ARMY RESEARCH LABORATORY

Refining VHDL Specifications Through Conformance Testing: Case Study of an Adaptive Computing Architecture

by Ali Y. Duale, Bruce D. McClure,
and M. Ümit Uyar

ARL-TR-2010 July 1999

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-2010 **July 1999**

Refining VHDL Specifications Through Conformance Testing: Case Study of an Adaptive Computing Architecture

Ali Y. Duale and M. Ümit Uyar
Electrical Engineering Department, City College of the City University of New York

Bruce D. McClure
Defence Science and Technology Organization, Salisbury, Australia

Abstract

The aim of this work is to enable a more rigorously tested product by integrating protocol specification and conformance test sequence generation. Such an integration will allow for the removal of costly mistakes from a specification at an early stage of the development process before they propagate into different implementations, possibly combined with other errors. This integrated approach has been applied to the VHDL specification of a military-oriented protocol prototype called the "Local Proxy." Based on the results of the conformance test generation process, the Local Proxy specification has been refined by uncovering various missing actions, removing redundancies, and restructuring the specification to improve its testability.

Acknowledgments

The authors wish to acknowledge with gratitude the assistance of Dr. Samuel Chamberlain in making possible the collaboration at the ARL that prduced the work reported in this document, and of Mr. George Hartwig and Mr. Frederick Brundick for their assistance with the preparation of this document.

INTENTIONALLY LEFT BLANK.

Table of Contents

INTENTIONALLY LEFT BLANK.

List of Figures

List of Tables

INTENTIONALLY LEFT BLANK.

1. Introduction

Today more than ever, society relies on systems built using complex software and hardware components. The military is no exception, relying on these technologies for almost every task from the foxhole to the command center. However, there have been many well-advertised failures of both software and hardware components in mission-critical systems. One approach for reducing this problem is increasing the effectiveness of conformance testing by combining it with the development process. The goal of conformance testing is to find discrepancies between a protocol's specification and its implementation. Recent advances in telecommunications systems development have shown the advantages of using conformance testing techniques as an integral part of the development process.

A protocol's external behavior is often modeled as an extended finite-state machine (EFSM)—a finite-state machine (FSM) that has access to storage variables. Due to the existence of context variables, timers, etc., automated conformance test generation for EFSMs is a challenging task.

Test generation methods available for the FSM models are only applicable to protocols modeled as consistent EFSMs [1]. An EFSM is said to be *consistent* if it is free of condition-to-condition, action-to-condition, and action-to-action inconsistencies. These inconsistencies occur when the variables of the actions and/or conditions are updated differently by the actions of the specification and/or when the same variable is used by more than one condition. (Note that these so-called inconsistencies in a specification are only impediments to the automated test generation techniques; their presence does not imply errors in the specification.) The Very high-speed integrated circuit Hardware Description Language (VHDL) is a hardware description language used for specifying the behaviour of hardware and software protocols. Algorithms for removing the inconsistencies from VHDL specifications were reported in [2].

The objective of this report is to propose the integration of protocol specification and conformance test sequence generation to enable a more rigorously tested product. Such an integration will allow for the removal of costly mistakes from a specification at an early stage of the development process, before they propagate into different implementations, possibly combining with other errors. In this report, the inconsistency detection and removal algorithms of [1, 2] are applied to generate test sequences for a military-oriented communication protocol prototype. The Adaptive Computing Architecture (ACA) [3], which takes into consideration managing the ever-changing defence network resources (according to predefined network policies), is used as a case study. The VHDL specification of the Local Proxy component of the ACA has been refined through conformance test generation.

Section 2 gives a description of the ACA, particularly the Local Proxy. Section 3 discusses generating a test sequence for the Local Proxy. Section 4 presents refinements for the Local Proxy. Section 5 draws our conclusions. The Appendix presents the VHDL specification for the Local Proxy.

2. Case Study: 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 that provide variable bandwidth and reliability, and mission-critical applications that must use these networks demands new approaches to building application architectures. The ACA has been proposed as a framework based on open distributed processing (ODP) bindings that supports policy-based adaptive management of network resources [3].

An adaptive network resource management framework needs to maintain three key kinds of information about the system. First, management policies need to be modelled to allow adjustments of policy at runtime. Second, the interface specifications and Quality of Service (QoS) requirements of applications and services need to be modelled so that the management system can support a dynamic environment of communicating objects. Finally, network topology and QoS measures need to be modelled so that the management system can make appropriate decisions based on network conditions. Figure 1 is a high-level logical depiction of the ACA. At the centre of the architecture is the Adaptive Quality of service Manager (AQM) that manages resources to satisfy application requirements according to current adaptation policies and network conditions.

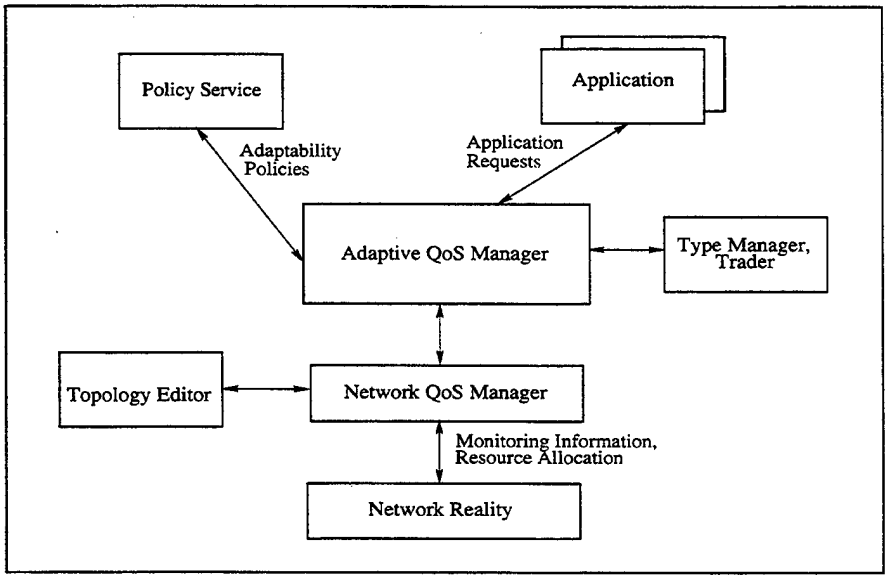

Figure 1. Adaptive Computing Architecture.

