FORMAL SPECIFICATION AND CONFORMANCE TESTING OF ARMY COMMUNICATIONS PROTOCOLS

Paul D. Amer
Mariusz A. Fecko
Adarshpal S. Sethi

Computer and Information Sciences Department
University of Delaware, Newark, DE

M. Ümit Uyar
Ali Y. Duale

Electrical Engineering Department
The City College of the City University of New York, NY

ABSTRACT

During the past six years, ATIRP-sponsored faculty and students from the University of Delaware and the City College of New York, collaborating with scientists from CECOM and ARL, have helped advance the state-of-the-art in the design, development, and testing of Army communications protocols[1]. *Working closely together, these groups specified a complex real-life protocol (MIL-STD 188-220) in Estelle, and then used that formal specification to generate conformance test sequences. The test generation effort involved identifying and publishing results on three theoretical problems: (1) the timing constraint problem, (2) the controllability problem, and (3) the conflicting timers problem. Based on ATIRP's research results, two software packages were written to generate conformance test sequences for 188-220. These packages helped generate tests for 188-220's Data Link Types 1 and 4 services that were realizable without timer interruptions while providing a 200% increase in test coverage. The test cases have been delivered and are being used by a CECOM conformance testing facility.*

Keywords: conformance testing, Estelle, formal description technique, formal specification, MIL-STD 188-220, protocol specification, test case generation

[1]This work supported by the US ARO (DAAH04-94-G-0093), and prepared through collaborative participation in the Advanced Telecommunications/Info Dist'n Research Program (ATIRP) Consortium sponsored by the US Army Research Lab under Fed Lab Program, Cooperative Agreement DAAL01-96-2-0002. M. Fecko is with Telcordia Applied Research, NJ. A. Duale is with IBM, NY.

I. Introduction

This paper summarizes a successful six-year effort to use the Estelle formal description technique to specify a complex real-life protocol - Military Standard (MIL-STD) 188-220 - and then use that specification to automatically generate conformance tests for use in implementation testing. A key factor in this success story has been the ATIRP-sponsored collaboration among five groups: University of Delaware (UD), City College of the City University of New York (CCNY), the Army Research Laboratory (ARL), US Army Communications-Electronics Command (CECOM), and the Joint Combat Net Radio Working Group (CNR-WG). 188-220 is being developed in the US Army, Navy and Marine Corps systems for mobile combat network radios [18]. As a result of this collaboration, 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 networking 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 six years. Section II provides the background on the collaboration among the MIL-STD 188-220 sponsors, research and development teams, and standards organizations. Sections III and IV overview the formal description technique Estelle and 188-220, respectively. Section V presents a part of the Estelle specification of 188-220, and gives examples of errors and ambiguities found as a result of formally specifying the protocol. A general approach adopted at UD and CCNY to test generation from an Estelle formal specification is described in Section VI. This section also summarizes four years of ATIRP-supported published research results in test generation based on formal specifications. Section VII presents two systems of software: (1) *efsm2fsm-rcpt*, and (2) INDEEL implemented to help generate conformance tests. Section VIII summarizes our practical test generation results - the technology transfer of tests from ATIRP to CECOM. Finally, Section IX presents the authors' personal perspective on how the protocol development process is in general improved thanks to using formal methods.

II. History of MIL-STD 188-220 Development

Formal methods in communications protocol specification and conformance testing have been widely used in the design and testing of real-life protocols [6], [16], [17], [28], [39], [40], [84]. In particular, the Estelle formal description technique (FDT) [11], [32], [61], [64] has been used on several occasions to resolve ambiguities within international protocols [8], [14], [38], [53], [62], [77].

In 1994, UD's Protocol Engineering Laboratory began its involvement with the US Army in using Estelle to formally specify the military standard MIL-STD 188-220 [18]. An initial small contract with the Army Research Laboratory supported both simulation and specification of the 1993 version of 188-220 [13], [49]. This formal specification research effort received the attention of the CECOM Software Engineering Center in NJ. CECOM 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, over fifty changes to the English specification of 188-220 resulted from UD's efforts using Estelle to formally specify the standard [2], [18] (see Figure 1). While the English text takes precedence in case of disagreement with the formal specification, *UD's Estelle specification of 188-220 is an official part of the military standard.* [2] It represents one of the first major national or international standards officially including an Estelle specification. Other examples include [31], [34], [35], [52].

[2]From this point on, 188-220 refers to version B, approved 1/98.

During this period, CECOM has been concurrently developing a Conformance Tester that can automatically evaluate a 188-220 implementation identifying its conformance with the standard. Our 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 is far beyond specifying with pure FSMs, hence the need to use a more powerful specification language such as Estelle, International Standard 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.

Automatic generation of tests from Estelle specifications presented various theoretical problems:

1. During testing, if active timers were not taken into account when the tests were generated, these timers can disrupt the test sequences, thereby failing correct implementations or worse, passing incorrect ones. For accurate testing, timers must 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) [7]. 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 are difficult or impossible to access during real testing conditions. 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

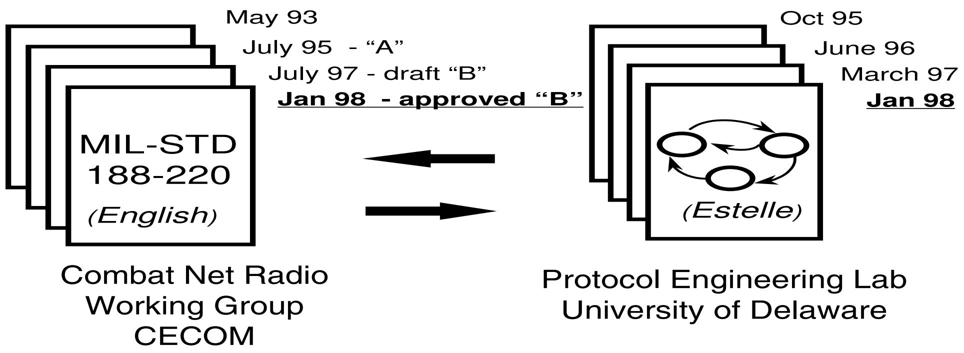


Fig. 1. History of MIL-STD 188-220 Development

conflicting conditions based on a protocol's variables are resolved (the INDEEL software package (Section VII-B addresses this problem).
4. In particular, infeasible test sequences may result from a protocol's variables modeling multiple timers that may be running simultaneously (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], [72], 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. The most recent research focuses on the conflicting timers problem.

III. Estelle

In 1989, Estelle was published as one of two ISO International Standard Formal Description Techniques (FDT) for the specification of computer communication protocols [11], [32]. 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 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 [61], [64].

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.

