizable in the test laboratory.

6.2 Controllability Problem

Consider a testing framework where the interface I_1 between the IUT and the (N)-layer in the System Under Test (SUT) [8] is not externally accessible (Figure 10). In other words, the inputs from (N+1)-layer cannot be directly applied to the IUT, nor can the outputs generated by the IUT be observed at (N+1)-layer. Such an interface I_1 is called *semicontrollable* if FSM_1 can be utilized to supply inputs to the IUT. On the other hand, the tester can apply inputs to the IUT directly by using a lower tester, which exchanges N-PDUs with the IUT by using the (N-1)-Service Provider. The interface I_0 between the lower tester and the IUT is therefore *directly controllable.*

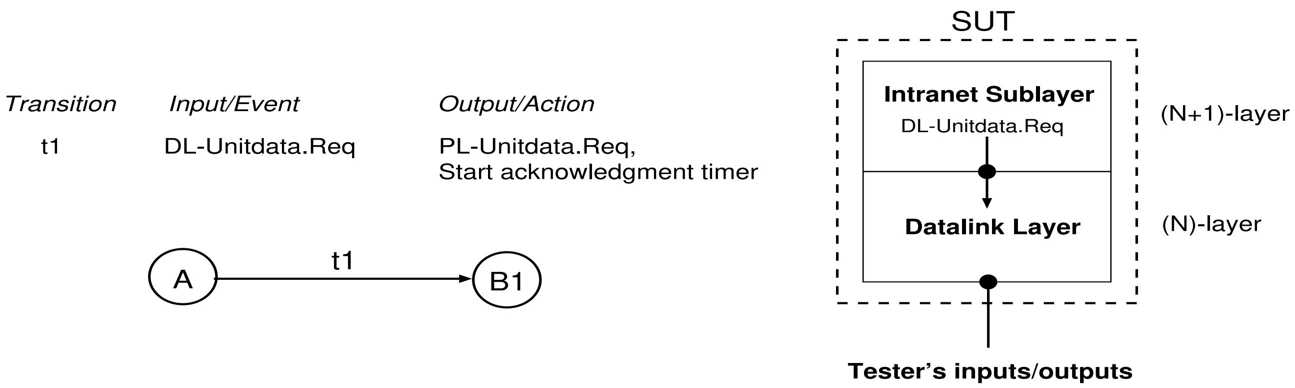

Fig. 11. MIL-STD 188-220: Example of the controllability problem

The methodology presented in [23] addresses the problem of generating optimal realizable test sequences in an environment with multiple semicontrollable interfaces. The methodology fully utilizes semicontrollable interfaces in an IUT while avoiding the race conditions. An algorithm is introduced in [23] to modify the directed graph representation of the IUT such that its semicontrollable portions become directly controllable, where possible. In the most general case, obtaining such a graph conversion may end up with exponentially large number of nodes. However, it is shown [23] that special considerations such as the small number of interfaces interacting with an IUT and diagnostics considerations make the problem size feasible for most practical cases.

6.2.1 Practical Motivation

As motivation for solving the controllability problem, a real protocol is considered where an SUT's (N+1)-layer must be utilized indirectly to test certain transitions within the (N)-layer IUT.

188-220 focuses on 3 layers: Physical, Datalink, and Network. The Network layer contains an Intranet sublayer. An SUT contains the (N)-layer IUT implemented in the Datalink layer, and the Intranet sublayer, which is part of the (N+1)-layer, as shown in Figure 11.

In the CECOM's environment used for testing 188-220 implementations, the upper layers cannot be directly controlled. Therefore, the IUT's transitions that are triggered by the inputs coming from the Network layer are not directly testable. An example SUT transition that causes a controllability problem is the transition $t1$ from the Class A–Type 1 Service Datalink module [19,21], shown in Figure 11. The *input/event* field for this transition requires a *DL_Unitdata_Req* from the (N+1)-layer. Unfortunately, the interface between the IUT and the (N+1)-layer is not directly accessible for generating this input. Initially, it appears that transition $t1$ is untestable.

To trigger this transition, which requires the (N+1)-layer to pass a *DL-Unitdata.Req* down to the (N)-layer, feedback from the (N+1)-layer must be used. To force a *DL-Unitdata.Req* from the (N+1)-layer, the tester sends a *PL-Unitdata.Ind* to the IUT (similar to the message a in Figure 10) that contains an intranet layer message telling the (N+1)-layer to relay the frame to a different network node. The IUT outputs this message to the (N+1)-layer (see message b in Figure 10), and the (N+1)-layer FSM responds by outputting the desired *DL-Unitdata.Req* (message c in Figure 10). Finally, the datalink layer generates the desired output *PL-Unitdata.Req* (corresponding to message d in Figure 10), which can be observed by the lower tester.

In fact, 70% of the transitions the Class A–Type 1 Datalink Service module are based on not directly controllable inputs. Without indirect testing, test coverage would be seriously limited; only approximately 200 transitions out of 750 would be testable. However, by applying the technique outlined in this paper, over 700 of defined transitions (>95%) can be tested. The application of the presented technique to 188-220 is described in more detail in [22].

Similar controllability problems can also be pointed out in testing the IEEE 802.2 LLC Connection Component [23,4].

6.2.2 *Optimizing Tests with Multiple Semicontrollable Interfaces*

To optimize tests with multiple semicontrollable interfaces, modeling SUT as a single FSM was proposed [23,24]. A semicontrollable interface I_i is implemented as a separate FIFO buffer. During testing, a buffer may be empty or store an arbitrary sequence of inputs to the IUT generated indirectly through I_i. For each I_i, we define variable ω_i that has a distinct value for each permutation of inputs that the i-th buffer can hold. The proposed model consists of graph G (which represents the IUT's FSM) and the variables $\omega_1, \omega_2, \ldots, \omega_F$.

An FSM modeling the SUT can be obtained by expanding G and $\omega_1, \omega_2, \ldots, \omega_F$ into $G^{'}(V^{'}, E^{'})$. An algorithm for converting $G(V, E)$ to $G^{'}(V^{'}, E^{'})$ proceeds as follows (a detailed description of the algorithm along with its pseudocode is available in [23,24]):

Step 0—Definitions:
Let B_i denote a sequence of inputs buffered at the i-th semicontrollable interface. Each state $v^{'} \in V^{'}$ has two components: the original state $v \in V$, and the current configuration of F buffers, i.e., $v^{'} = (v, \hat{B}_1, \ldots, \hat{B}_F)$. The algorithm constructs all possible buffer configurations with up to b_i inputs buffered at I_i.