The Policy Service stores the current adaptation policies. The Network Quality of service Manager (NQM) maintains the system's model of the network topology, QoS, and the current state of the network. The Local Monitor (LM) performs monitoring of QoS delivered to applications. The Type Manager is an ODP service and is used to store interface and QoS descriptions for applications and services. The Trader is another ODP function that manages

current offers of service and matches them against requests from clients via the AQM.

2.1 Adaptive QoS Manager

The function of the AQM is to establish and manage bindings between application and service objects. While these bindings are made at the request of individual client objects, the AQM is responsible for managing enterprise resource usage according to the current policies. This is done by mediating new requests for service and actively managing resource usage by existing bindings. The emphasis of the AQM, its information model and actions, is on dealing with loss of service for critical applications—typically this involves managing heterogeneous wide-area network resources.

If network resources are not available (and policy dictates), the AQM may take action to manage resource usage. In the worst case, it can pre-emptively shut down an existing binding from another application to free up resources. The AQM can also adjust an existing binding to optimise its resource usage. For example, it could cause the binding to change the service objects and communications paths used to shed load from overloaded network links. Similarly, network load can be reduced by inserting filter objects within a binding.

2.2 Policy Service

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: (1) usage preferences that are specific to a particular kind of application, (2) policies that address the trade-off of cost vs. performance and (3) policies that are modal and/or sensitive to user identity. As an example of category (3), when a field unit is on high alert, only messages authorised by particular officers should be sent over the unit's high-frequency link. Our model for adaptation policies is based on Sloman's policy language [4].

2.3 Network QoS Manager

The NQM is the component in the adaptive resource management architecture that holds knowledge of the state of the network. This knowledge consists of two distinct parts. The nominal network topology and its associated QoS measures (e.g., link capacities) are expressed using the Network Quality of service Specification Language (NQSL) described in McClure et al. [5].

The second knowledge component maintained by the NQM 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 may also receive monitoring information from LMs regarding the QoS delivered to established bindings.

The other functions of the NQM include using the information that it gathers to perform

route selections and preemption on the basis of required QoS parameters. This function is used by the AQM to allow it to make optimal service selections if there are several servers available.

To validate the ACA it is desirable to test it on a larger network than that used by our inital prototype. The Australian Defence Science and Technology Organisation (DSTO) is currently building an Experimental C3I Technology Environment (EXC3ITE). This environment is an interstate heterogeneous network infrastructure including satellite links. The EXC3ITE network will support a number of emerging Common Object Request Broker Architecture (CORBA)-based applications. Potential users of the EXC3ITE network require method for moderating the use of network resources in this environment. Therefore the EXC3ITE network has been modelled as a realistic exemplar heterogeneous defence network to support this simulation.

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 Proxy, which together with the policy service 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 that have wider impact form part of a LAN proxy.

The main function provided by the Local Proxy process is to manage a set of Local Proxy2 processes, which in turn, redirect application requests to the Local Area Network (LAN) Proxy component and performs local monitoring for the application. The relationship between Local Proxy and Local Proxy2 processes is depicted in Figure 2.

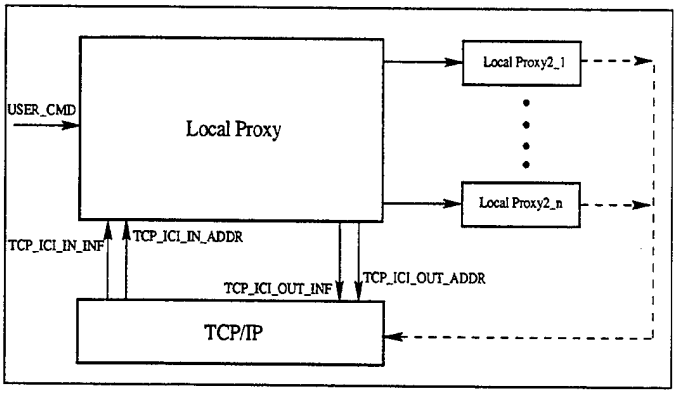

Figure 2. The Local Proxy of ACA.

The behavioral model of a VHDL specification can be used as a formal description for a communication protocol [6, 7]. The behavioral model of the Local Proxy component has been specified in VHDL, where the architecture is represented in a single 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. A portion of the EFSM model of the Local Proxy, that portrays the component's *listen* mode, is presented in Figure 3. For simplicity, the number of connections

is assumed to be, at most, two. 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}.

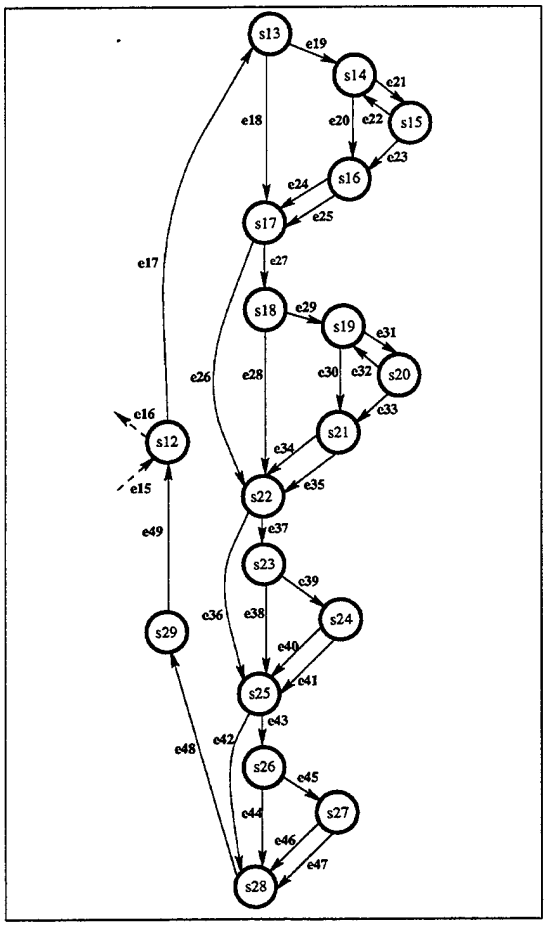

Figure 3. EFSM Model of the Local Proxy.

