

CONFORMANCE TESTING: TOWARDS REFINING VHDL SPECIFICATIONS*

Ali Y. Duale[†]

M. Ümit Uyar

Dept. of Electrical Engineering
The City College of the
City University of New York
New York, NY 10031
umit@ee-mail.engr.cuny.cuny.edu

Bruce D. McClure^{†§}

[†]Defence Science and Technology Org.
PO Box 1500
Salisbury, SA 5108, Australia
[§]Dept. of Computer Science
and Electrical Engineering
The University of Queensland
Brisbane, Qld 4072, Australia
Bruce.McClure@dsto.defence.gov.au

Sam Chamberlain

US Army Research Laboratory
AMSRL-IS-TP
Aberdeen Proving Ground
MD 21005
wildman@arl.mil

ABSTRACT

Unexpected failures of mission-critical communication systems used in the military necessitate the integration of conformance testing with the protocol design process. Such an integration will allow for the removal of costly mistakes from a specification at an early stage of the development process and will enhance the confidence in systems. This paper demonstrates how a close relationship between the protocol specification and conformance test sequence generation enables a more rigorously tested product.

The study presented in this paper applies the algorithms developed earlier for the detection and removal of inconsistencies in VHDL specifications to the Local Proxy component of the Adaptive Computing Architecture (ACA) specification. The ACA, a military-oriented network architecture prototype, is designed to manage the ever-changing defense network resources according to pre-defined network policies. Based on the results of the study, we recommend modifying the design to remove redundancies, uncover missing actions, and reorganize portions of the specification to improve its testability.

Keywords: interoperability, military communication systems, networking, adaptive computing architecture, conformance testing, VHDL.

1. INTRODUCTION

Despite advances in formal methods of specification and technology, there is no guarantee that a complex communication or a digital system is implemented according to its specification. To increase the confidence and detect errors in mission-critical systems used by the military, the systems must be tested rigorously before they are deployed as services. In conformance testing, which is the most formidable task among the existing test methodologies, discrepancies between an implementation and its specification are detected.

The US Department of Defence adopted VHDL (Very High

*Prepared through collaborative participation in the Advanced Telecommunications & Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-000-2.

[†]This paper was prepared during the summer of 1998 while Ali Duale and Bruce McClure were at the ARL, Aberdeen, MD, as part of the ATIRP staff rotation program and as an exchange, respectively.

Speed Integrated Circuit Hardware Description Language) as the only acceptable specification language for the department's digital ASIC systems. VHDL is widely used to model digital systems at different levels of abstraction ranging from the algorithmic level to the gate level. Recently the language has been also used to model communication protocols [4, 5].

Due to the use of internal variables, the external behavior of protocols specified in VHDL are often modeled as Extended Finite-State Machines (EFSMs), finite-state machines (FSMs) combined with internal variables. Since certain paths of the EFSM model may require conflicting conditions, generating feasible test sequences for an EFSM model is quite challenging.

If a VHDL specification is modeled as a *consistent* EFSM, FSM-based test generation methods can be directly applied to generate minimum-length test sequences. An EFSM is consistent if it is free of condition-to-condition, action-to-condition, and action-to-action inconsistencies. An input set that satisfies a condition may become invalid for subsequent condition(s) of the same path, causing condition-to-condition inconsistency [8]. Furthermore, variables that are updated differently by actions of the specification may create action-to-condition and/or action-to-action inconsistencies for condition(s) and/or action(s) that occur later in the specification, respectively. Algorithms for the detection and removal of inconsistencies in VHDL models have been developed earlier in this research [8, 9].

This paper illustrates the importance of integrating protocol specification and conformance test sequence generation to enable the detection of costly errors at the design level before they are implemented. The VHDL specification of the Local Proxy component of the Adaptive Computing Architecture (ACA), a military-oriented network architecture prototype [2], is used as a case study. The ACA is designed to manage the ever-changing communication network resources according to pre-defined network policies (e.g, higher priorities are given to applications belonging to high ranking commanders).

By integrating conformance test generation and protocol design of the Local Proxy of the ACA, various improvements/recommendations have been submitted to the protocol de-

signers, such as the identification of various missing actions, the removal of redundancies, and the re-organization of portions of the ACA specification to enhance its testability. Similar case studies of military communication protocols are currently being considered.