Step 1—Initialize:
$r^{'}$, root of $G^{'}$, as $(r, \emptyset, \cdots, \emptyset)$ (root of G and configuration of empty buffers); $E^{'}$ as empty set; $V^{'}$ as $\{r^{'}\}$; Q, queue of vertices, as $V^{'}$

Step 2—Repeat until Q is empty:

(1) extract $v^{'} = (v_{start}, \hat{B}_1, \ldots, \hat{B}_F)$ as first element from Q, where $(\hat{B}_1, \ldots, \hat{B}_F)$ is current configuration

(2) given the current vertex $v^{'} = (v_{start}, \hat{B}_1, \ldots, \hat{B}_F)$, perform the following steps for each original outgoing edge $e = (v_{start}, v_{end}) \in E$:
- create new configuration $(B_1, \ldots, B_F)$ based on the class of e (Figure 12):
  - **Class 1:** e is triggered by an input from and generates output(s) to an LT;
  - **Class 2:** e is triggered by an input from an LT and generates an output $o_{q,l}$ (buffered in B_q to create a new configuration) at I_q;
  - **Class 3:** e is triggered by $a_{p,k}$ (extracted from B_p to create a new configuration) from I_p and generates output(s) to an LT;
  - **Class 4:** e is triggered by an input $a_{p,k}$ from I_p and generates an output $o_{q,l}$ at I_q. Apply rules for *Class 3* and *Class 2* to create a new configuration.
- create new vertex $v^{'}_{new} = (v_{end}, B_1, \ldots, B_F) \in V^{'}$, and new edge $e^{'}_{new} = (v^{'}, v^{'}_{new}) \in E^{'}$

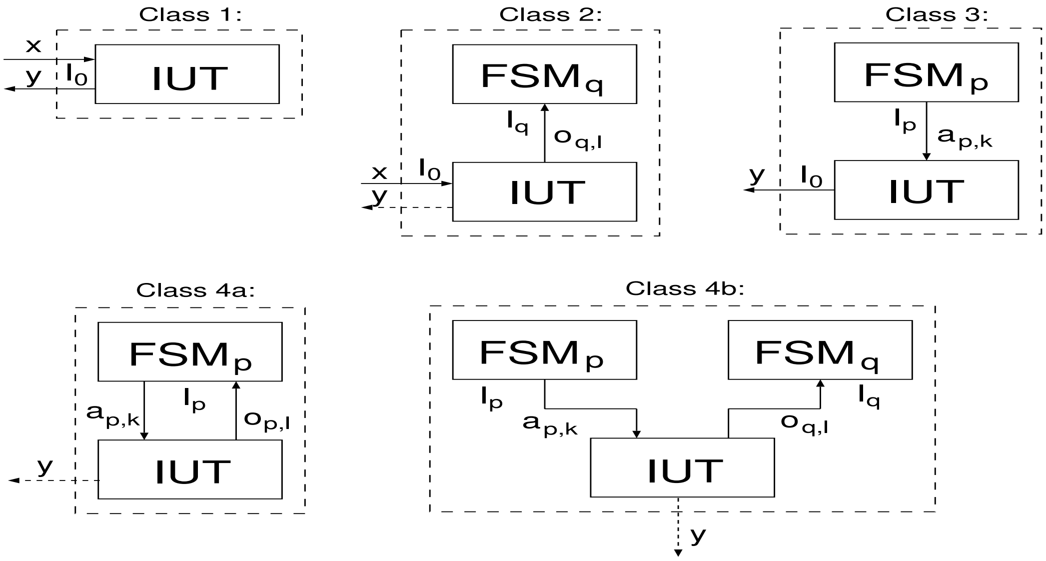

Fig. 12. Classes of edges in $G^{'}$ (dashed-lined outputs are optional).

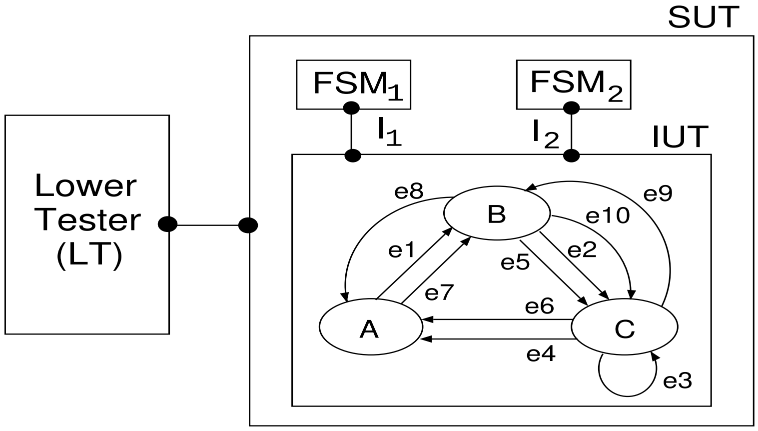

Fig. 13. IUT interacting with two semicontrollable interfaces.

- include new edges in $E^{'}$ iff inputs in $(\hat{B}_1, \ldots, \hat{B}_F)$ cannot trigger other edges outgoing from v_{start}
- append to Q end vertices $v^{'}_{new} \in V^{'}$ of new edges included in $E^{'}$

Step 3—Retain only strongly connected states:
remove from $V^{'}$ all vertices from which $r^{'}$ cannot be reached, and remove from $E^{'}$ all edges incident to such vertices

Based on the practical considerations discussed in [23], the algorithm can be refined to meet the following objective: *"generate a test sequence that, at any point in time, avoids storing more than one input in only one of the buffers (where possible)."* Satisfying this objective yields a linear running time in the number of semicontrollable interfaces and the number of edges in G. If this objective cannot be satisfied, the running time grows and nondeterminism may not be avoided during testing.

Example : *Consider the IUT of Figure 13 which is interacting with semicontrollable FSM_1 and FSM_2 through the semicontrollable interfaces I_1 and I_2, respectively. The IUT's FSM (represented by graph*

Table 1
Inputs and outputs for the edges of Figure 13. $A?x$ denotes receiving input x from A. $B!y$ denotes sending output y to B.