3. Test Generation for the Local Proxy

The VHDL specification for the Local Proxy of the ACA is an EFSM that can be used to automatically generate conformance tests. An EFSM model requires the detection and removal of the inconsistencies among the conditions and actions of its specification (if any) [1, 2].

The algorithm presented in Uyar and Duale [1] uses symbolic execution [8] with graph-splitting techniques to detect inconsistencies. Based on the variables used in the actions and conditions, infeasible paths are searched in the EFSM model. If at least one infeasible path is found, the EFSM is called *inconsistent*; otherwise the EFSM is *consistent*.

The EFSM model of the Local Proxy is found to be *inconsistent*. As mentioned earlier, finding inconsistencies in an EFSM model simply means that some edges in the graph

representation of the EFSM cannot be traversed together with certain other edges. For example, a test sequence with the edges e_{27}, e_{37}, and e_{43} requires *TCP_ICI_IN_INF_TYPE* to be *ESTAB*, *SG_FWD*, and *CLOSE/ABORT*, respectively. Executing a test sequence such as *TCP_ICI_IN_INF_TYPE* is not feasible because the input signal is not updated in the walk containing these edges.

Once the inconsistencies are detected in the Local Proxy, they can be removed from the EFSM model by the two algorithms reported in Uyar and Duale [2]. The first algorithm eliminates the action-to-condition and action-to-action inconsistencies from the EFSM model of the Local Proxy. If the conditions for outgoing edges of s_i use variables modified 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 second algorithm removes the condition-to-condition inconsistencies. During the removal, 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 there is a path between the two nodes, without 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 4 shows the final graph after infeasible edges were deleted, which is free of inconsistencies, obtained by the algorithms of Uyar and Duale [2].

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 [9]. The RCP method is developed for FSM models and, therefore, cannot address the issue of inconsistencies. However, after the inconsistencies are removed, the EFSM model of a specification effectively becomes an FSM model that can be used with the RCP method. A minimum-length test sequence, generated by the RCP method for the Local Proxy, consisting of 192 steps is shown in Table A-2, in the Appendix. (During this early step of the case study, the test sequence was generated without using the UIO sequences. A single test step was, however, dedicated for each state that needs to be verified.)

4. Refining the Local Proxy

During the initial phase of a protocol design, it is important that the external functionality of the protocol is well-defined, while the internal functionality is considered as a black box [10]. This hierarchical approach enables a successful process cycle of design, development and conformance testing. Within this framework, conformance test generation for the Local Proxy produced several specification changes. Various recommendations were presented to the protocol designers to enhance the completeness of the Local Proxy specification and improve its testability. The following sections highlight the observations regarding

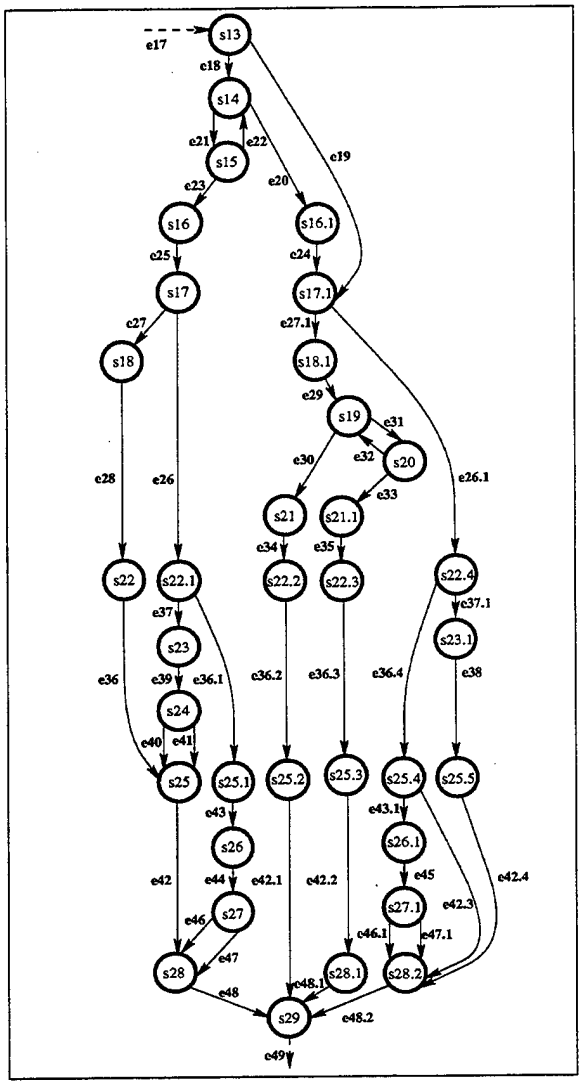

Figure 4. EFSM Model of the Local Proxy After Inconsistencies Were Removed.

the testability of Local Proxy.

4.1 Redundancies in Specifications

UIO sequences are planned to be used for the state verification during conformance testing. It has been shown that, if a state of an FSM/EFSM does not have an equivalent state, it possesses a UIO sequence [11]. 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 [12]. 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 nonconformant 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 [1, 2]. 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 final EFSM graph.

4.2 Reachability

Conformance testing methods require that each node of the directed graph of the FSM/EFSM model of the specification can be reached from any other node.

During the inconsistency removal process, the EFSM graph is split into subgraphs. Infeasible transitions are dropped from the new subgraphs. A node with no incoming edges becomes unreachable and is removed from the graph. The final graph with no inconsistencies, shown in Figure 4 for the Local Proxy, has no unreachable nodes.

As indicated in section 2, the EFSM model of the Local Proxy contains complementary edges (i.e., the edges, with *null* outputs, created for *if* statements whose corresponding *else* statements were not present in the VHDL specification) 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 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* were 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.

4.3 Structure of EFSMs

Due to the inconsistencies among the actions and conditions [1], automated test generation for EFSM modeled protocols becomes a difficult process. The graph of 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, without tradeoffs.

In general, 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. 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 conditional 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. However, their corresponding EFSM models differ significantly.

```
if (x = 1) then A := b;                 if (x = 1) then A := b;
else if (x = 2) then A := c;            else null;
else null;                              end if;
end if;                                 if (x = 2) then A := c;
end if;                                 else null;
                                        end if;
```

(a) (b)

Figure 5. Examples of Two VHDL Specifications.

All paths for the EFSM model of Figure 5(a) are valid and thus the EFSM is consistent. On the other hand, 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.