The ACA and test generation for the Local Proxy are discussed in Section 2. Section 3 presents the improvements for the Local Proxy. Conclusions are given in Section 4.

2. THE LOCAL PROXY OF THE ADAPTIVE COMPUTING ARCHITECTURE

The requirements of defence networking and computing present significant challenges to current architectures for distributed computing. In particular, the mix of distributed computing, networks which provide variable bandwidth and reliability and mission critical applications which must use these networks demands new application architectures. The Adaptive Computing Architecture (ACA) has been proposed as a framework based on ODP bindings that supports policy-based adaptive management of network resources [2]. The ACA maintains three key kinds of information about the system: adaptive management policies, interface specifications (with QoS requirements) and network topology and QoS measures.

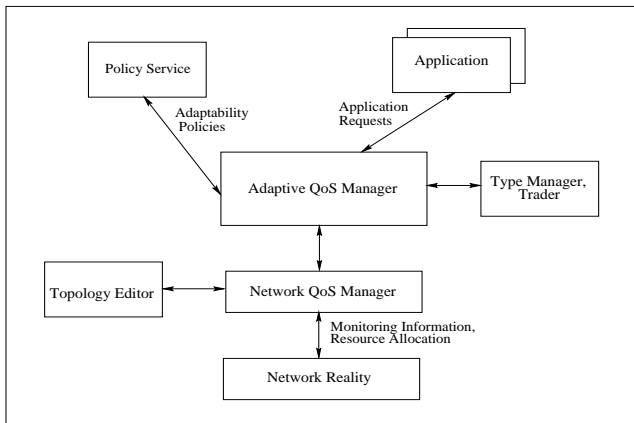


Figure 1: Adaptive Computing Architecture

Figure 1 is a high-level logical depiction of the ACA. The **Adaptive QoS Manager** establishes and manages bindings between application and service objects. The AQM is responsible for managing enterprise resource usage according to the current policies by mediating new requests for service and actively managing resource usage by existing bindings. If network resources are not available (and policy dictates), the AQM can pre-emptively shut down or adjust an existing binding to optimize its resource usage. Similarly, network load can be reduced by inserting filter objects within a binding. The **Policy Service** stores the adaptation policies that specify *how* and *when* the AQM should adapt to changing resource availability. Adaptation policies typically fall into one of three categories: i) usage preferences that are specific to a particular kind of application, ii) policies that address the trade-off of cost versus performance, iii) policies that are modal and/or sensitive to user identity. The **Network QoS**

Manager holds knowledge of the state of the network, consisting of two distinct parts. The nominal network topology and its associated QoS measures are expressed using the Network QoS Specification Language (NQSL). The second part deals with the current network state. The NQM must be informed of the current network load by management interfaces on critical links and gateways. The NQM also uses the information that it gathers to perform route selections and preemption on the basis of required QoS parameters. The **Trader** and **Type Manager** are supporting ODP services.

In order to validate the ACA it is desirable to test it on a larger network than that used by our initial prototype. DSTO is building an Experimental C3I Technology Environment (EXC3ITE). Therefore, we have chosen to model the EXC3ITE network as a realistic exemplar heterogeneous defence network to support a simulation of the ACA. The simulation model of the ACA represents an *implementation* of the architecture. The AQM is implemented as two distributed components. The first part is a local component that is present on all workstations. This part of the AQM implements policies local to the workstation and provides an interface between applications and the architecture. Second, aspects of the AQM which have wider impact form part of a LAN component. The local component is modeled as two related processes: Local Proxy and Local Proxy2. The main function provided by the Local Proxy process is to manage a set of Local Proxy2 processes which in turn re-direct application requests to the LAN Proxy component and performs local monitoring for the application.

The behavioral model of a VHDL specification can be used as a formal description for a communication protocol [4, 5]. The behavioral model of the Local Proxy component has been specified in VHDL, where the architecture is represented in a single normalized process. An EFSM representation of the Local Proxy is constructed from its VHDL behavior description (since internal variables are used in the specification, a simpler FSM model could not be utilized). The main functionality of the local Proxy component is depicted when the component is in its *listen* mode. Due to space constraints of this paper, only a portion of the EFSM model of the Local Proxy, that portrays the component's *listen* mode, is presented in Figure 3. In the specification, when the *else* part of an *if* statement is missing, a "complementary" *else* statement with a *null* output is created. The complementary edges of Figure 3 include e_{26} , e_{28} , e_{36} , and e_{42} .