Edge	*Input*	*Output*	*Edge*	*Input*	*Output*
e1	$LT?x_1$	$FSM_1!o_{1,1}$	e6	$LT?x_6$	$LT!y_6$
e2	$LT?x_2$	$FSM_2!o_{2,1}$	e7	$LT?x_7$	$LT!y_7$
e3	$FSM_1?a_{1,1}$	$LT!y_3$	e8	$FSM_1?a_{1,2}$	$LT!y_8$
e4	$FSM_2?a_{2,1}$	$FSM_1!o_{1,2}$	e9	$LT?x_9$	$LT!y_9$
e5	$LT?x_5$	$FSM_2!o_{2,2}$	e10	$LT?x_{10}$	$LT!y_{10}$

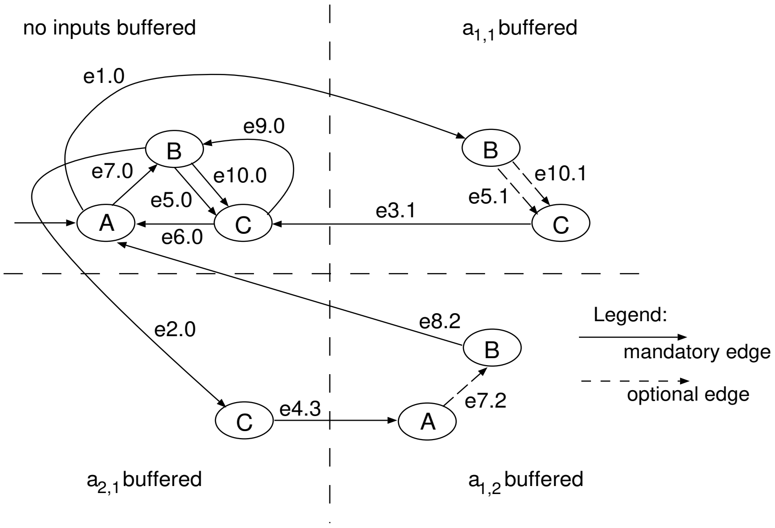

Fig. 14. Graph transformation applied to the graph of Fig. 13. Mandatory and optional edges appear in solid and dashed lines, respectively.

Table 2
Minimum-length test sequence for the IUT of Figure 13.

Step	*Edge*	*Input*	*Output*	*Step*	*Edge*	*Input*	*Output*
$\rightarrow$ 1	e1.0	$LT?x_1$	$FSM_1!o_{1,1}$	8	e7.2	$LT?x_7$	$LT!y_7$
2	e5.1	$LT?x_5$	$FSM_2!o_{2,2}$	$\rightarrow$ 9	e8.2	$FSM_1?a_{1,2}$	$LT!y_8$
$\rightarrow$ 3	e3.1	$FSM_1?a_{1,1}$	$LT!y_3$	10	e7.0	$LT?x_7$	$LT!y_7$
$\rightarrow$ 4	e6.0	$LT?x_6$	$LT!y_6$	$\rightarrow$ 11	e5.0	$LT?x_5$	$FSM_2!o_{2,2}$
$\rightarrow$ 5	e7.0	$LT?x_7$	$LT!y_7$	$\rightarrow$ 12	e9.0	$LT?x_9$	$LT!y_9$
$\rightarrow$ 6	e2.0	$LT?x_2$	$FSM_2!o_{2,1}$	13	e10.0	$LT?x_{10}$	$LT!y_{10}$
$\rightarrow$ 7	e4.3	$FSM_2?a_{2,1}$	$FSM_1!o_{1,2}$	14	e6.0	$LT?x_6$	$LT!y_6$

G) is described in Table 1. Transition e1, triggered by input x_1 from the lower tester, generates output $o_{1,1}$ to FSM_1. In response, FSM_1 sends input $a_{1,1}$ which triggers transition e3. (In general, $a_{i,j}$ is the expected response to $o_{i,j}$.) Transition e2, which is triggered by a lower tester's input x_2, outputs $o_{2,1}$ to FSM_2, which responds with input $a_{2,1}$ triggering e4. Then e4 outputs $o_{1,2}$ to FSM_1, which responds with $a_{1,2}$ triggering e8. On the other hand, transitions e5, e6, e7, e9, and e10, can be triggered directly by the lower tester. e6, e7, e9, and e10, do not generate outputs to the semicontrollable interfaces. e5 generates output $o_{2,2}$ to FSM_2, which does not send any input to the IUT.

After conversion (Figure 14), each state of G is replaced with at most four related states in G' corresponding to the buffer configurations at a semicontrollable interface. Each edge e is annotated as e.x, where $x = 0, 1, 2, 3$, depending on the input buffered in the e.x's start state, as shown in Figure 14. The solid edges in Figure 14 are the mandatory edges that are incident to nodes that correspond to the case where both buffers are empty; the dashed-line edges are the ones that can be traversed only when either buffer contains an input. Due to the practical diagnostic considerations [23], we prefer testing edges when no inputs are buffered in semicontrollable interfaces. The Aho et al. [1] optimization technique gives the minimum-length test sequence for G' shown in Table 2. Steps with ($\rightarrow$) indicate that an edge is tested in this step. Note that, for simplicity, the UIO sequences [52] are not included in this sequence.

6.3 Conflicting Timers Problem

Suppose that a protocol specification defines a number of timers, $c_1, \ldots, c_k$, such that a timer c_i may be started and stopped in transitions defined in states v_i and u_i, respectively. Each timer c_i can be associated with a boolean variable T_i whose value is $TRUE$ if c_i is running, and $FALSE$ if c_i is not running. Let ϕ be a formula obtained from variables $T_1, \ldots, T_k$ by using logical connectives $\wedge$, $\vee$, and $\neg$. Suppose that a specification contains transitions with conditions of a form "if ϕ" for some formula ϕ. It is clear that there may exist infeasible paths in the FSM modeling a protocol, if two or more edges in a path have inconsistent conditions. For example, for transitions *e1: if (T_i) then* $\{\}$ and *e2: if ($\neg T_i$) then* $\{\}$, a path *e1,e2* is inconsistent unless *e1* sets T_i to $FALSE$ (which happens when timer c_i expires in transition *e1*). The problem of generating tests free of such conflicts is called the *conflicting timers problem.*

Note that the conflicting timers problem is a special case of the general feasibility problem of test sequences. There are two distinctive features of the conflicting timers problem: (1) time variables are linear, and (2) time variable values implicitly increase as time elapses. By considering these two features, we expect to solve this case more efficiently than we could solve the general feasibility problem; hence, it is beneficial to formulate this problem separately.