5. Conclusions

This report* proposes the integration of protocol specification and conformance test sequence generation to enable a more rigorously tested product. Through such an approach, specification errors can be detected at an early stage of the development process. As a case study, the inconsistency detection and removal algorithms of [1, 2] with automated test generation techniques of Aho et al. [9] are applied to the VHDL specification of the Local Proxy component of the ACA [3]. Based on the results of the conformance test generation process [1, 2], 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 reorganization 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.

*The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S Government.

6. References

1. M. U. Uyar and A. Y. Duale, "Modeling vhdl specifications as consistent efsms," in *MILCOM'97*, pp. 740–744, October 1997.
2. M. U. Uyar and A. Y. Duale, "Removal of inconsistencies in vhdl specifications," in *Advanced Telecommunications/Information Distribution Research Program (ATIRP)*, pp. 225–229, February 1998.
3. S. Crawley, J. Inulska, and B. McClure, "Opd-based adaptive management of network resources in heterogeneous defence networks," in *IEEE Workshop on Distributed Systems Operations and Management*, October 1998.
4. M. Sloman, "Management issues for distributed services," in *IEEE Second International Workshop on Services in Distributed and Networked Environments*, pp. 52–59, 1995.
5. B. McClure, J. Indulska, and S. Crawley, "Adaptive computing architecture for heterogeneous defence networks," in *University of Queensland Technical Report UQ-TR-429*, February 1998.
6. B. Nguyen, "Using vhdl as an sdl," in *Advanced Telecommunications/Information Distribution Research Program (ATIRP)*, pp. 289–293, January 1997.
7. J. G. Gowens, B. Nguyen, and W. Butler, "An experiment using vhdl to model and simulate the isdn lapd data link layer," in *Advanced Telecommunications/Information Distribution Research Program (ATIRP)*, pp. 163–167, February 1998.
8. W. E. Howden, "Symbolic testing and dissect evaluation system," *IEEE Trans. on Software Eng.*, vol. SE-2, pp. 266–278, July 1977.
9. 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, pp. 1604–1615, November 1991.
10. G. J. Holtzmann, *Design and Validation of Computer Protocols.* Prentice Hall, Englewood Cliffs, NJ, 1991.
11. K. K. Sabnani and A. T. Dahbura, "A protocol test generation procedure," *Computer Networks and ISDN Systems*, vol. 15, pp. 285–297, 1988.
12. R. J. Linn and M. U. Uyar, "Conformance testing methodologies and architecture for osi protocols," in *IEEE Computer Society*, 1994.

INTENTIONALLY LEFT BLANK.

Appendix:

VHDL Specification of the Local Proxy

INTENTIONALLY LEFT BLANK.

```
    library IEEE;


    use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use std.all;


entity LOCAL_PROXY is
    port (USER_CMD: in std_logic;
        TCP_ICI_OUT_INF: out std_logic_vector (0 to 15);
        TCP_ICI_OUT_ADDR: out std_logic_vector (0 to 63);
        TCP_ICI_IN_INF: in std_logic_vector (0 to 15);
        TCP_ICI_IN_ADDR: in std_logic_vector (0 to 63);
        CMD_OUT_ADDR: out std_logic_vector (0 to 63);
        CONNS_OUT_PORT: out std_logic_vector (0 to 31);
        CONNS_OUT_ADDR: out std_logic_vector (0 to 63);
        ACCESS_ADDR: out std_logic_vector (0 to 9);
        LP_MSG_OUT1, LP_MSG_OUT2, LP_MSG_OUT3, LP_MSG_OUT4,
        LP_MSG_OUT5, LP_MSG_OUT6, LP_MSG_OUT7, LP_MSG_OUT8, LP_MSG_OUT9,
        LP_MSG_OUT10: out std_logic_vector (0 to 15);
        TCP_ICI_PACKET_INDX: in integer);
end LOCAL_PROXY; architecture LOCAL_PROXY of LOCAL_PROXY is
begin
process
    constant MAX_CONN: INTEGER:= 10;
    constant TYPE_0 : INTEGER:= 0;
    constant TYPE_3 : INTEGER:= 3;
    constant CONN_ID_4: INTEGER:= 4;
    constant CONN_ID_8: INTEGER:= 8;
    constant STRM_INDX_9 :INTEGER:= 9;
    constant STRM_INDX_11 : INTEGER:= 11;
    constant NUM_PKS_12 : INTEGER:= 12;
    constant NUM_PKS_14 : INTEGER:= 14;
    constant port_block_0 : INTEGER:= 0;
    constant port_block_15 :INTEGER:= 15;
    constant URGENT_15 : INTEGER:= 15;
    constant REM_PORT_32 : INTEGER:= 32;
    constant REM_PORT_47 : INTEGER:= 47;
    constant REM_ADDR_48 : INTEGER:= 48;
    constant REM_ADDR_63 : INTEGER:= 63;
    constant LOCAL_PORT_0 :INTEGER:= 0;
    constant LOCAL_PORT_15 :INTEGER:= 15;
    constant LAST_SIGNAL_0: INTEGER:= 0;
    constant LAST_SIGNAL_3: INTEGER:= 3;
```

```
constant LAN_REM_PORT_32: INTEGER:= 32;
constant LAN_REM_PORT_47: INTEGER:= 47;
constant LAN_REM_ADDR_16: INTEGER:= 16;
constant LOCAL_INDEX_0: INTEGER:= 0;
constant LOCAL_INDEX_3: INTEGER:= 3;
constant LAN_REM_ADDR_31: INTEGER:= 31;
constant EMPTY : std_logic_vector (0 to 15):= "1111111111111111";
constant CONN_UNUSED: std_logic_vector (0 to 4):= "11111";
constant ZER_O : std_logic_vector (0 to 15):= "0000000000000000";
constant BEG_EXC : std_logic:= '0';
constant END_EXC : std_logic:= '1';
constant PKT_0: std_logic_vector (0 to 2):= "000";
constant PKT_1: std_logic_vector (0 to 2):= "001";
constant UNKNOWN_ID: std_logic_vector (0 to 4):= "00000";
constant NONE: std_logic_vector (0 to 2):= "000";
constant MSG_OPEN: std_logic_vector (0 to 3):= "0001";
constant LAN_RX: std_logic_vector (0 to 3):= "0010";
constant MSG_RX: std_logic_vector (0 to 3):= "0011";
constant MSG_CLOSE:std_logic_vector (0 to 3):= "0100";
constant CLOSE: std_logic_vector (0 to 3):= "0101";
constant ABORT:std_logic_vector (0 to 3):= "0110";
constant SEG_FWD:std_logic_vector (0 to 3):= "0111";
constant ESTAB:std_logic_vector (0 to 3):= "1111";
constant FIRST_LP:std_logic_vector(0 to 9):= "0000000001";
constant SECOND_LP:std_logic_vector(0 to 9):= "0000000010";
constant THIRD_LP:std_logic_vector(0 to 9):= "0000000100";
constant FOURTH_LP:std_logic_vector(0 to 9):= "0000001000";
constant FIFTH_LP:std_logic_vector(0 to 9):= "0000010000";
constant SIXTH_LP:std_logic_vector(0 to 9):= "0000100000";
constant SEVENTH_LP:std_logic_vector(0 to 9):= "0001000000";
constant EIGHTH_LP:std_logic_vector(0 to 9):= "0010000000";
constant NINETH_LP:std_logic_vector(0 to 9):= "0100000000";
constant TENTH_LP:std_logic_vector(0 to 9):= "1000000000";
constant NO_LP:std_logic_vector(0 to 9):= "0000000000";
constant CMD_OPEN_ACTIVE : std_logic_vector (0 to 3):= "0001";
constant CMD_OPEN_PASSIVE : std_logic_vector (0 to 3):="0010";
constant CMD_CLOSE : std_logic_vector (0 to 3):="0011";
constant CMD_ABORT: std_logic_vector (0 to 3):="0100";
constant CMD_RECEIVE: std_logic_vector (0 to 3):="0101";
subtype cmd_addr is std_logic_vector(0 to 63);
subtype cmd_inf is std_logic_vector(0 to 15);
subtype ind_addr is std_logic_vector(0 to 63);
subtype ind_inf is std_logic_vector(0 to 15);
subtype conn_in is std_logic_vector(0 to 15);
type conns_in_array is array (0 to MAX_CONN-1) of conn_in;
```

```
subtype conn_por is std_logic_vector(0 to 63);
type conns_por_array is array (0 to MAX_CONN-1) of conn_por;
subtype conn_add is std_logic_vector(0 to 63);
type conns_add_array is array (0 to MAX_CONN-1) of conn_add;
constant LAN_PROXY_PORT: std_logic_vector (0 to 15):= "0000100000000000";
constant PORT_BLOCK_START: std_logic_vector (0 to 15):= "0000100000000000" ;
constant LAN_PROXY_ADDRESS: std_logic_vector (0 to 15):= "0000001000000000";
constant LAN_IDX: INTEGER:= 0;
constant NEGAT_1: INTEGER:= -1;
constant SERVER_PORT: std_logic_vector (0 to 15):= "0000010000000000";
constant DELTA_1: std_logic_vector (0 to 15):= "0000000000000001";
- - Declare all the variables here.
variable conns_inf: conns_in_array;
variable conns_addr: conns_add_array;
variable conns_port: conns_por_array;
variable temp_inf: cmd_inf;
variable msg1: cmd_inf;
variable cmd_ici_inf: cmd_inf;
variable cmd_ici_addr: cmd_addr;
variable ind_ici_inf: cmd_inf;
variable ind_ici_addr: cmd_addr;
variable port_block: std_logic_vector (0 to 15);