2.1. Conformance Testing

The Local Proxy is formally specified in VHDL, which is generally modeled as an EFSM. During the process of generating the conformance test sequences, the existence of inconsistencies among actions and conditions of the EFSM model of the Local Proxy are checked. Such inconsistencies solely indicate that some of the graph edges representing the EFSM cannot be traversed together with certain other edges. Therefore, inconsistencies in an EFSM model do not imply errors in the specification. The inconsistency detection algorithm reported in [8], which uses symbolic execution [6] and graph splitting

techniques, is applied to the EFSM model of the Local Proxy.

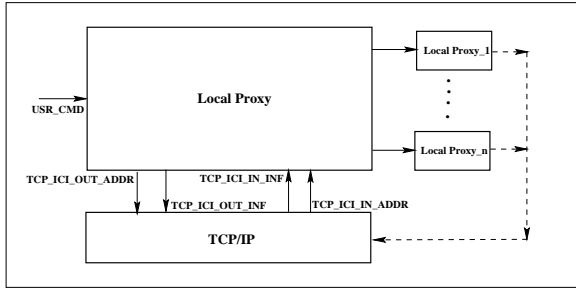


Figure 2: The Local Proxy of ACA

To determine if an EFSM model of a VHDL specification is consistent, action-to-action and action-to-condition inconsistencies are first eliminated. In a breadth-first manner, variables used for the conditions and actions for the outgoing edges of a node, s_i , are updated to their current symbolic values. If at least one of these variables is updated differently in the path(s) from the initial node to s_i , all the nodes accessible from s_i are split into parallel nodes. The next step is to detect condition-condition inconsistency. In a depth-first manner, the total predicate of the path leading to node s_i is accumulated. If the path predicate is not feasible, the EFSM is *inconsistent*. Otherwise the detection is continued in the depth-first manner.

The EFSM model of the Local Proxy is found to be *inconsistent*. For example, a test sequence with the edges e_{27} , e_{37} , and e_{43} requires $TCP_ICL_IN_INF_TYPE$ to be $ESTAB$, SG_FWD , and $CLOSE/ABORT$, respectively. Since the signal $TCP_ICL_IN_INF_TYPE$ is not updated in the walk containing these edges, executing such a test sequence is not feasible.

Once the inconsistencies are detected in the Local Proxy, they can be removed from the EFSM model by applying the algorithms reported in [9]. The first algorithm eliminates action-to-condition and action-to-action inconsistencies from the EFSM model. In a breadth-first manner, if the conditions and/or actions for the outgoing edges of s_i use variables modified differently in the paths leading to s_i , then s_i and all nodes accessible from it are split into parallel subgraphs. The number of times that a node is split is determined by the number of different values for the variables causing the inconsistencies at s_i . The condition-to-condition inconsistencies are removed by applying the second algorithm reported in [9]. During the removal, in a depth-first manner path conditions up to node s_i are accumulated. If the conditions for outgoing edges from s_i conflict with the accumulated path conditions, s_i and all nodes accessible from it are split into parallel subgraphs. The number of times that a node is split depends upon the number of condition sets for the variables of the outgoing edges of s_i .

During the node splits, a node that can be accessed from s_i is split only if the path between the two nodes does not contain any intermediate read condition(s) for the variable(s) that are causing inconsistencies. For example, in the Local Proxy, because of the "wait" statement in e_{49} (i.e., a "read" type of statement), the nodes accessible from s_{28} are not split. Figure