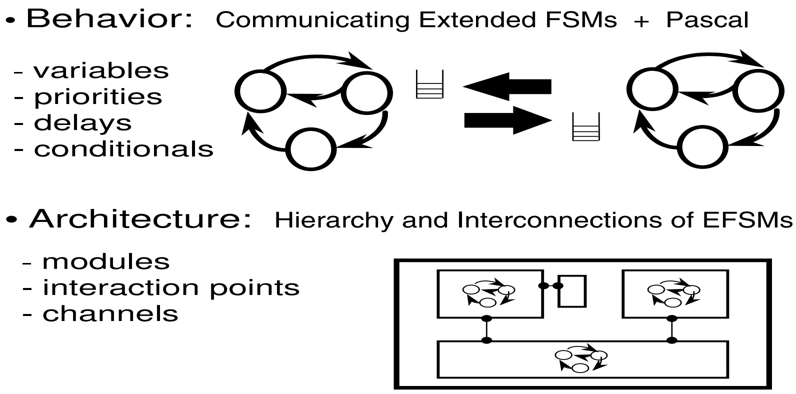


Fig. 2. Estelle: ISO International Standard 9074

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.

IV. 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 are: seamless connectivity of C^4I systems (discussed briefly in Section IX), 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.

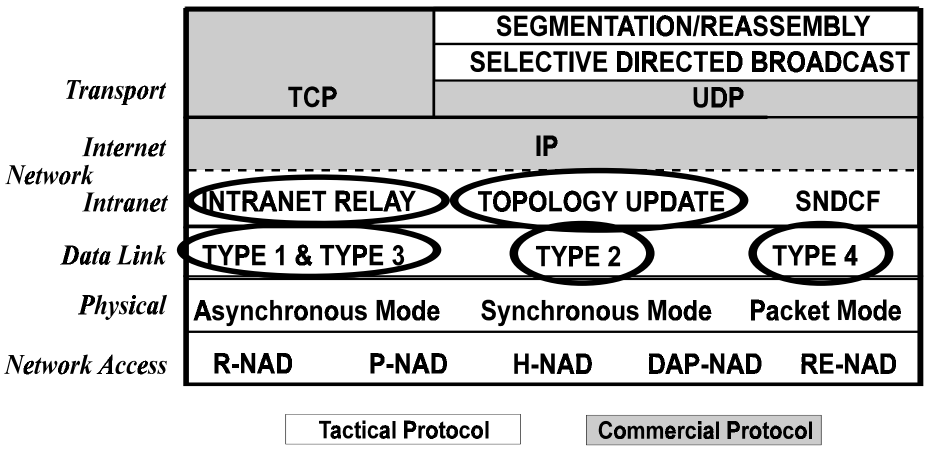


Fig. 3. MIL-STD 188-220 Protocol Architecture. The circles indicate those parts of the protocol where FDTs were used during the development.

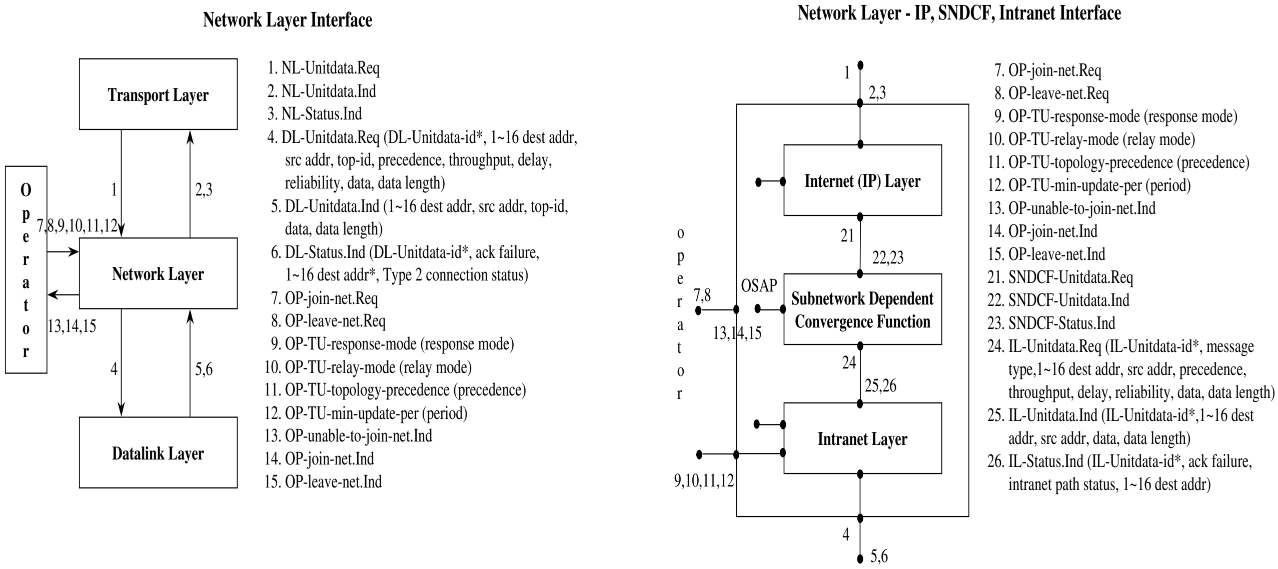


Fig. 4. Network Layer Interface and Architecture

V. 188-220 Estelle Specification

To help a reader realize the magnitude of formally specifying a protocol of 188-220 size and complexity, we provide some numbers. The Datalink and Network layer specifications consist of 69 and 19 documents, respectively, describing the architecture, interfaces, EFSM, and state table of each module. The Datalink layer specification is accompanied by three Estelle source code files (for Datalink classes A, B, and C) 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 should see `www.cis.udel.edu/~amer/CECOM/`. In the next section, we present an overview of the Network Layer

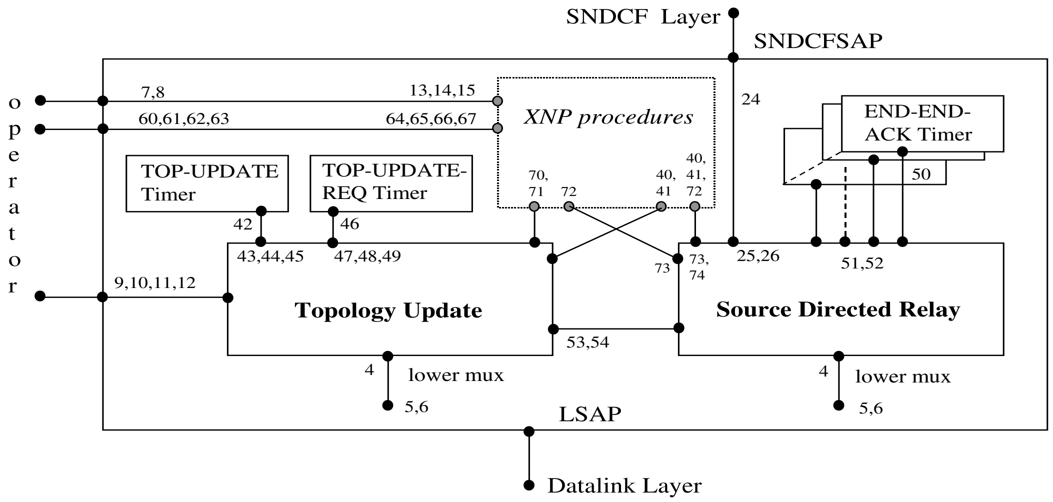


Fig. 5. Intranet Layer Architecture

architecture with a focus on the Topology Update and Source Directed Relay functions of the Intranet sublayer.