188-220 Datalink Layer defines several timers that can run concurrently and affect behavior of the protocol. For example, $BUSY$ and ACK timers may be running independently in *FRAME_BUFFERED* state. If either timer is running, a buffered frame cannot be transmitted. If ACK timer expires while $BUSY$ timer is <u>not</u> running, a buffered frame is retransmitted. If, however, ACK timer expires while $BUSY$ timer is running, no output is generated.

In the test cases delivered to CECOM, such conflicts are handled by manually expanding EFSMs based on the number of conflicting timers. An efficient solution to the conflicting timers problem

Table 3
188-220 Datalink tests. A single step corresponds to one input/output exchange.

Test set	*# of states*	*# of transitions*	*# of test steps*
	Class A Type 1 service		
general behavior	298	799	1732
precedence	303	401	1316
multidestination	112	119	145
	Class C Type 1 service		
general behavior	298	799	1732
precedence	193	357	1314
multidestination	112	119	145
	Class C Type 4 service		
general behavior	235	925	2803
outstanding frames	48	172	264
multidestination	112	119	145

that eliminates the redundancies of manual state expansion is expected to significantly shorten the test sequences while providing the same fault coverage.

Similar inconsistencies, but based on arbitrary linear variables, are present in EFSMs modeling VHDL specifications [62]. Uyar and Duale presented algorithms for detecting [62] and removing [63] inconsistencies in VHDL specifications. Current research in UD and CCNY is focused on adapting these algorithms to detecting and removing inconsistencies caused by a protocol's conflicting timers.

7 Results: Generation and Delivery of Test Scripts

Using the initial results, UD and CCNY collaboratively generated tests for the SAP components of 188-220's Data Link Layer Class A. Class A stations implement Type 1 (Unacknowledged and Coupled Acknowledged Datalink Connectionless) Service, with the original EFSM consisting of 1 state and 15 transitions. Based on the Class A SAP functionalities, the original EFSM was divided into three EFSMs modeling: (1) general behavior of the SAP component interacting with two destinations, (2) datalink precedence, and (3) an IUT's behavior when interacting with up to 16 destinations. Since the total number of states/transitions that would be obtained after full expansion to a pure FSM was infeasibly large, each of the three EFSMs was expanded to a form closer to a pure FSM, but still containing some extensions.

To avoid state explosion problem, each expanded EFSM focused on certain protocol functionalities while restricting others. For example, in 188-220, a sender can interact with up to 16 destinations, each of which may be free or busy. In general behavior tests, destinations are allowed transit between free or busy mode, but the sender is restricted to communicate with at most two of them. In multidestination tests, the sender communicates with up to 16 destinations, which are forced to remain in free mode at all times.

Each expanded EFSM was then used in automated test generation. Table 3 shows the sizes of the expanded EFSMs and the tests that were generated from them. For example, the precedence tests set for Class A–Type 1 Service was based on an expanded EFSM of 303 states and 401 transitions. The minimum-length test sequence generated for this machine consists of 1,316 input/output pairs covering every transition in the expanded EFSM at least once.

In 1997, these Class A tests were delivered to CECOM for use in its 188-220 testing facility. Figure 15 shows a sample of the delivered test scripts. The figure depicts the test group #92 from Datalink Class A–Type 1 service tests. Each test group is a subsequence of a full test sequence that starts and ends in the initial state. In the first step, the technique of utilizing semicontrollable interfaces presented in Section 6.2 is used. The lower tester sends a packet with three destination addresses: *IUT_addr*, *des_addr_1*, and *des_addr_2*. The setting *Relay=Yes* in the *INTRANET* clause tells the first addressee, i.e., the IUT, to relay the packet to the two remaining addressees. As a result, the IUT sends a packet with its address as a source, and *des_addr_1* and *des_addr_2* as destinations, as if it were originated by the IUT's Intranet Layer. In the second and third steps, the IUT's packet sent in the first step is acknowledged by *des_addr_2* and *des_addr_1*, respectively. Each test step is further annotated with the test description, the number of the corresponding Estelle transition(s), and the appropriate section(s) from the 188-220 official document.

Since the summer of 1997, the work on test generation expanded to include Class C. Class C also allows Type 1 Service as in Class A, but it additionally defines Type 4 (Decoupled Acknowledged Connectionless) Service. As in the case of Class A, three EFSMs were used to generate three sets of tests for each Class C service. The sizes of the EFSMs and the corresponding minimum-length tests are shown in Table 3. For example, the general behavior tests set for Class C Type 4 Service was based on an EFSM of 235 states and 925 transitions. The minimum-length test sequence generated for this machine consists of 2,803 input/output pairs. These tests were delivered to CECOM in 1998.