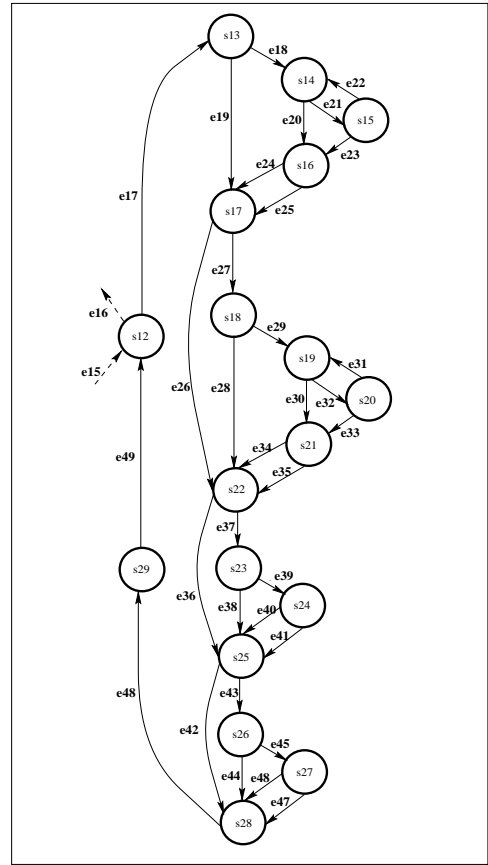


Figure 3: The EFSM Model of the Local Proxy

4 shows the final graph after inconsistencies are removed.

The conformance tests for the Local Proxy are generated by using the Rural Chinese Postman (RCP) method, which combines the rural postman tours and the unique input/output (UIO) sequences [1]. The RCP method is developed for FSM models and therefore does not address the issue of inconsistencies. However, after the inconsistencies are removed, the EFSM model of a specification effectively becomes an FSM model which then can be used with the RCP method. A minimum length test sequence generated by the RCP method for the Local Proxy consists of 185 steps. (During this early step of the case study, the test sequence is generated without using the UIO sequences.)

3. IMPROVEMENTS TO THE LOCAL PROXY SPECIFICATION

The integration of the conformance test generation for the Local Proxy with the protocol design improved both the testability and the completeness of the protocol. Through the conformance test generation for the Local Proxy, some redundancies were identified and eliminated from the specification, certain portions of the specification were restructured to improve the testability of the protocol, and missing actions were identified. The following sections highlight the observations regarding the testability of the Local Proxy.

3.1 Redundancies in Specifications

UIO sequences are planned to be used for the state verification during conformance testing. It has been shown that, if a

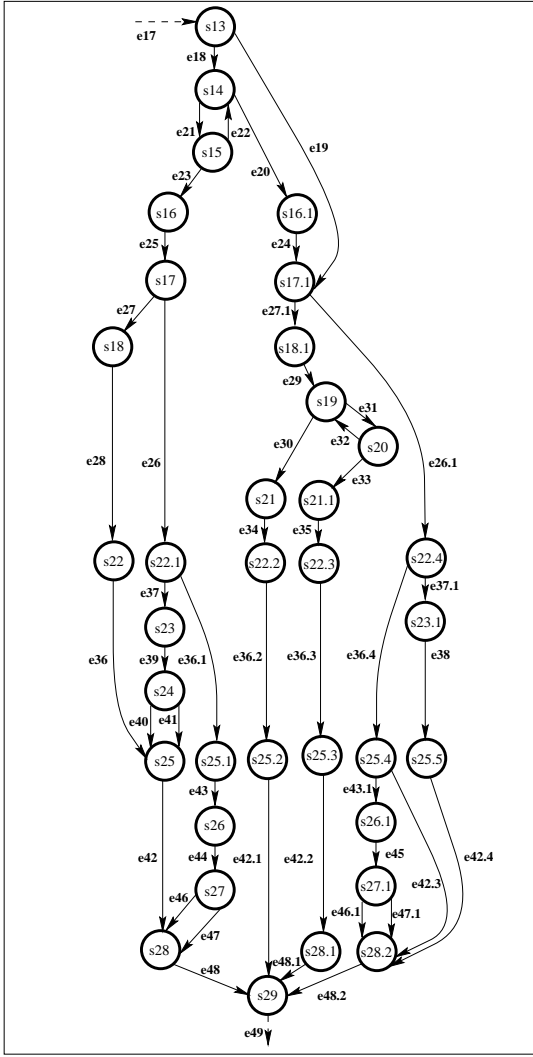


Figure 4: EFSM Model of the Local Proxy After Inconsistencies Were Removed

state of an FSM/EFSM does not have an equivalent state, it possesses a UIO sequence [7]. Two states are said to be equivalent if the permissible input set for one is a subset for the permissible input set of the other and the corresponding outputs and next states are the same [3]. Two states are also k -equivalent if the pair cannot be distinguished with a sequence of length k . The main reason that UIO sequences cannot be used for FSMs/EFSMs containing equivalent states is due to the inability to detect transfer errors. A transfer error on an edge from state s_i to s_j can cause the implementation under test (IUT) to move to a different, but equivalent, state from s_j . Failure to identify a state may allow for non-conformant IUTs to successfully pass the conformance tests.