A. 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.[3]

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.

B. Problems and Ambiguities Found in 188-220 through Formal Specification

The primary goals in developing a formal specification of a protocol are to:

[3]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).

1. discover and document problems and ambiguities that are commonly seen in a standard written in natural language,
2. verify the protocol,
3. simulate the protocol,
4. automate code implementation, and
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 and Intranet Layers, more than fifty problems in the original English specification have been documented. All of these problems were reported back to the CNR Working Group and subsequently corrected in the standard. Here we present just two 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 ambiguous 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.

VI. 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 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 [46], [47], [65], [66], [82]. 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 expansion would be useful since methods exist for generating tests directly from pure FSMs (e.g., [1]). Unfortunately, completely converting even a simple EFSM 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, the UD and CCNY ATIRP research group used 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], [7], [43], [50], [59], [66], using a deter-

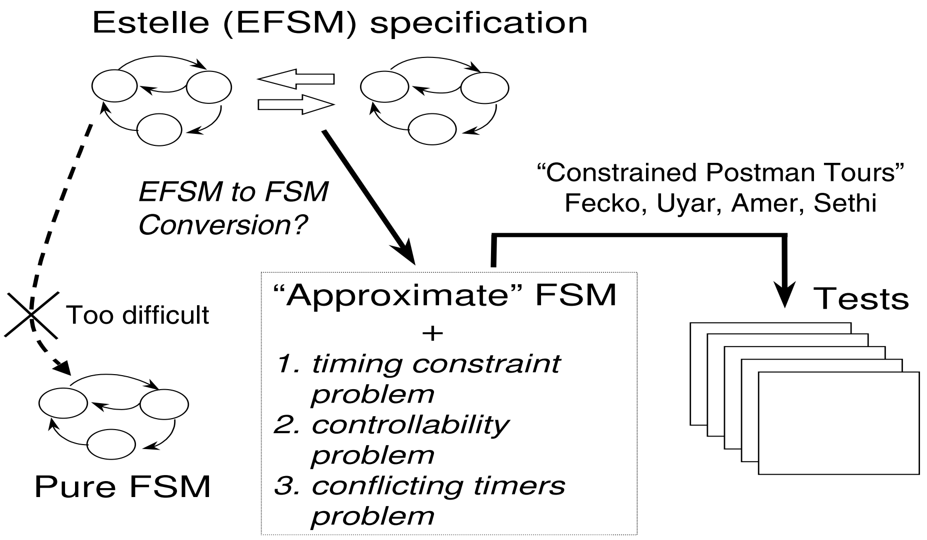


Fig. 6. Test Generation from Extended FSMs

ministic 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 VI-A and VI-B, we outline our earlier research results to eliminate the timing constraints and controllability problems which appear in the EFSM model of the 188-220. Our recent research focus on the so-called *conflicting timers problem*, where infeasible test sequences may be generated unless conflicting conditions based on timers are resolved. These results are described in Section VI-C.

A. Research Area 1: The 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.

Our research resuls [72], [73] optimize 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 [48] 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 [72]. The later paper [73] generalizes these earlier results to both self-loop and non-self-loop verification sequences.

A.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 [18] for Combat Net Radio communication, and LAPD [74], 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 188-220's Intranet Layer has three active states in which one or two timers are running [72].

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 [58], distinguishing sequences [5], [42], or characterizing sequences [5], [42].

A.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 [72], we prove that the minimum number of times vertex v_i must be visited in a test sequence is as follows:

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

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

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

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

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

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

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

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

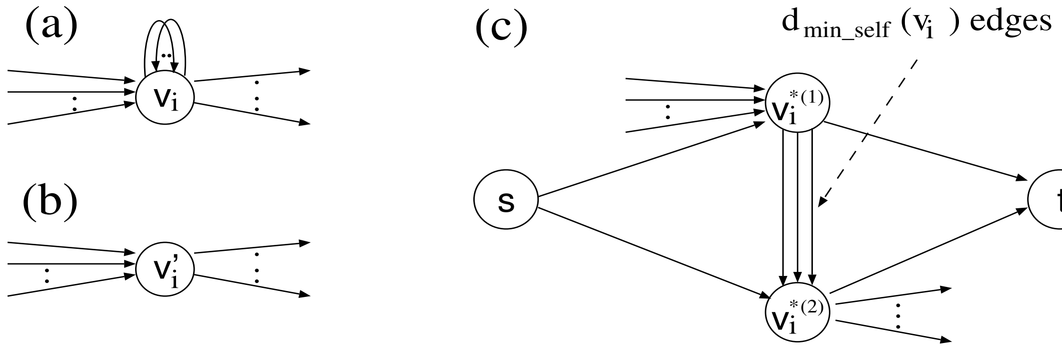


Fig. 7. Conversion of v_i in G (part (a)), to $v_i^{'}$ in $G^{'}$ (part (b)) and to $v_i^{*(1)}, v_i^{*(2)}$ in G^* (part (c)).

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 [68] when applied to the graph without any self-loop repetition constraint results in the test sequence

$$\underline{e0, \mathbf{e0}, e1, \mathbf{e2}, e2, \mathbf{e2}, e10}, \mathbf{e9}, e9, \mathbf{e9}, e12, \mathbf{e0}, e1, e3, \mathbf{e2}, e4, \mathbf{e6}, \mathbf{e7}, e6, \mathbf{e6}, \mathbf{e7}, e11, \mathbf{e9}, e12, e1, e4, e7, \mathbf{e6}, \mathbf{e7}, e8, \mathbf{e6}, \mathbf{e7}, e5, \mathbf{e0} \quad (6)$$

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 to the graph shown in Figure 9. Since in this example all UIO sequences are self-loops, the simplified conversion presented in [72] 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

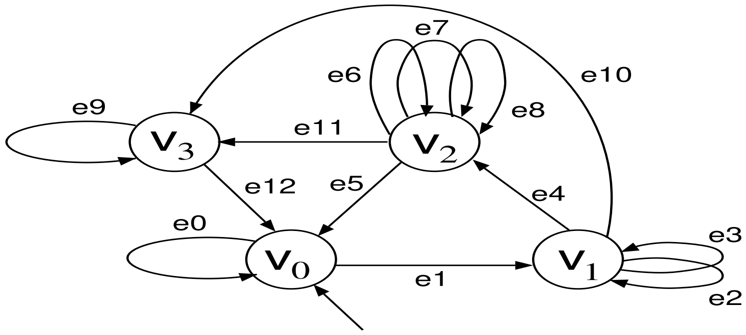
$$e0, \mathbf{e0}, e1, \mathbf{e2}, e10, \mathbf{e9}, e9, \mathbf{e9}, e12, \mathbf{e0}, e1, e2, \mathbf{e2}, e4, \mathbf{e6}, \mathbf{e7}, e11, \mathbf{e9}, e12, e1, e3, \mathbf{e2}, e4, e6, \mathbf{e6}, \mathbf{e7}, e5, \mathbf{e0}, e1, e4, e7, \mathbf{e6}, \mathbf{e7}, e5, e1, e4, e8, \mathbf{e6}, \mathbf{e7}, e5 \quad (7)$$

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.