At the time this paper was written, the implementations from three major manufacturers were being tested at CECOM. The tests generated by the UD and CCNY team have uncovered several implementation errors, including lack of mandatory capabilities in Datalink layer, and problems with multi-hop Intranet Relaying.

```
//                Test Group #92
// -------------------------------------------------
    TESTGROUP=92;
    LAYER=Datalink;

// Test 1
    STIMULUS=send; // PL-Unitdata.Ind
    TIME=long;
    // DL1

    INTRANET={
      Type=IP;
      LowDelay=Yes;
      HighThroughput=No;
      HighReliability=No;
      Precedence=1; // PRIORITY
      OrgAddr=des_addr_17;
      DestRelay={
        Addr=IUT_addr;
        Distance=1;
        Des=No;
        Relay=Yes;
        Ack=No;
      };
      DestRelay={
        Addr=des_addr_1;
        Distance=2;
        Des=Yes;
        Relay=No;
        Ack=No;
      };
      DestRelay={
        Addr=des_addr_2;
        Distance=2;
        Des=Yes;
        Relay=No;
        Ack=No;
      };
    };
    DATALINK={
      CtrlField={
        SendSeq=1;
        RecSeq=1;
        ControlSpare=1;
        DLPrec=1; // PRIORITY
        IDNum=1;
        PDU=ui_0;
      };
      Command=Yes;
      SrcAddr=des_addr_17;
      DestAddr=IUT_addr;
    };

    RESULTS=receive; // PL-Unitdata.Req
    TIME=normal;
    // DL1
    DATALINK={
      CtrlField={
        SendSeq=1;
        RecSeq=1;
        ControlSpare=1;
        DLPrec=1; // PRIORITY
        IDNum=1;
        PDU=ui_1;
      };
      Command=Yes;
      SrcAddr=IUT_addr;
      DestAddr=des_addr_1;des_addr_2;
    };
  TESTDESCRIPTION={
    Intranet layer passes down a multidestination packet
    which is queued by datalink layer. Packet requires
    a coupled ack. There are no outstanding frames.
    No outstanding frame. Queued frame transmitted to multiple
    destinations. Frame requires a coupled ack. Ack timer
    started.
    };
  // ESTELLE     TYPE1SAP_3,4,TYPE1SAP_18
  // SECTION(S)  5.3.16_5.3.6.1.1_C4.3,5.3.4.2.2.2.1_5.3.6.1.1

// Test 2
    STIMULUS=send; // PL-Unitdata.Ind
    TIME=normal;
    // DL1
    DATALINK={
      CtrlField={
        SendSeq=1;
        RecSeq=1;
        ControlSpare=1;
        DLPrec=2; // ROUTINE
        IDNum=1;
        PDU=urr_0;
      };
      Command=No;
      SrcAddr=des_addr_2;
      DestAddr=IUT_addr;
    };
    RESULTS=noop; // none

    TESTDESCRIPTION={
      Second destination acks a multidestination packet.
      First has not acked yet.
     };
    // ESTELLE     TYPE1SAP_12
    // SECTION(S)  5.3.7.1.5.5_5.3.6.1.6_C4.3

// Test 3
    STIMULUS=send; // PL-Unitdata.Ind
    TIME=normal;
    // DL1
    DATALINK={
      CtrlField={
        SendSeq=1;
        RecSeq=1;
        ControlSpare=1;
        DLPrec=2; // ROUTINE
        IDNum=1;
        PDU=urr_0;
      };
      Command=No;
      SrcAddr=des_addr_1;
      DestAddr=IUT_addr;
    };
    RESULTS=noop; // none

    TESTDESCRIPTION={
      First destination acks a packet. Ack timer is stopped.
      No frame queued for transmission.
      };
    // ESTELLE     TYPE1SAP_12
    // SECTION(S)  5.3.7.1.5.5_5.3.6.1.6_C4.3
```

Fig. 15. A sample of test scripts delivered to CECOM.

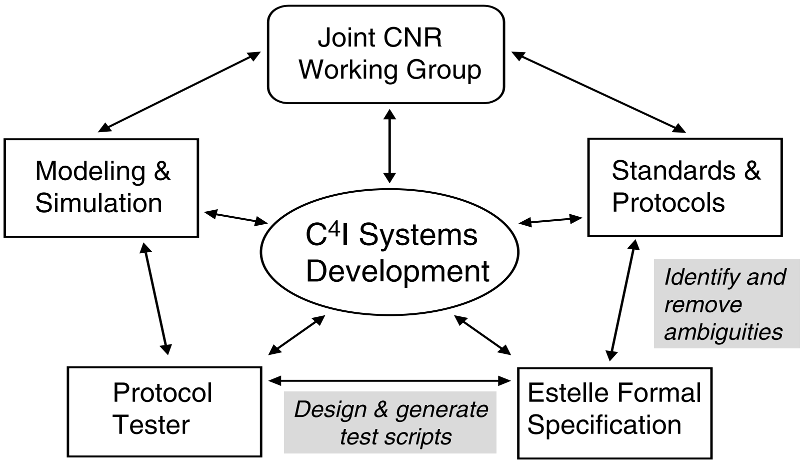

Fig. 16. Estelle as part of synergistic efforts to develop C^4I systems.

8 Conclusions: Improvements to Protocol Development Process

8.1 Integration of Estelle into System Development

Traditional sequential process of system development is known to be inefficient since it allows unnecessary duplication and does not facilitate tracking of rapidly changing technology. With 188-220 as a critical component, a synergistic framework for C^4I (Command, Control, Communications, Computers, and Intelligence) systems development has been established [20] (Figure 16). It combined several parallel activities: developing protocol standards and specifications, formally specifying protocols in Estelle, building conformance tester hardware and software, "field testing", modeling and simulation, as well as resolving and documenting the solutions to standards-related technical issues by the Joint CNR Working Group. (WG participants include representatives from DoD services/agencies, industry, and academia.)

Using formal methods as part of this process helped create a high quality protocol standard, which is robust, efficient, and relatively error-free. Also, due to the structured nature of Estelle, the specification process progressed at an accelerated pace compared to the English specifications (188-220 was completed on time, setting a rare example in protocol standards arena).

Since it is relatively easier to extract modeling information from a formal specification, the researchers at UD and CCNY were able to solve a number of theoretical problems, which resulted in the development of new testing methodologies. By applying these new results, the conformance tests for 188-220 were generated while the protocol was still evolving. Performing initial conformance tests on prototypes uncovered several interoperability errors very early in the development process.

Following this success of the 188-220 development, the synergistic efforts to develop C^4I systems with the help of formal methods serves as a model for DoD standards process and development for the future [20].

8.2 Advantages of Formal Methods in Eliminating Protocol Errors

The difficulties of describing protocol operations with clarity, precision, and consistency by using a natural language are illustrated by the examples in Section 5.2. In addition to the vagueness introduced by a natural language description, ambiguities and contradictions are difficult to detect when related protocol functionalities are defined in different document sections separated by pages of unrelated text. Such problems are eliminated in a formal Estelle specification. All actions in a particular context are defined in one place within the Estelle specification. The specifications make the conditions for state transitions explicit through Estelle constructs. Indeed, the very process of creating these constructs enables formal specifiers to detect some of these types of ambiguities which are difficult to see in normal reading.

8.3 Observations on Applicability of Formal Methods

As concluding remarks for this paper, we report the following observations based on our experience during the formal specification and test generation for 188-200.

To develop an Estelle formal specification of a protocol, we must not only define its architecture and interface components (e.g., as in Figures 4 and 5 for 188-220), but we must also carefully specify the behavior of each module of these components. This definition, achieved through the creation of EFSMs, is the most difficult and time-consuming step of creating a formal specification. A syntax-directed editor improves the readability for testers who are not FDT-trained; it also is useful in writing non-trivial specifications. Moreover, the modeling and specification languages, such as SDL [26,27] and UML [49], enjoy widespread industrial popularity, partially due to their standard graphical representation. Therefore, it will be a natural extension for Estelle to include a graphical editor (such an editor has recently been created [54]). Once all states and transitions of a protocol (including inputs and outputs) are finalized, the writing of the Estelle code itself is fast and straightforward.