variable i: INTEGER;
variable conn_id: std_logic_vector(0 to 4);
variable conn_idx: INTEGER;
variable temp_index: INTEGER;
variable LAN_PROXY_ADDR: std_logic_vector(0 to 15);
- - - - - - - - - - - - - - Actual process starts here - - - - - - - - - - - - - - - - - - -
begin
wait on USER_CMD;
if (USER_CMD = BEG_EXC) then
- - - - - - - - - - - - - - - - - Do initialization - - - - - - - - - - - - - - - - - - -
   temp_inf (CONN_ID_4 to CONN_ID_8):= CONN_UNUSED;
   for i in 0 to MAX_CONN loop
      conns_inf(i):= temp_inf;
   end loop;
   wait for 103000 ms;
   port_block:= PORT_BLOCK_START;
   cmd_ici_inf(TYPE_0 to TYPE_3):= CMD_OPEN_ACTIVE;
   cmd_ici_inf(CONN_ID_4 to CONN_ID_8):= UNKNOWN_ID;
   cmd_ici_inf(STRM_INDX_9 to STRM_INDX_11):= NONE;
   cmd_ici_inf(NUM_PKS_12 to NUM_PKS_14):= PKT_0;
   cmd_ici_inf(URGENT_15):= '0';
   cmd_ici_addr(port_block_0 to port_block_15):= port_block;
   cmd_ici_addr(REM_PORT_32 to REM_PORT_47):= LAN_PROXY_PORT;
```

```
cmd_ici_addr(REM_ADDR_48 to REM_ADDR_63):= LAN_PROXY_ADDR;
port_block:= port_block + DELTA_1;
cmd_ici_inf(NUM_PKS_12 to NUM_PKS_14):= PKT_0;
- - Now output the ICI to TCP
TCP_ICI_OUT_INF <= cmd_ici_inf;
TCP_ICI_OUT_ADDR <= cmd_ici_addr;
- - - - - - - - This is now the START state - - - - - - - - - - - - - - - - - -
wait on TCP_ICI_IN_INF, USER_CMD;
if (USER_CMD /= END_EXC) then
   while (TCP_ICI_IN_INF(TYPE_0 to TYPE_3) /= ESTAB) loop
      - - This is the LAN_OP state
      conn_id:= TCP_ICI_IN_INF(CONN_ID_4 to CONN_ID_8);
      temp_inf(CONN_ID_4 to CONN_ID_8):= TCP_ICI_IN_INF(CONN_ID_4 to CONN_ID_8);
      conn_idx:= LAN_IDX;
      temp_inf(STRM_INDX_9 to STRM_INDX_11):= NONE;
      conns_addr(conn_idx):= TCP_ICI_IN_ADDR;
      conns_port(conn_idx):= TCP_ICI_IN_ADDR;
      - - CONNS_OUT_PORT <= conns_port(conn_idx);
      CONNS_OUT_ADDR <= conns_addr(conn_idx);
      cmd_ici_inf:= TCP_ICI_IN_INF;
      cmd_ici_inf(NUM_PKS_12 to NUM_PKS_14):= PKT_1;
      cmd_ici_inf(TYPE_0 to TYPE_3):= CMD_RECEIVE;
      TCP_ICI_OUT_INF <= cmd_ici_inf;
      TCP_ICI_OUT_ADDR <= cmd_ici_addr; - - consist of ports and addrs
      CMD_OUT_ADDR <= cmd_ici_addr;
      conns_inf(conn_idx):= temp_inf;
      wait on TCP_ICI_IN_INF, USER_CMD;
   end loop;
   wait for 1000 ms;
   - - Then open passively for application clients
   - - This is state Pending
   cmd_ici_inf(CONN_ID_4 to CONN_ID_8):= UNKNOWN_ID;
   cmd_ici_addr(REM_PORT_32 to REM_PORT_47):= EMPTY;
   cmd_ici_addr(LOCAL_PORT_0 to LOCAL_PORT_15):= SERVER_PORT;
   cmd_ici_addr(REM_ADDR_48 to REM_ADDR_63):= ZER_O;
   cmd_ici_inf(TYPE_0 to TYPE_3):= CMD_OPEN_PASSIVE;
   cmd_ici_inf(NUM_PKS_12 to NUM_PKS_14):= PKT_0;
   TCP_ICI_OUT_INF <= cmd_ici_inf;
   TCP_ICI_OUT_ADDR <= cmd_ici_addr;
   CMD_OUT_ADDR <= cmd_ici_addr;
   - - Process anything from LAN or application
   - - This is the LISTEN state
   wait on USER_CMD, TCP_ICI_IN_INF;
   while (USER_CMD /= END_EXC) loop
      ind_ici_inf:= TCP_ICI_IN_INF;
```

```
ind_ici_addr:= TCP_ICI_IN_ADDR;
conn_id:= ind_ici_inf(CONN_ID_4 to CONN_ID_8);
- - Work out the conn_idx
if (conn_id = CONN_UNUSED) then
   conn_idx:= NEGAT_1;
else
   i:= 0 ;
   temp_inf:= conns_inf(i);
   while (i < MAX_CONN and
            temp_inf(CONN_ID_4 to CONN_ID_8) /= conn_id) loop
      i:= i + 1;
      temp_inf:= conns_inf(i);
   end loop;
   if (i < MAX_CONN) then
      conn_idx:= i;
   else
      conn_idx:= NEGAT_1;
   end if;
end if;
if (ind_ici_inf(TYPE_0 to TYPE_3) = ESTAB) then
   if (conn_idx = NEGAT_1) then
   - - - - - - - - - DoOpen - - - - - - - - - - - -
   - - find an empty spot in the connection array
      i:= 1; - - 0 reserved for LAN proxy
      temp_inf:= conns_inf(1);
      while (i < MAX_CONN and
            temp_inf(CONN_ID_4 to CONN_ID_8) /= CONN_UNUSED) loop
         i:= i+1;
         temp_inf:= conns_inf(i);
      end loop;
      if (i < MAX_CONN) then
         conn_idx:= i;
         conns_inf(conn_idx):= ind_ici_inf;
         conns_addr(conn_idx):= ind_ici_addr;
         conns_port(conn_idx):= conns_port(LAN_IDX); - - for LAN info.
         - - Do another passive open
         cmd_ici_inf(TYPE_0 to TYPE_3):= CMD_OPEN_PASSIVE;
         cmd_ici_inf(CONN_ID_4 to CONN_ID_8):= UNKNOWN_ID;
         cmd_ici_addr(LOCAL_PORT_0 to LOCAL_PORT_15):= SERVER_PORT;
         cmd_ici_addr(REM_PORT_32 to REM_PORT_47):= EMPTY;
         cmd_ici_addr(REM_ADDR_48 to REM_ADDR_63):= ZER_O; - - ie no address
         cmd_ici_inf(NUM_PKS_12 to NUM_PKS_14):= PKT_0;
         TCP_ICI_OUT_INF <= cmd_ici_inf;
         TCP_ICI_OUT_ADDR <= cmd_ici_addr;
         - - Now send the ICI to the Local Proxy2 process
```

```
            - - this is to avoid a spawn !!
            - - conns_inf(conn_idx, TYPE_0 to TYPE_3 ):= MSG_OPEN;
            temp_inf:= conns_inf(conn_idx);
            temp_inf(TYPE_0 to TYPE_3 ):= MSG_OPEN;
            conns_inf (conn_idx):= temp_inf;
            msg1:= conns_inf(conn_idx);
            case i is
               when 0 => LP_MSG_OUT1 <= msg1;
                     ACCESS_ADDR(0 to 9) <= FIRST_LP;
               when 1 => LP_MSG_OUT2 <= msg1;
                  ACCESS_ADDR(0 to 9) <= SECOND_LP;