B. Research Area 2: The Controllability Problem

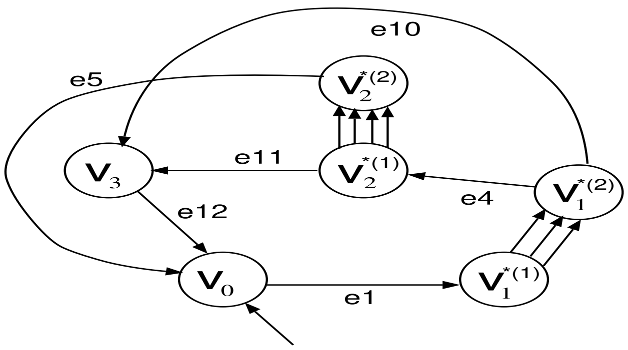
Consider a testing framework where the interface I_1 between the IUT and the (N)-layer in the System Un-



Test sequence (34 edges)

e0 **e0** e1 **e2** e2 **e2** e10 **e9** e9 **e9** e12 **e0** e1 e3 **e2** e4
e6 e7 e6 **e6 e7** e11 **e9** e12 e1 e4 e7 **e6 e7** e8 **e6 e7** e5 **e0**

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



Test sequence (40 edges)

e0 **e0** e1 **e2** e10 **e9** e9 **e9** e12 **e0** e1 e2 **e2** e4 **e6 e7** e11 **e9** e12
e1 e3 **e2** e4 e6 **e6 e7** e5 **e0** e1 e4 e7 **e6 e7** e5 e1 e4 e8 **e6 e7** e5

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

der Test (SUT) [7] 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.*

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.

B.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

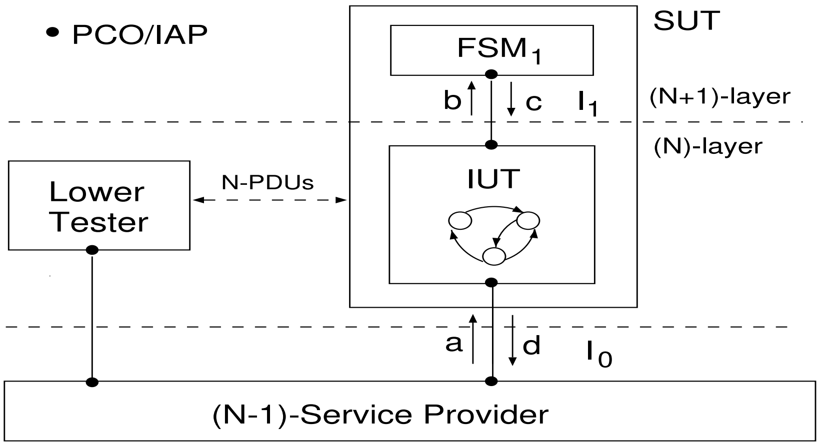


Fig. 10. Testing (N)-layer IUT with an (N+1)-layer semicontrollable interface.

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 [18], [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], [36].

B.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 $G^{'}(V^{'}, E^{'})$ pro-

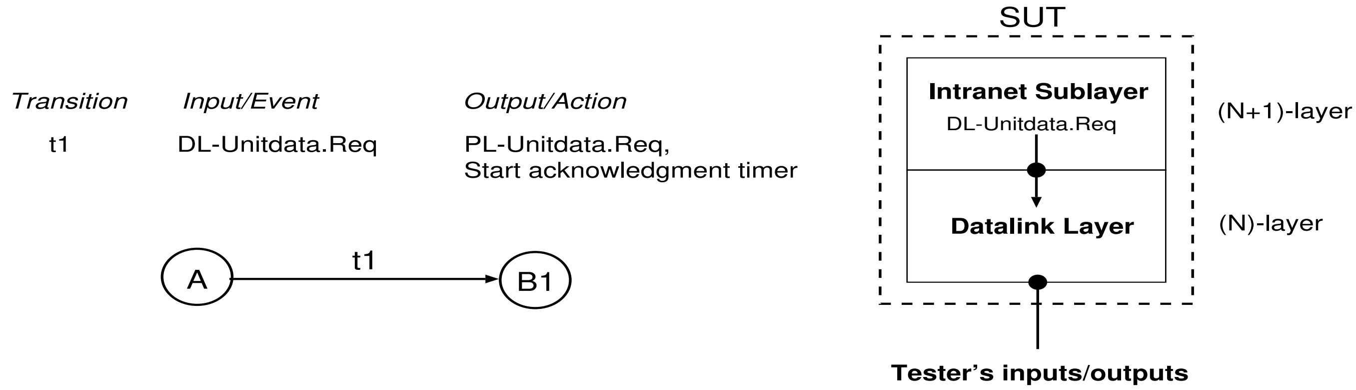


Fig. 11. MIL-STD 188-220: Example of the controllability problem

ceeds 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 $e^{'}_{new} = (v^{'}, v^{'}_{new}) \in E^{'}$
• 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 G) is described in Table I. 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

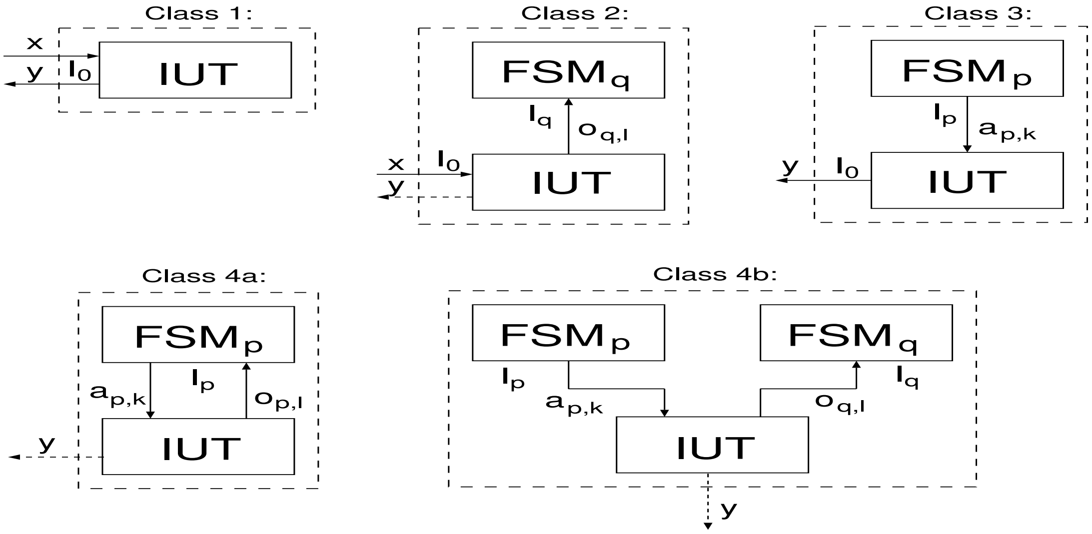


Fig. 12. Classes of edges in $G^{'}$ (dashed-lined outputs are optional).