Since 188-220 is a multilayer, multifunction protocol of a considerable size and complexity, manual generation of conformance test sequences would be both inefficient and ineffective. As seen from Table 3, the tests already delivered to CECOM contain approximately 10,000 test steps. It is clear that manually generating test sets of this size from the protocol textual description is not a trivial task.

A number of conformance test generation techniques have been proposed [1,8,10,45,51,53,57,60], each of which is expected to give better results for a certain class of protocol specifications depending on the nature and size of the protocol. The experience obtained in generating tests for 188-220 suggests that to successfully test today's complex protocols by using formal methods, an ideal test generation tool should support multiple test generation techniques [40]. They can range from Postman tours [1] or fault-oriented tests [67,70] for mid-size protocols (i.e., the number of states in the order of thousands), to guided random walk approaches [38,71] for large protocols (i.e., the number of states larger than tens of thousands).

State explosion problem has been a major issue for generating FSM models out of EFSM representations of protocols [16,50,69,70]. One common procedure for converting EFSMs into FSMs simultaneously performs reachability analysis and online minimization [16,39]; this conversion is based on combining *equivalent states* [52] by using the notion of *bisimulation equivalence* [46]. Another approach proposes the elimination of inconsistencies in EFSM models [62,63]. Efficient algorithms such as these should be implemented in any test generation tool using FSM models. If the final FSM model is not confined to a manageable size, the test sequences generated from it will be infeasibly long regardless of the test generation method.

Finally, a test house may require its own proprietary format for the executable tests. Although TTCN is accepted as input by many test tools, a proprietary test format may be preferable for a given protocol, if this format is more readable by the domain experts, or is simpler to parse by software tools. The output of a test generation tool should be easily custom-tailored for a particular format, possibly by using simple application generators.

Acknowledgments

We would like to thank Brian Kind of ARINC, Inc., Annapolis, MD, for his valuable suggestions and comments. We are also grateful to the anonymous reviewers for providing many insightful comments that have improved the quality of this paper.

References

[1] 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 rural Chinese postman tours. *IEEE Trans. Commun.*, 39(11):1604–1615, Nov. 1991.

[2] P. D. Amer, G. Burch, A. S. Sethi, D. Zhu, T. Dzik, R. Menell, and M. McMahon. Estelle specification of MIL-STD 188-220A DLL. In *Proc. IEEE Milit. Commun. Conf. (MILCOM)*, Oct. 1996.

[3] P. D. Amer, M. A. Fecko, A. S. Sethi, M. U. Uyar, T. J. Dzik, R. Menell, and M. McMahon. Using Estelle to evolve MIL-STD 188-220. In Budkowski et al. [13], pp. 55–58.

[4] ANSI/IEEE. *ISO/IEC International Standard 8802-2*, Dec. 1994.

[5] B. Baumgarten, H.-J. Burkhardt, and A. Giessler, eds. *Proc. IFIP Int'l Work. Test. Communicat. Syst. (IWTCS)*, Darmstadt, Germany, Sept. 1996. Boston, MA: Kluwer Academic Publ.

[6] A. Bhattacharyya. *Checking Experiments in Sequential Machines.* Wiley & Sons, New York, NY, 1989.

[7] J. Bi and J. Wu. Application of a TTCN-based conformance test environment to the Internet email protocol. In M. Kim, S. Kang, and K. Hong, eds, *Proc. IFIP Int'l Work. Test. Communicat. Syst. (IWTCS)*, pp. 324–330, Cheju Island, Korea, Sept. 1997. Boston, MA: Kluwer Academic Publ.

[8] B. S. Bosik and M. U. Uyar. FSM-based formal methods in protocol conformance testing: from theory to implementation. *Comput. Networks ISDN Syst.*, 22(1):7–34, Sept. 1991.

[9] J. Bredereke and R. Gotzhein. Specification, detection, and resolution of IN feature interactions with Estelle. In *Proc. IFIP Formal Desc. Tech. (FORTE)*, pp. 376–378. Chapman & Hall, 1995.

[10] E. Brinksma. A theory for the derivation of tests. In *Proc. IFIP Protocol Specif. Test. Verif. (PSTV)*. Amsterdam: North-Holland, 1988.

[11] S. Budkowski, A. Cavalli, and E. Najm, eds. *Proc. IFIP Joint Int'l Conf. FORTE/PSTV*, Paris, France, Nov. 1998. Boston, MA: Kluwer Academic Publ.

[12] S. Budkowski and P. Dembinski. An introduction to Estelle: A specification language for distributed systems. *Comput. Networks ISDN Syst.*, 14(1):3–24, 1991.

[13] S. Budkowski, S. Fischer, and R. Gotzhein, eds. *Proc. Int'l Work. FDT Estelle*, Evry, France, Nov. 1998. Evry, France: Institut National des Télécommunications (INT).

[14] R. Burch, P. Amer, and S. Chamberlain. Performance evaluation of MIL-STD 188-220A: Interoperability standard for digital message transfer device subsystems. In *Proc. IEEE Milit. Commun. Conf. (MILCOM)*, San Diego, CA, Nov. 1995.

[15] O. Catrina, E. Lallet, and S. Budkowski. Automated implementation of the Xpress Transport Protocol (XTP) from an Estelle specification. *Electr. J. Networks Distrib. Process.*, (7):3–19, Dec. 1998.

[16] K. T. Cheng and A. S. Krishnakumar. Automatic generation of functional vectors using the extended finite state machine model. *ACM Trans. Design Automation Electronic Syst.*, 1(1):57–79, Jan. 1996.

[17] S.-K. Cheong, K.-H. Lee, and T.-W. Jeong. The analysis of integrating test results for ATM switching systems. In Baumgarten et al. [5], pp. 83–89.

[18] J. Y. Choi and B. K. Hong. Generation of conformance test suites for B-ISDN signalling relevant to multi-party testing architecture. In Baumgarten et al. [5], pp. 316–330.

[19] DoD. *Military Standard—Interoperability Standard for Digital Message Device Subsystems (MIL-STD 188-220B)*, Jan. 1998.

[20] T. Dzik and M. McMahon. MIL-STD 188-220A evolution: A model for technical architecture standards development. In *Proc. IEEE Milit. Commun. Conf. (MILCOM)*, Monterey, CA, Nov. 1997.