               when 2 => LP_MSG_OUT3 <= msg1;
                  ACCESS_ADDR(0 to 9) <= THIRD_LP;
               when 3 => LP_MSG_OUT4 <= msg1;
                  ACCESS_ADDR(0 to 9) <= FOURTH_LP;
               when 4 => LP_MSG_OUT5 <= msg1;
                  ACCESS_ADDR(0 to 9) <= FIFTH_LP;
               when 5 => LP_MSG_OUT6 <= msg1;
                  ACCESS_ADDR(0 to 9) <= SIXTH_LP;
               when 6 => LP_MSG_OUT7 <= msg1;
                  ACCESS_ADDR(0 to 9) <= SEVENTH_LP;
               when 7 => LP_MSG_OUT8 <= msg1;
                  ACCESS_ADDR(0 to 9) <= EIGHTH_LP;
               when 8 => LP_MSG_OUT9 <= msg1;
                  ACCESS_ADDR(0 to 9) <= NINETH_LP;
               when 9 => LP_MSG_OUT10 <= msg1;
                  ACCESS_ADDR(0 to 9) <= TENTH_LP;
               when others =>
                  ACCESS_ADDR(0 to 9) <= NO_LP;
            end case;
         end if;
      end if;
   end if;
   if (ind_ici_inf(TYPE_0 to TYPE_3) = SEG_FWD) then
   - - - - - - - - - - - - - - - - - - - - - - DoSegFwd - - - - - - - - - - - - - - - - - - - - - -
   - - - Ask to read from socket.
      cmd_ici_inf(CONN_ID_4 to CONN_ID_8):= UNKNOWN_ID;
      cmd_ici_addr(REM_PORT_32 to REM_PORT_47):= EMPTY;
      cmd_ici_addr(REM_ADDR_48 to REM_ADDR_63):= ZER_O;
      cmd_ici_addr(LOCAL_PORT_0 to LOCAL_PORT_15):= SERVER_PORT;
      cmd_ici_inf(NUM_PKS_12 to NUM_PKS_14):= PKT_1;
      cmd_ici_inf(TYPE_0 to TYPE_3):= CMD_RECEIVE;
      TCP_ICI_OUT_INF <= cmd_ici_inf;
      TCP_ICI_OUT_ADDR <= cmd_ici_addr;
      if (conn_idx /= NEGAT_1) then
```

```
    - - connection is valid
        if (conn_idx = LAN_IDX) then
            temp_index:= TCP_ICI_PACKET_INDX;
            temp_inf:= conns_inf(temp_index);
            temp_inf(TYPE_0 to TYPE_3):= LAN_RX;
        else
            - - Received a packet from the application
            temp_inf:= conns_inf(conn_idx);
            temp_inf(TYPE_0 to TYPE_3):= MSG_RX;
        end if;
        - - Now send the ICI to the Local Proxy2 process
        - - this is a kludge to avoid a spawn !!
        conns_inf(conn_idx):= temp_inf;
        msg1:= conns_inf(conn_idx);
        case i is
            when 0 => LP_MSG_OUT1 <= msg1;
                ACCESS_ADDR(0 to 9) <= FIRST_LP;
            when 1 => LP_MSG_OUT2 <= msg1;
                ACCESS_ADDR(0 to 9) <= SECOND_LP;
            when 2 => LP_MSG_OUT3 <= msg1;
                ACCESS_ADDR(0 to 9) <= THIRD_LP;
            when 3 => LP_MSG_OUT4 <= msg1;
                ACCESS_ADDR(0 to 9) <= FOURTH_LP;
            when 4 => LP_MSG_OUT5 <= msg1;
                ACCESS_ADDR(0 to 9) <= FIFTH_LP;
            when 5 => LP_MSG_OUT6 <= msg1;
                ACCESS_ADDR(0 to 9) <= SIXTH_LP;
            when 6 => LP_MSG_OUT7 <= msg1;
                ACCESS_ADDR(0 to 9) <= SEVENTH_LP;
            when 7 => LP_MSG_OUT8 <= msg1;
                ACCESS_ADDR(0 to 9) <= EIGHTH_LP;
            when 8 => LP_MSG_OUT9 <= msg1;
                ACCESS_ADDR(0 to 9) <= NINETH_LP;
            when 9 => LP_MSG_OUT10 <= msg1;
                ACCESS_ADDR(0 to 9) <= TENTH_LP;
            when others =>
                ACCESS_ADDR(0 to 9) <= NO_LP;
        end case;
    end if;
end if;
if (ind_ici_inf (TYPE_0 to TYPE_3) = CLOSE or
        ind_ici_inf(TYPE_0 to TYPE_3) = ABORT) then
- - ICI_INT_TYPE = INT_CLOSE or INT_ABORT
- - - - - - - - - -DoClose - - - - - - - - - -
    if (conn_idx /= NEGAT_1) then
```

```
            - - connection is valid
               if (conn_idx /= LAN_IDX) then
                  temp_inf:= conns_inf(conn_idx);
                  temp_inf(TYPE_0 to TYPE_3):= MSG_CLOSE;
                  conns_inf(conn_idx):= temp_inf;
               end if;
               - - Now send the ICI to the Local Proxy2 process
               - - this is a kludge to avoid a spawn !!
               msg1:= conns_inf(conn_idx);
               case i is
                  when 0 => LP_MSG_OUT1 <= msg1;
                     ACCESS_ADDR(0 to 9) <= FIRST_LP;
                  when 1 => LP_MSG_OUT2 <= msg1;
                     ACCESS_ADDR(0 to 9) <= SECOND_LP;
                  when 2 => LP_MSG_OUT3 <= msg1;
                     ACCESS_ADDR(0 to 9) <= THIRD_LP;
                  when 3 => LP_MSG_OUT4 <= msg1;
                     ACCESS_ADDR(0 to 9) <= FOURTH_LP;
                  when 4 => LP_MSG_OUT5 <= msg1;
                     ACCESS_ADDR(0 to 9) <= FIFTH_LP;
                  when 5 => LP_MSG_OUT6 <= msg1;
                     ACCESS_ADDR(0 to 9) <= SIXTH_LP;
                  when 6 => LP_MSG_OUT7 <= msg1;
                     ACCESS_ADDR(0 to 9) <= SEVENTH_LP;
                  when 7 => LP_MSG_OUT8 <= msg1;
                     ACCESS_ADDR(0 to 9) <= EIGHTH_LP;
                  when 8 => LP_MSG_OUT9 <= msg1;
                     ACCESS_ADDR(0 to 9) <= NINETH_LP;
                  when 9 => LP_MSG_OUT10 <= msg1;
                     ACCESS_ADDR(0 to 9) <= TENTH_LP;
                  when others =>
                     ACCESS_ADDR(0 to 9) <= NO_LP;
               end case;
            end if;
         end if;
         wait on USER_CMD, TCP_ICI_IN_INF;
      end loop;
   end if;
end if;
end process;
end LOCAL_PROXY;
```