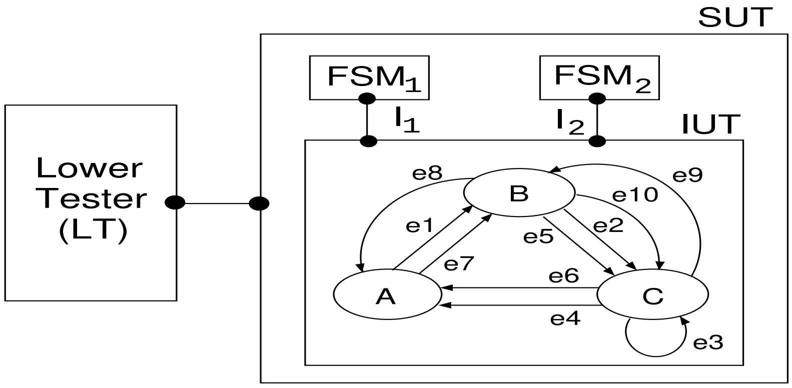


Fig. 13. IUT interacting with two semicontrollable interfaces.

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 II. Steps with ($\rightarrow$) indicate that an edge is tested in this step. Note that, for simplicity, the UIO sequences [58] are not included in this sequence.

C. Research Area 3: The Conflicting Timers Problem

To ensure feasibility of tests in a laboratory, automated test generation for network protocols with timer requirements must consider conflicting conditions based on a protocol's timers. Our ATIRP research developed a new model for testing real-time protocols with multiple timers, which captures complex timing dependencies by using simple linear expressions involving timer-related variables. Similar dependencies, but based on arbitrary linear variables, are present in EFSM models of VHDL specifications [69]. Uyar and Duale present algorithms for detecting [69] and removing [19], [76] such inconsistencies in VHDL specifications. The new modeling technique combined with the inconsistency removal algorithms are expected to significantly shorten test sequences without compromising their fault coverage.

The model, specifically designed for testing purposes, avoids performing a full reachability analysis and significantly limits the explosive growth of the number of test scenarios. These goals are achieved by incorporating certain rules for the graph traversal without reducing the set of testable transitions. The technique also models a realistic testing framework in which each I/O exchange takes a certain time to realize, and a tester has an ability to turn timers *on* and *off* in arbitrary transitions and to algorithmically find proper timeout settings.

The methodology presented in this paper is expected to detect transfer and output faults [45], where an IUT moves into a wrong state (a state other than the one specified) or generates a wrong output (an output other than the one specified) to a given input, respectively. The detection of transfer faults can significantly be improved by using the well-known state verification methods such as UIO sequences, characterization sets, or distinguishing sequences. These techniques should be applied while generating a minimum-cost test sequence from the final conflict-free graph.

The proposed solution is likely to have a broader application due to a proliferation of protocols with real-time requirements. The functional errors in such protocols are usually caused by the unsatisfiability of time constraints and (possibly conflicting) conditions involving timers; therefore, significant research is required to develop efficient algorithms for test generation for such protocols. Our methodology is expected to contribute towards achieving this goal. The preliminary results are reported in [26].

In the test cases delivered to CECOM (see Section VIII), conflicting conditions based on 188-220's timers are resolved by manually expanding EFSMs based on the set of conflicting timers. This procedure results in test sequences that are far from minimum-length. The technique presented here allows us to automatically generate conflict-free test sequences for 188-220.

Suppose that a protocol specification defines a set of timers $K = \{tm_1, \ldots, tm_{|K|}\}$, such that a timer tm_j may be started and stopped by arbitrary transitions defined in the specification. Each timer tm_j can be associated with a boolean variable T_j whose value is true if tm_j is running, and false if tm_j is not running. Let ϕ be a time formula obtained from variables $T_1, \ldots, T_k$ by using logical operands $\wedge$, $\vee$, and $\neg$. Suppose that a specification contains transitions with time conditions of a form "if ϕ" for some time formula ϕ. It is clear that there may exist infeasible paths in an FSM modeling a protocol, if two or more edges in a path have inconsistent conditions. For example, for transitions e_1: $\texttt{if } (T_j) \texttt{ then } \{\varphi_1\}$ and e_2: $\texttt{if } (\neg T_j) \texttt{ then } \{\varphi_2\}$, a path (e_1, e_2) is inconsistent unless the action of φ_1 in e_1 sets T_j to false (which happens when timer tm_j expires in transition e_1). The solution to the above problem is expected to allow generating low-cost tests free of such conflicts.

188-220's Datalink Layer Estelle specification defines several timers that can run concurrently and affect the protocol's behavior. 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 not running, a buffered frame is retransmitted. If, however, ACK timer expires while $BUSY$ timer is running, no output is generated. Besides Estelle specifications, feasibility constraints related to multiple concurrent timers are also of special concern for specifications in SDL.

The conflicting timers problem is a special case of the feasibility problem of test sequences, which is an open research problem for the general case [27], [67]. However, there are two simplifying features of the conflicting timers problem: (1) timer-related variables are linear, and (2) the values of time-keeping variables implicitly increase with time. Considering these features makes it possible to find an efficient solution to this special case.

C.1 General approach

The goal of the presented technique is to achieve the following fault coverage: *cover every feasible state transition defined in the specification at least once.* During the testing of a system with multiple timers, when a node v_p is visited, an efficient test sequence should either (1) traverse as many self-loops (i.e., transitions that start and end in the same state) as possible before a timeout or (2) leave v_p immediately through a non-timeout transition. Once the maximum allowable number of self-loops are traversed, a test sequence may leave v_p through any outgoing transition. Such an approach does not let perform full reachability analysis; however, it can be shown that considering only the above two cases is sufficient to include at least one feasible path for each transition provided such a feasible path is not prohibited by the original specification.

Suppose that there are 15 untested self-loops (each requiring 1 sec to test) in state v_{57}, and that, when the test sequence visits v_{57}, the earliest timer to expire is tm_4, with 10.5 sec remaining until its timeout. In this example, the test sequence will either leave v_{57} immediately or traverse 10 of the untested self-loops. Suppose that the latter option is chosen and, later during the test sequence traversal, v_{57} is visited again with tm_2 leaving 3.1 sec until the earliest timeout. In this case, 3 more untested self-loops of v_{57} can be covered by the test sequence. Traversal will continue until all of the v_{57}'s self-loops are tested.

In more complicated cases, in addition to the aforementioned timing constraints, traversal of a self-loop requires that its associated time condition be satisfied, i.e., certain timers be active (or, similarly, other timers be inactive). These time conditions will also be taken into account while selecting which self-loops to traverse. In the above example, if 6 or more self-loops of v_{57} have 'tm_4 not running' as their time condition, the test sequence, which tries to execute 10 of the untested self-loops, will cause a timer conflict due to the unsatisfiability of the time condition.

In general, the goal of an optimization is to generate a low-cost test sequence that follows the above guidelines, satisfies time conditions of all composite edges and is not disrupted by timeout events during traversal (i.e., contains only feasible transitions).