To describe the protocol behavior precisely, a protocol designer may repeat certain portions of the specification several times. Such redundancies will eventually lead to the formation of equivalent states, impairing the testability of the IUT. Therefore, balancing the clarity of the specification with its testability is an important issue to be addressed at the design stage.

In the process of forming UIO sequences for the Local Proxy, it is observed that some of the states produce identical outputs for the same inputs, and their next states are 2 or 3-equivalent. For example, the input/output set for s_{20} is identical to that of s_{15} and the sets of their adjacent nodes ($\{s_{19}, s_{21}\}$ for s_{20} and $\{s_{14}, s_{16}\}$ for s_{15}) contain 2 or 3-equivalent nodes. Therefore, nodes $s_{19}, s_{20},$ and s_{21} can be merged with $s_{14}, s_{15},$ and $s_{16},$ respectively. Since the number of nodes and edges will be reduced after the merge, the length of test sequence will be also shorter.

For testing purposes, the behavior of an EFSM is transformed to an FSM by duplicating some of the nodes and edges of the EFSM graph [8, 9]. The number of states could dramatically increase if proper methods of expansion are not employed. The removal of the redundant states is also helpful in reducing the number of states in the resulting FSM graph.

3.2 Structure of EFSMs

Generally, a protocol can be specified in several different ways leading to different FSMs/EFSMs all of which accomplish the same task(s). The EFSM model of a VHDL specification consists of interconnected data flow subgraphs. In this section, the effects of the EFSM structure on the length of the test sequences and error detection are discussed. By rearranging the ending nodes for some edges of the graph representing the EFSM model of the Local Proxy reduces the length of the test sequences. Furthermore, for each *if* statement whose corresponding *else* statement was not present in the VHDL specification, a complementary edge, with a *null* output, was created and augmented to the graph representing the EFSM model of the Local Proxy. Some of these complementary edges later led to identification of missing actions that were mistakenly omitted.

Due to the inconsistencies among the actions and conditions [8], automated test generation for EFSM-modeled protocols becomes a difficult process. The graph of an EFSM model might be split multiple times to remove the inconsistencies, where each split node may increment the length of the test sequence. The complexity of splitting nodes and edges of the EFSM graph (and hence the length of test sequence) can be reduced at the design stage by excluding some of the inconsistencies from the specification.

<pre> if (x = 1) then A := b; else if (x = 2) then A := c; else null; end if; end if; (5.a) </pre>	<pre> if (x = 1) then A := b; else null; end if; if (x = 2) then A := c; else null; end if; (5.b) </pre>
---	--

Figure 5: Examples of Two VHDL Specifications

The topological interconnection among these subgraphs has a direct impact on the length of tests generated for the EFSM. A cascade of data flow subgraphs that use the same condi-

tional variables can be interconnected such that the number of infeasible paths are minimum (provided that the controlling variables are not updated in these data flow subgraphs). By minimizing the infeasible paths, the test sequence length can be shortened.

Figure 5 shows two fragments of different VHDL specifications, which are equivalent in terms of their functionalities. Their corresponding EFSM models, however, differ significantly. All paths for the EFSM model of Figure 5.a are valid and thus the EFSM is consistent. However, the EFSM model of Figure 5.b contains condition-to-condition inconsistency and an infeasible path. The EFSM model of Figure 5.b, therefore, requires mechanisms to remove the inconsistency, which increases the complexity of the test generation process.

A simple but important example that supports this case can be found in the Local Proxy. As stated in Section 3, it is not possible to include e_{27} , e_{37} , and e_{43} in the same test sequence. This problem will be resolved when the inconsistencies are removed from the EFSM model. The work of splitting some nodes, however, could have been prevented if the tail nodes for e_{28} , e_{34} , e_{35} , e_{38} , e_{40} , and e_{41} were combined as s_{28} . Such combination of tail states can be achieved by using the format depicted in Figure 5.a for conditions of outgoing edges of s_{17} , s_{22} , and s_{25} (which currently use the format of Figure 5.b).

Therefore, where possible, the conformance testers provide recommendations for the sections of the EFSM graph that requires heavy splitting due to inconsistencies. Restructuring the specification may reduce the complexity of the test generation process, while reducing the test sequence length (without trade-offs).