[21] M. A. Fecko, P. D. Amer, A. S. Sethi, M. U. Uyar, T. Dzik, R. Menell, and M. McMahon. Formal design and testing of MIL-STD 188-220A based on Estelle. In *Proc. IEEE Milit. Commun. Conf. (MILCOM)*, Monterey, CA, Nov. 1997.

[22] M. A. Fecko, M. U. Uyar, P. D. Amer, and A. S. Sethi. Using semicontrollable interfaces in testing Army communications protocols: Application to MIL-STD 188-220B. In *Proc. IEEE Milit. Commun. Conf. (MILCOM)*, Atlantic City, NJ, Nov. 1999.

[23] M. A. Fecko, M. U. Uyar, A. S. Sethi, and P. D. Amer. Issues in conformance testing: Multiple semicontrollable interfaces. In Budkowski et al. [11], pp. 111–126.

[24] M. A. Fecko, M. U. Uyar, A. S. Sethi, and P. D. Amer. Conformance testing in systems with semicontrollable interfaces. In S. Budkowski, ed, *Protocol Engineering*, vol. X of *Annals Telecommun.* (special issue), Dec. 1999. (in press).

[25] R. Gecse. Conformance testing methodology of Internet protocols: Internet application-layer protocol testing—HTTP. In A. Petrenko and N. Yevtushenko, eds, *Proc. IFIP Int'l Work. Test. Communicat. Syst. (IWTCS)*, pp. 35–48, Tomsk, Russia, Sept. 1998. Boston, MA: Kluwer Academic Publ.

[26] D. Hogrefe. Validation of SDL systems. *Comput. Networks ISDN Syst.*, 28(12), 1996.

[27] Int'l Telecomm. Union, Geneva, Switzerland. *ITU Recommendation Z100: Specification and Description Language (SDL)*, 1989.

[28] ISO, Information Processing Systems—OSI, Geneva, Switzerland. *ISO/IEC International Standard 8571-1: File Transfer, Access and Management—Part 1: General introduction*, 1988.

[29] ISO, Information Processing Systems—OSI. *ISO International Standard 9074: Estelle—A Formal Description Technique Based on an Extended State Transition Model*, 1989.

[30] ISO, Information Technology—OSI, Geneva, Switzerland. *ISO/IEC International Standard 13712-3: Remote Operations: OSI realizations—Remote Operations Service Element (ROSE) protocol specification*, 1995.

[31] ISO, Information Technology—OSI, Geneva, Switzerland. *ISO/IEC International Standard 8327-1: Connection-oriented Session Protocol—protocol specification*, 1996.

[32] ISO, Information Technology—OSI, Geneva, Switzerland. *ISO/IEC International Standard 10026-3: Distributed Transaction Processing—Part 3: Protocol specification*, 1998.

[33] ITU. *Recommendation Q.2110: Service Specific Connection-Oriented Protocol (SSCOP).*

[34] A. Jirachiefpattana and R. Lai. Uncovering ISO ROSE protocol errors using Estelle. *Comput. Stand. Interf.*, 17(5-6):559–583, 1995.

[35] S. Kang, Y. Seo, D. Kang, M. Hong, J. Yang, I. Koh, J. Shin, S. Yoo, and M. Kim. Development and application of ATM protocol conformance test system. In G. Csopaki, S. Dibuz, and K. Tarnay, eds, *Proc. IFIP Int'l Work. Test. Communicat. Syst. (IWTCS)*, pp. 331–346, Budapest, Hungary, Sept. 1999. Boston, MA: Kluwer Academic Publ.

[36] T. Kato, T. Ogishi, A. Idoue, and K. Suzuki. Intelligent protocol analyzer with TCP behavior emulation for interoperability testing of TCP/IP protocols. In *Proc. IFIP Joint Int'l Conf. FORTE/PSTV*, pp. 449–464, Osaka, Japan, Nov. 1997.

[37] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, New York, NY, 1978.

[38] D. Lee, K. K. Sabnani, D. M. Kristol, and S. Paul. Conformance testing of protocols specified as communicating FSMs—a guided random walk approach. *IEEE Trans. Commun.*, 44(5), May 1996.

[39] D. Lee and M. Yannakakis. Online minimization of transition systems. In *Proc. 24th Annual ACM*, Victoria, Canada, 1992.

[40] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines—a survey. *Proc. IEEE*, 84(8):1090–1123, Aug. 1996.

[41] D. Y. Lee and J. Y. Lee. Test generation for the specification written in Estelle. In *Proc. IFIP Protocol Specif. Test. Verif. (PSTV)*, Stockholm, Sweden, June 1991.

[42] D. Y. Lee and J. Y. Lee. A well-defined Estelle specification for the automatic test generation. *IEEE Trans. Comput.*, 40(4), Apr. 1991.

[43] J. K. Lenstra and A. H. G. Rinnooy Kan. On general routing problems. *Networks*, 6:273–280, 1976.

[44] H. Li, P. Amer, and S. Chamberlain. Estelle specification of MIL-STD 188-220A: Interoperability standard for digital message transfer device subsystems. In *Proc. IEEE Milit. Commun. Conf. (MILCOM)*, San Diego, CA, Nov. 1995.

[45] R. E. Miller and S. Paul. On the generation of minimal-length conformance tests for communication protocols. *IEEE/ACM Trans. Network.*, 2(1):116–129, Feb. 1993.

[46] R. Milner. *Communication and Concurrency.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[47] P. Mondain-Monval. ISO Session Service specification in Estelle. Technical Report SEDOS Rep. 70, ESPRIT Project, Nov. 1986.

[48] C. Negulescu and E. Borcoci. SSCOP protocol throughput evaluation—simulation based on Estelle specification. In Budkowski et al. [13], pp. 75–98.

[49] Object Management Group, Framingham, MA. *OMG Standard: Unified Modeling Language (UML) 1.1*, 1997.

[50] D. H. Pitt and D. Freestone. The derivation of conformance tests from LOTOS specifications. *IEEE Trans. Softw. Eng.*, 16(12):1337–1343, 1990.

[51] J. Romijn and J. Springintveld. Exploiting symmetry in protocol testing. In Budkowski et al. [11], pp. 337–351.

[52] K. K. Sabnani and A. T. Dahbura. A protocol test generation procedure. *Comput. Networks ISDN Syst.*, 15:285–297, 1988.

[53] B. Sarikaya, G. v. Bochmann, and E. Cerny. A test design methodology for protocol testing. *IEEE Trans. Softw. Eng.*, 13(5):518–531, May 1987.