Similar inconsistencies, but based on arbitrary linear variables, are present in EFSMs modeling VHDL specifications. ATIRP researchers Uyar and Duale presented algorithms for detecting [69] and removing [70] inconsistencies in VHDL specifications. Recent research in UD and CCNY focused on adapting these algorithms to detecting and removing inconsistencies caused by a protocol's conflicting timers. The software implementation of these algorithms developed within ATIRP is described in the next section.

VII. Software for Automated Test Generation

The process of generating tests involved the development of two systems of software: (1) *efsm2fsm-rcpt*, and (2) INDEEL. These two systems are now described in turn.

A. efsm2fsm-rcpt

Figure 15 depicts the major software components that were developed to generate test sequences from an EFSM [25]. The software contains two packages: (1) *efsm2fsm*, and (2) *rcpt*. The former was designed and implemented at UD. The latter was based on the software written at CCNY, which originally was able to handle graphs of at most 100 transitions in a plain input/output format, without any of the additional parameters specifically required for 188-220B tests. This component was enhanced to generate tests for 188-220B for a proprietary CECOM's format. Also, the software was significantly redesigned to process large graphs (1000s of transitions), which enabled its application to more complex real-life protocols.

A.1 *efsm2fsm*

efsm2fsm takes a protocol's EFSM representation as input and performs its expansion to an FSM. Each EFSM's transition is associated with the following parameters: transition name in the Estelle specification, transition description, start and end states, input and output names, numerical values specifying the corresponding fields in 188-220B's PDUs, and changes in the variables' values (i.e., start and end configurations. To express the start and end configurations, a simple notation was defined. In the potential future work on this package, it is essential that this notation be replaced with a different one, which should be more expressive and flexible.

To facilitate creating the input to *efsm2fsm*, spontaneous transitions are allowed to be specified in the input EFSM. These transitions are then concatenated with regular transitions (i.e., triggered by an external input) to eliminate spontaneous transitions from the resulting FSM. This procedure can be briefly described as follows. Suppose that in a path

$$v_0 \xrightarrow{t_1} v_1 \xrightarrow{t_2} v_2 \ldots v_{i-1} \xrightarrow{t_i} v_i \ldots v_{n-1} \xrightarrow{t_n} v_n \qquad (8)$$

where v_i and t_i denote a state and a transition, respectively, t_1 is regular and $t_2, \ldots, t_n$ are spontaneous. Then transitions $t_1, \ldots, t_n$ are concatenated into a single transition $t_{1,n}$ from state v_0 to state v_n. Their inputs, outputs, and other parameters are combined and associated with transition $t_{1,n}$. States $v_2, \ldots, v_{n-1}$ are marked as temporary, and subsequently removed from the FSM along with their outgoing transitions.

After the expansion to an FSM, transitions that are equivalent from a testing point of view could be identified, leading to a minimum-cost test sequence covering at least one transition from each equivalence class. However, building such a test sequence is NP-hard [25]. Therefore, simple heuristics bringing about 20%-30% reduction in the number of transitions were implemented.

It is possible to manually prepare the input file for the package such that an EFSM's states are divided into two groups: (1) states with no inputs buffered, and (2) states with one input buffered at a semicontrollable interface. Then semicontrollable interfaces can be utilized for certain simplified cases such as using the 188-220B Intranet layer for indirect testing of 188-220B Datalink layer (in these tests, only one semicontrollable interface is used with a small number of semicontrollable inputs). A self-loop repetition constraint can be taken into account for the case of self-loop state verification sequences.

To run the package for a protocol's EFSM specified in file *protocol.efsm*, the following command must be used:

efsm2fsm protocol.efsm [-options]

producing two files *protocol.fsm* and *protocol.stat*. The former contains the output FSM. All information associated with transitions in the input EFSM is preserved. This enables the *rcpt* package to populate the fields defined in the CECOM's proprietary format for test sequences. The latter file contains statistics such as the number of states and transitions in the EFSM/FSM, and the percentage effectiveness of the reduction heuristics.

Note that the original EFSM to FSM conversion technique implemented should be replaced by the application of the inconsistency elimination algorithms implemented in INDEEL (see Section VII-B). Using INDEEL to eliminate inconsistencies results in a conflict-free EFSM that is significantly smaller than the FSM.

A.2 *rcpt*

The FSM produced by *efsm2fsm* is then fed to *rcpt*, which builds a corresponding directed graph representation G. Then, network flow techniques are applied to find a rural symmetric augmentation of G as G''. Finally, *rcpt* finds an Euler tour of G'', and outputs to a file a resulting test sequence conforming to the CECOM's proprietary format.

Suppose that *protocol.fsm* is an input file containing a protocol's FSM. Then the following command runs the package:

rcpt [-cecom/-plain] protocol.fsm output_file

where *plain* option refers to a plain input/output file format. *protocol.fsm* file in plain format can be prepared manually. The *cecom* option selects test generation in the CECOM format. In this case, the input file *protocol.fsm* should be generated by the *efsm2fsm* package. The tests are stored in the number of files named *protocol.i*, where i is the index of a test group.

B. INDEEL: Software for Inconsistency Detection and Elimination

Feasible test sequence generation is essential for assuring the proper operation and interoperability of different components in computer and communication systems. The use of formal description languages such as VHDL and Estelle enable the precise description of such systems to minimize the implementation errors due to misinterpretations. However, the specifications written in VHDL and Estelle are often extended finite-state machines (EFSMs), making the automated test generation a more complex task due to the inconsistencies among the action and condition variables [19].

Within ATIRP, we studied the problem of generating feasible test sequences for the EFSM by analyzing the interdependencies among the action and condition variables of the EFSM models. In the earlier phases of this research, *action* and *condition* inconsistencies in the EFSM models were defined [75], [76]. It has been shown that once the inconsistencies are eliminated, the existing finite-state machine (FSM)-based test generation methods can be used to generate feasible test sequences from the resulting consistent EFSM graphs.

The basic concepts for the inconsistency elimination algorithms were outlined in [76], which were later were generalized to include graphs with loops [71]. The formal descriptions of the inconsistency detection and elimination algorithms have been given in [19].

A software package, called INDEEL (INconsistencies DEtection and ELimination), has been implemented at CCNY based on the inconsistency elimination algorithms. As part of the ongoing collaboration between the CCNY and the UD, the application of these algorithms was extended to generate test sequences for the protocols with conflicting timers such as 188-220.

INDEEL contains 13,000+ lines of C code. As its input, the software reads a user specified file containing the description of an EFSM graph with the following properties:

- The specification consists of a single process and thus there are no communicating EFSMs.
- If the specification contains function calls, they can be described within the process with a simple transformation.
- Pointers, recursive functions, and syntactically endless loops are assumed not to be present in the specification.
- All conditions and actions are linear.

INDEEL uses an iterative approach: every time an action or condition inconsistency is detected and eliminated, an intermediate output graph is generated in a file, using the same format as in the input file. This intermediate output file then becomes the new input file to INDEEL for continued analysis. This iterative procedure is repeated until the graph becomes free of inconsistencies. The intermediate and the final output graphs are provided as files.