As indicated earlier, the EFSM model of the Local Proxy contains complementary edges such as e_{26} , e_{28} , and e_{36} . The actions of these complementary edges of Local Proxy are further discussed with the designers to clarify if they were inserted correctly. During these discussions, it is observed that some of the complementary edges were the only edges connecting some subgraphs to the rest of the EFSM graph. For example, if e_{26} and e_{28} (complementary edges) were removed from Figure 4, $s_{22.1}$ and s_{18} would become nodes without incoming and outgoing edges, respectively.

This observation revealed that certain actions were left unspecified in the Local Proxy. As an example, let us consider the actions of the complementary edge e_{28} . The messages received from TCP/IP are of three types: *connection established*, *segment received*, and *connection closed/aborted*. The Local Proxy takes appropriate actions dictated by the type of the message received from TCP/IP. After the action-to-action and the action-to-condition inconsistencies are removed, it must be true for all nodes of the subgraph starting s_{16} and ending s_{28} of Figure 4 that $conn_idx \neq NEGAT_1$. Thus, traversing the complementary edge of e_{28} , with $conn_idx \neq NEGAT_1$ input and *null* output, is feasible. However, discussions with the designers revealed that the unspecified actions were left out by mistake and e_{28} should not have been a complementary edge. The actions of e_{28} , corresponding to the

TCP/IP message signaling for *connection established* needed to be specified when $TCP_ICI_IN_INF_TYPE = ESTAB$ and $conn_idx \neq NEGAT_1$.

The modification made to the Local Proxy specification based on the reachability analysis demonstrates the need for integrating the protocol development with the conformance test generation process. This integration will allow for the removal of costly mistakes from the specification at an early stage of the development, before they propagate into many different implementations possibly combined with other errors.

4. CONCLUSIONS

This paper presents a case study to illustrate that the process of formally specifying a protocol can be combined with the process of conformance test generation, resulting in a more rigorously tested product. In the case study presented in this paper, the inconsistency detection and removal algorithms of [8, 9] with automated test generation techniques of [1] are applied to the VHDL specification of the Local Proxy component of the ACA [2]. Based on the results of the conformance test generation process [8, 9], various improvements/recommendations have been submitted to the Local Proxy designers, such as the identification of various missing actions, the removal of redundancies, and the re-organization of portions of the specification to enhance its testability. Several other VHDL specifications of protocols used in the military are planned to be studied within this integrated framework.

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. on Communications*, vol. 39, No. 11, pp. 1604-1615, Nov. 1991.
- [2] S. Crawley, J. Indulska, and B. McClure, "ODP-Based Adaptive Management of Network Resources in Heterogeneous Defence Networks," IEEE Workshop on Distributed Systems Operations and Management, Newark, DE., Oct. 1998.
- [3] R. J. Linn and M. U. Uyar, *Conformance Testing Methodologies and Architecture for OSI Protocols*, IEEE Computer Society, Los Alamitos, CA., 1994.
- [4] J. G. Gowens, B. Nguyen, and W. Butler, "An Experiment Using VHDL to Model and Simulate the ISDN LAPD Data Link Layer Protocol," Advanced Telecommunications/Information Distribution Research Program (ATIRP), pp. 163-167, College Park, MD., Feb. 1998.
- [5] B. Nguyen, "Using VHDL as an SDL," Advanced Telecommunications/Information Distribution Research Program (ATIRP), pp. 289-293, College Park, MD., Jan. 1997.
- [6] W. E. Howden, "Symbolic Testing and Dissect Evaluation System," *IEEE Trans. on Software Eng.*, Vol. SE-2, No. 4, pp. 266-278, July 1977.
- [7] K. K. Sabnani and A. T. Dahbura, "A Protocol Test Generation Procedure," *Computer Networks and ISDN Systems*, vol. 15, pp. 285-297, 1988.
- [8] M. U. Uyar and A. Y. Duale, "Modeling VHDL Specifications as Consistent EFSMs," Proc. of IEEE Military Comm. Conf. (MILCOM), pp. 740-744, Monterey, CA., Oct. 1997.
- [9] M. U. Uyar and A. Y. Duale, "Resolving Inconsistencies in EFSM-Modeled Specifications," Proc. of IEEE Military Comm. Conf. (MILCOM), Atlantic City, NJ., Oct. 1999.