[54] J. Templemore-Finlayson, J l. Raffy, P. Kritzinger, and S. Budkowski. A graphical representation and prototype editor for the formal description technique Estelle. In Budkowski et al. [11], pp. 37–55.

[55] R. Tenney. A tutorial introduction to Estelle. Technical Report 88-1, Univ. of Mass, Boston, June 1988.

[56] J. Thees. Protocol implementation with Estelle—from prototypes to efficient implementations. In Budkowski et al. [13], pp. 187–193.

[57] J. Tretmans. Conformance testing with labelled transitions systems: Implementation relations and test generation. *Comput. Networks ISDN Syst.*, 29(1):49–79, 1996.

[58] K. Turner. *Formal Description Techniques.* North-Holland, Amsterdam, 1989.

[59] H. Ural and B. Yang. A test sequence selection method for protocols specified in Estelle. Technical Report TR-88-18, Univ. of Ottawa, June 1988.

[60] H. Ural and B. Yang. A test sequence selection method for protocol testing. *IEEE Trans. Commun.*, 39(4), 1991.

[61] M. U. Uyar and A. T. Dahbura. Optimal test sequence generation for protocols: the Chinese postman algorithm applied to Q.931. In *Proc. IEEE Global Telecomm. Conf. (GLOBECOM)*, pp. 68–72, Dec. 1986.

[62] M. U. Uyar and A. Y. Duale. Modeling VHDL specifications as consistent EFSMs. In *Proc. IEEE Milit. Commun. Conf. (MILCOM)*, Monterey, CA, Nov. 1997.

[63] M. U. Uyar and A. Y. Duale. Removal of inconsistencies in VHDL specifications. In *Proc. US Army Res. Lab. (ARL) Fed. Lab. Symp.–ATIRP Conf.*, College Park, MD, Feb. 1998.

[64] M. U. Uyar, M. A. Fecko, A. S. Sethi, and P. D. Amer. Minimum-cost solutions for testing protocols with timers. In *Proc. IEEE Int'l Perf. Comput. Commun. Conf. (IPCCC)*, pp. 346–354, Phoenix, AZ, Feb. 1998.

[65] M. U. Uyar, M. A. Fecko, A. S. Sethi, and P. D. Amer. Testing protocols modeled as FSMs with timing parameters. *Comput. Networks*, 31(18):1967–1988, Sept. 1999.

[66] M. U. Uyar and M. H. Sherif. Protocol modeling for conformance testing: Case study for the ISDN LAPD protocol. *AT&T Technical J.*, 69(1), Jan. 1990.

[67] G. v. Bochmann, A. Das, R. Dssouli, M. Dubuc, A. Ghedamsi, and G. Luo. Fault models in testing. In *Proc. IFIP Int'l Work. Protocol Test Syst. (IWPTS)*, pp. 17–30. Amsterdam: North-Holland, 1992.

[68] E. Vázquez, P. Sandoval, M. Sedano, and J. Vinyes. Automatic implementation of TP4/IP with an Estelle workstation—development methodology and performance evaluation. In *Proc. IFIP Protocol Specif. Test. Verif. (PSTV)*, pp. 125–139. Amsterdam: North-Holland, 1992.

[69] C. J. Wang and M. T. Liu. Axiomatic test sequence generation for extended finite state machines. In *Proc. 12th Conf. Distrib. Comput. Syst.*, pp. 252–259, 1992.

[70] C. J. Wang and M. T. Liu. Generating test cases for EFSM with given fault models. In *Proc. IEEE Int'l Conf. Comput. Commun. (INFOCOM)*, pp. 774–781, 1993.

[71] C. West. Protocol validation by random state exploration. In *Proc. IFIP Protocol Specif. Test. Verif. (PSTV)*. Amsterdam: North-Holland, 1986.

[72] J. Wytrębowicz and P. Roliński. Analysis tools for Estelle specifications. In Budkowski et al. [13], pp. 141–155.

[73] XTP Forum, Santa Barbara, CA. *Xpress Transport Protocol Specification, Rev. 4.0*, 1995.

[74] S. Yoo, L. Collica, and M. Kim. Conformance testing of ATM Adaptation Layer protocol. In Baumgarten et al. [5], pp. 237–252.

Appendix

A Examples of errors found in 188-220

- [3] The standard indicates in Section 5.3.8.1.2.e that k is a link parameter defined to be the "maximum number of outstanding I PDU's." But in Section 5.3.7.2.5.1, we see that the "...RR command is sent to a destination station when the k value at the originating station reaches half of the k value for that connection." In Section 5.3.7.2.5.1, the first reference to k indicates that k is a variable which is being used as a counter, while the second reference to k has k being a static parameter for that connection. In this case, the standard was changed such that the variable counting the number of outstanding I PDUs was not referred to as k.
- Section 5.3.7.2.5.4.2 states that "When an I PDU has been received and not more than one frame is missing, the station may retain the information field of the out-of-sequence I PDUs and send a SREJ PDU for the missing I PDU." This sentence implies at most one missing I PDU when sending an SREJ PDU. The next sentence, however, states "A station may transmit one or more SREJ PDUs, each containing a different N(R) with P-bit set to 0." This sentence implies that there can be more than one missing I PDU (otherwise why send multiple SREJ PDUs with different N(R)?). At the 13 Mar meeting, the WG changed this section to indicate that an SREJ PDU may be sent when *at least one*, rather than "no more than one," I PDU is detected as missing.
- In Section 5.3.6.1.9, the standard indicates that "the URNR response PDU shall be used to reply to a UI command PDU with the P-bit set to 1, if the UI command cannot be processed due to a busy condition." Since the UI cannot be processed, this indicates the URNR does *not* acknowledge the UI. However, Section 5.3.5.2.3.2 states that "the F-bit set to 1 shall be used to acknowledge the receipt of a command PDU with the P-bit set to 1." And Section 5.3.7.1.5.4 indicates that "a URNR response PDU, with F-bit set to 1, may be sent by the remote station to advise the originator of the associated UI command PDU that it is experiencing a busy condition and is unable to accept UI PDU's." The combination of the last two excerpts implies that URNR response PDU's *do* acknowledge the corresponding UI PDU. A correction of this contradiction was approved at the 13 Mar 96 WG meeting. Section 5.3.5.2.3.2 was changed to read "the F-bit set to 1 shall be used to *respond* to the receipt of a command PDU with P-bit set to 1." No acknowledgment is implied.

[3] Section numbers refer to the MIL-STD 188-220 official document.