INDEEL starts its analysis by considering the action inconsistencies; it then proceeds to the detection and elimination of the condition inconsistencies (if any). During the analysis of the action inconsistencies, INDEEL constructs a set of *Action Update Matrix (AUM) pairs* for each node. The AUM pairs represent the effects of the actions of the traversed edges leading to a given node v_i. Similarly, the accumulated different conditions of the paths leading to v_i can be represented as a set of *Accumulated Condition Matrix* (ACM) triplets containing the coefficients, operators, and constants of the edge conditions.

To reduce the space complexity, during the AUM and ACM constructions, the software uses a single matrix called *path_matrices* in which the numbers of the edges in the paths from the initial node to v_i are stored.

INDEEL implements a two-phase modified breadth-first graph traversal, called P1-MBF and P2-MBF, to handle the detection of the action inconsistencies. P1-MBF is the main graph traversal from which P2-MBF may be invoked multiple times. During the condition inconsistency detection phase, the graph is traversed in a regular depth-first (DF) manner.

The software: *efsm2fsm-rcpt*, and (2) INDEEL were used to help generate tests for 188-220 that have been delivered to CECOM. This technology transfer is described in the next section.

VIII. ATIRP to CECOM Technology Transfer Results

Using research results from Section VI, and software as described in Section VII-B, UD and CCNY collaborated with CECOM to generate 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 sixteen 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 sixteen destinations, each of which may be free or busy. In general behavior tests, destinations are allowed to transit between free or busy mode, but the sender is restricted to communicate with at most two of them. In multi-destination tests, the sender communicates with up to sixteen destinations, which are forced to remain in free mode at all times.

Each expanded EFSM was then used in automated test generation. Table III 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 16 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 VI-B 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.

In 1998, 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 III. 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 have been delivered to CECOM.

In the final phase of ATIRP, we have been investigating test generation for 188-220 Class B which includes Types 1,3, and 2 service. Class B is much more complex than Classes A and C, and involves generating problems for reliable connection-oriented service test case generation.

The implementations of 188-220 from several manufacturers are 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.

IX. Conclusions: Improvements to Protocol Development Process

A. 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 17). It combines 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 and efficient. Due to the structured nature of Estelle, the specification process progressed at an accelerated pace compared to the other standards. 188-220 was completed on time, setting a rare example in the 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 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].

B. 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 V-B. 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 several 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 of a document written in English.

C. 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 188-220.

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 [29], [30] and UML [54], 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 [60]. 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 III, 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], [7], [9], [50], [57], [59], [63], [66], 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 [45]. They can range from Postman tours [1] or fault-oriented tests [78], [80] for mid-size protocols when the number of states ranges on the order of thousands, to guided random walk approaches [43], [81] for larger protocols when the number of states ranges in the tens of thousands.

The state explosion problem has been a major issue for generating FSM models out of EFSM representations of protocols [15], [56], [79], [80]. One common procedure for converting EFSMs into FSMs simultaneously performs reachability analysis and online minimization [15], [44]; this conversion is based on combining *equivalent states* [58] using *bisimulation equivalence* [51]. Another approach proposes the elimination of inconsistencies in EFSM models [69], [70]. 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 testers, 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.

X. Acknowledgments

The authors thank Samuel Chamberlain of ARL; Ted Dzik and Ray Menell of CECOM; and Mike McMahon and Brian Kind of ARINC, Inc. for their collaboration in this research.

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. *Proc. IEEE 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. [12], 55–58.

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

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

[6] J. Bi and J. Wu. Application of a TTCN-based conformance test environment to the Internet email protocol. In Kim et al. [41], 324–330.

[7] 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.

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

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

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

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

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

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

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

[15] K. T. Cheng and A. S. Krishnakumar. Automatic generation of functional vectors using the extended finite state machine model. *ACM Trans. Design Automation of Electronic Syst.*,

1(1):57–79, Jan. 1996.

[16] S.-K. Cheong, K.-H. Lee, and T.-W. Jeong. The analysis of integrating test results for ATM switching systems. In Baumgarten et al. [4], 83–89.

[17] 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. [4], 316–330.

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

[19] A. Duale and U. Uyar, Generation of feasible test sequences for EFSM models. In H. Ural, R. Probert, and G. v. Bochmann, eds, *Proc. IFIP Int'l Conf. Testing o Communicating Systems, TestCom*, Ottawa, Sept. 2000, 91-109.

[20] T. Dzik and M. McMahon. MIL-STD 188-220A evolution: A model for technical architecture standards development. *Proc. IEEE 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. *Proc. IEEE 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. *Proc. IEEE MILCOM*, Atlantic City, NJ, Oct. 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. [10], 111–126.

[24] M. A. Fecko, M. U. Uyar, A. S. Sethi, and P. D. Amer. Conformance testing in systems with semicontrollable interfaces. *Annals of Telecommun.*, 55(1):70–83, Jan. 2000.

[25] M. A. Fecko. Timing and control issues in conformance testing of protocols PhD Dissertation, CISC Dept., Univ. of Delaware, 1999.

[26] M. A. Fecko, M. U. Uyar, A. Y. Duale, and P. D. Amer. Test generation in the presence of conflicting timers. In H. Ural, R. Probert, and G. v. Bochmann, eds, *Proc. IFIP Int'l Conf. Testing o Communicating Systems, TestCom*, Ottawa, Sept. 2000.

[27] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Trans. Software Eng.*, 17(6):591–603, Jun. 1991

[28] R. Gecse. Conformance testing methodology of Internet protocols: Internet application-layer protocol testing—HTTP. In Petrenko and Yevtushenko [55], 35–48.

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

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

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

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

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

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

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

[36] ISO/IEC. *International Standard ISO/IEC 8802-2, ANSI/IEEE Std. 802.2*, 2nd edition, Dec. 1994.

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

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

[39] 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. *Proc. IFIP Int'l Workshop Test. Communicat. Syst. (IWTCS)*, Budapest, Hungary, Sept. 1999.

[40] T. Kato, T. Ogishi, A. Idoue, and K. Suzuki. Intelligent protocol analyzer with TCP behav-

ior emulation for interoperability testing of TCP/IP protocols. *Proc. IFIP Joint Int'l Conf. FORTE/PSTV*, 449–464, Osaka, Japan, Nov. 1997.

[41] M. Kim, S. Kang, and K. Hong, editors. *Proc. IFIP Int'l Workshop Test. Communicat. Syst. (IWTCS)*, Cheju Island, Korea, Sept. 1997. Boston, MA.

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

[43] 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.

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

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

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

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

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

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

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

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

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

[53] C. Negulescu and E. Borcoci. SSCOP protocol throughput evaluation—simulation based on Estelle specification. In Budkowski et al. [12], 75–98.

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

[55] A. Petrenko and N. Yevtushenko, editors. *Proc. IFIP Int'l Workshop Test. Communicat. Syst. (IWTCS)*, Tomsk, Russia, Sept. 1998. Boston, MA: Kluwer Academic Publishers.

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

[57] J. Romijn and J. Springintveld. Exploiting symmetry in protocol testing. In Budkowski et al. [10], 337–351.

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

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

[60] 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. [10], 37–55.

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

[62] J. Thees. Protocol implementation with Estelle—from prototypes to efficient implementations. In Budkowski et al. [12], 187–193.

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

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

[65] 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.

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

[67] H. Ural. Formal methods for test sequence generation. *Computer Communications*, 15(5):311-325, Jun. 1992.

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

[69] M. U. Uyar and A. Y. Duale. Modeling VHDL specifications as consistent EFSMs. *Proc. IEEE MILCOM*, Monterey, CA, Nov. 1997.

[70] M. U. Uyar and A. Y. Duale. Removal of inconsistencies in VHDL specifications. *Proc. US Army Research Lab ATIRP Conf.*, College Park, MD, Feb. 1998.

[71] M. U. Uyar and A. Y. Duale. Conformance tests for Army communication protocols. *Proc. US Army Research Lab ATIRP Conf.*, College Park, MD, Mar. 2000.

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

[73] 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.

[74] 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.

[75] M. U. Uyar and A. Y. Duale, "Modeling VHDL Specifications as Consistent EFSMs," *Proc. IEEE MILCOM*, Monterey, CA, Oct. 1997, 740-744.

[76] M. U. Uyar and A. Y. Duale. Resolving inconsistencies in VHDL Specifications. *Proc. IEEE MILCOM*, Atlantic City, NJ, Oct. 1999, No. 5.1.3.

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

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

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

[80] C. J. Wang and M. T. Liu. Generating test cases for EFSM with given fault models. *Proc. IEEE INFOCOM*, 774–781, 1993.

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

[82] J. Wytrębowicz and P. Roliński. Analysis tools for Estelle specifications. In Budkowski et al. [12], 141–155.

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

[84] S. Yoo, L. Collica, and M. Kim. Conformance testing of ATM Adaptation Layer protocol. In Baumgarten et al. [4], 237–252.

TABLE I

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	$LT?x_{10}$	$LT!y_{10}$

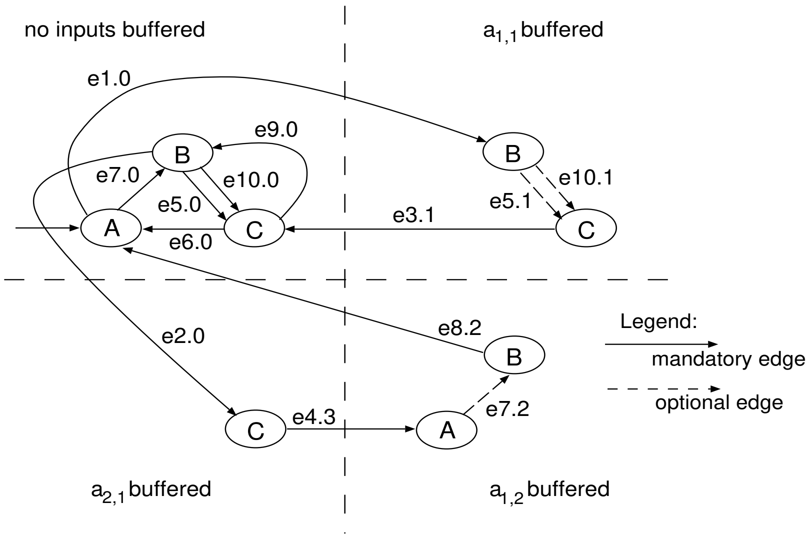


Fig. 14. Graph transformation applied to the graph of Fig. 13. Mandatory and optional edges appear in solid and dashed lines, respectively.

TABLE II

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	$LT?x_6$	$LT!y_6$

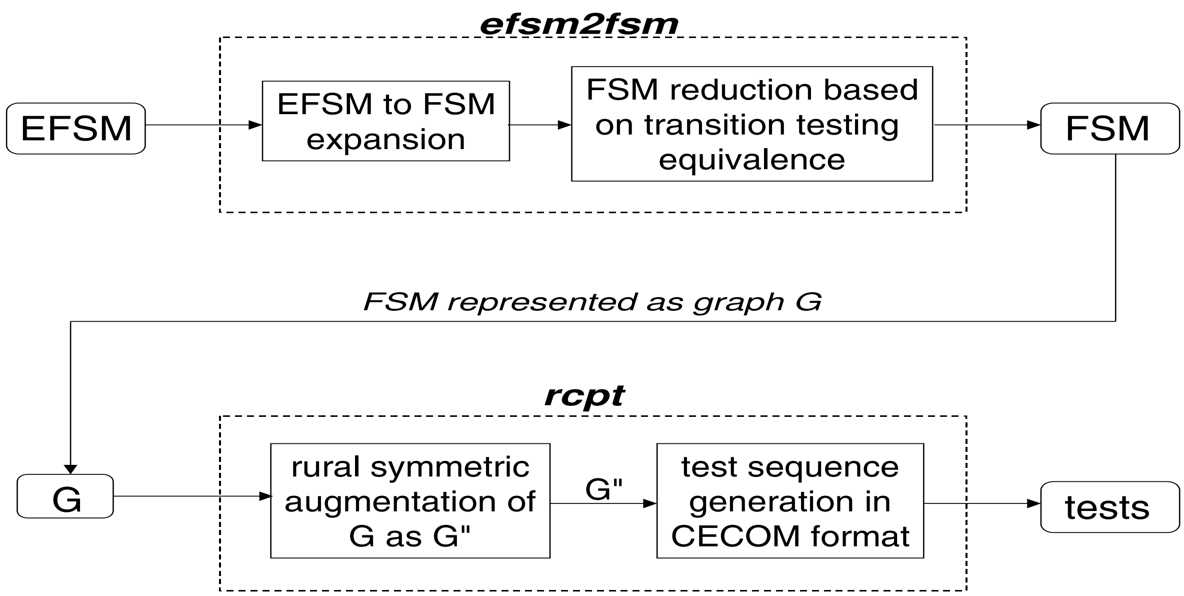


Fig. 15. Software for automated test generation.

TABLE III
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

```
//                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. 16. A sample of test scripts delivered to CECOM.

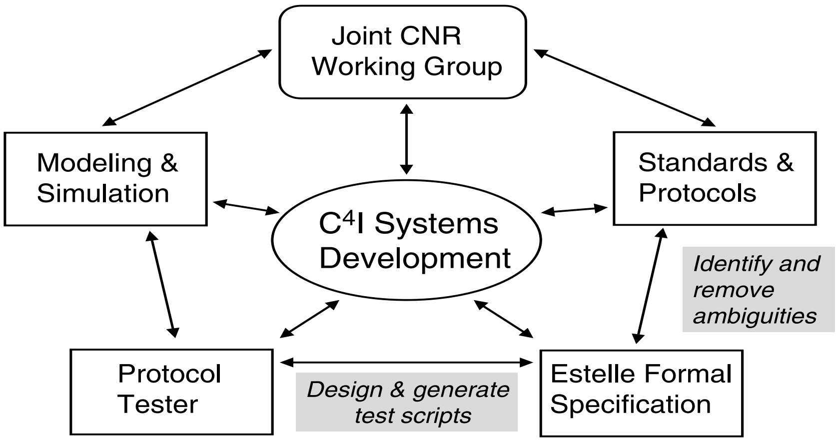


Fig. 17. Estelle as part of synergistic efforts to develop C^4I systems.