Table A-1. Edge Conditions for the Local Proxy

Edge	Condition
e_0	1 = 1
e_1	USER_CMD = USER_CMD_1
e_2	USER_CMD_1 ≠ BEG_EXC
e_3	USER_CMD_1 = BEG_EXC
e_4	i < MAX_CONN
$e_{4.1}$	1 = 1
e_5	i >= MAX_CONN
e_6	1 = 1
e_7	TCP_ICI_IN_INF = TCP_ICI_IN_INF_1 or USER_CMD = USER_CMD_2
e_8	USR_CMD = END_EXC
e_9	USR_CMD ≠ END_EXC
e_{10}	TCP_ICI_IN_INF_TYPE = ESTAB
e_{11}	TCP_ICI_IN_INF_TYPE ≠ ESTAB
e_{12}	1 = 1
e_{13}	TCP_ICI_IN_INF = TCP_ICI_IN_INF_2 or USER_CMD = USER_CMD_3
e_{14}	1 = 1
e_{15}	TCP_ICI_IN_INF = TCP_ICI_IN_INF_3 or USER_CMD = USER_CMD_4
e_{16}	USR_CMD = END_EXC
e_{17}	USR_CMD ≠ END_EXC
e_{18}	conn_id ≠ CONN_UNUSED
e_{19}	conn_id = CONN_UNUSED
e_{20}	i >= MAX_CONN
e_{21}	i < MAX_CONN
e_{22}	conns(i).conn_id ≠ cond_id
e_{23}	conns(i).conn_id = cond_id
e_{24}	i >= MAX_CONN
e_{25}	i < MAX_CONN
e_{26}	TCP_ICI_IN_INF_TYPE ≠ ESTAB
e_{27}	TCP_ICI_IN_INF_TYPE = ESTAB
e_{28}	conn_idx ≠ NEGAT_1
e_{29}	conn_idx = NEGAT_1
e_{30}	i >= MAX_CONN
e_{31}	i < MAX_CONN
e_{32}	conns(i).conn_id ≠ cond_id
e_{33}	conns(i).conn_id = cond_id
e_{34}	i >= MAX_CONN
e_{35}	i < MAX_CONN
e_{36}	TCP_ICI_IN_INF_TYPE ≠ SG_FWD
e_{37}	TCP_ICI_IN_INF_TYPE = SG_FWD
e_{38}	conn_idx = NEGAT_1
e_{39}	conn_idx ≠ NEGAT_1

Edge Conditions for the Local Proxy (Continued)

Edge	Condition
e_{40}	conn_idx $\neq$ LAN_IDX
e_{41}	conn_idx = LAN_IDX
e_{42}	TCP_ICI_IN_INF_TYPE $\neq$ CLOSE or ABORT
e_{43}	TCP_ICI_IN_INF_TYPE = CLOSE or ABORT
e_{44}	conn_idx $\neq$ NEGAT_1
e_{45}	conn_idx = NEGAT_1
e_{46}	conn_idx $\neq$ LAN_IDX
e_{47}	conn_idx = LAN_IDX
e_{48}	1 = 1
e_{49}	TCP_ICI_IN_INF = TCP_ICI_IN_INF_4 or USER_CMD = USER_CMD_5

Table A-2. Test Sequence for the Local Proxy

Step	From State	To State	Input/Output
1	N0	N1	I0/O1
2	N1	N2	I1/O3
3	N2	N2	verify N2
4	N2	N3	I3/O3
5	N3	N3	verify N3
6	N3	N3	I4/O4
7	N3	N4	I5/O5
8	N4	N4	verify N4
9	N4	N5	I6/O6
10	N5	N5	verify N5
11	N5	N6	I7/O7
12	N6	N6	verify N6
13	N6	N7	I9/O9
14	N7	N7	verify N7
15	N7	N8	I11/O11
16	N8	N8	verify N8
17	N8	N9	I12/O12
18	N9	N9	verify N9
19	N9	N7	I13/O13
20	N7	N10	I10/O10
21	N10	N10	verify N10
22	N10	N11	I14/O14
23	N11	N11	verify N11
24	N11	N12	I15/O15
25	N12	N12	verify N12
26	N12	N13	I17/O25
27	N13	N13	verify N13
28	N13	N14	I18/O18
29	N14	N15	I28/O28
30	N15	N15	verify N15
31	N15	N16	I23/O23
32	N16	N16	verify N16
33	N16	N17	I25/O25
34	N17	N17	verify N17
35	N17	N22.1	I26/O26
36	N22.1	N22.1	verify N22.1
37	N22.1	N25.1	I36.1/O36.1
38	N25.1	N25.1	verify N25.1
39	N25.1	N26	I43/O43
40	N26	N26	verify N26

Test Sequence for the Local Proxy (Continued)

Step	From State	To State	Input/Output
41	N26	N27	I44/O44
42	N27	N27	verify N27
43	N27	N28	I47/O47
44	N28	N28	verify N28
45	N28	N29	I48/O28
46	N29	N29	verify Ns29
47	N29	N29	verify N29
48	N29	N12	I49/O49
49	N12	N13	I17/O25
50	N13	N14	I18/O18
51	N14	N15	I28/O28
52	N15	N16	I23/O23
53	N16	N17	I25/O25
54	N17	N22.1	I26/O26
55	N22.1	N25.1	I36.1/O36.1
56	N25.1	N26	I43/O43
57	N26	N27	I44/O44
58	N27	N28	I46/O46
59	N28	N29	I48/O28
60	N29	N12	I49/O49
61	N12	N13	I17/O25
62	N13	N14	I18/O18
63	N14	N15	I28/O28
64	N15	N16	I23/O23
65	N16	N17	I25/O25
66	N17	N22.1	I26/O26
67	N22.1	N23	I37/O37
68	N23	N23	verify N23
69	N23	N24	I39/O39
70	N24	N24	verify N24
71	N24	N25	I41/O41
72	N25	N25	verify N25
73	N25	N28	I42/O42
74	N28	N29	I48/O28
75	N29	N12	I49/O49
76	N12	N13	I17/O25
77	N13	N14	I18/O18
78	N14	N15	I28/O28
79	N15	N16	I23/O23
80	N16	N17	I25/O25

Test Sequence for the Local Proxy (Continued)

Step	From State	To State	Input/Output
81	N17	N22.1	I26/O26
82	N22.1	N23	I37/O37
83	N23	N24	I39/O39
84	N24	N25	I40/O40
85	N25	N28	I42/O42
86	N28	N29	I48/O28
87	N29	N12	I49/O49
88	N12	N13	I17/O25
89	N13	N14	I18/O18
90	N14	N15	I28/O28
91	N15	N16	I23/O23
92	N16	N17	I25/O25
93	N17	N18	I27/O27
94	N18	N18	verify N18
95	N18	N22	I28/O28
96	N22	N22	verify N22
97	N22	N25	I36/O36
98	N25	N28	I42/O42
99	N28	N29	I48/O28
100	N29	N12	I49/O49
101	N12	N13	I17/O25
102	N13	N14	I18/O18
103	N14	N15	I28/O28
104	N15	N14	I22/O22
105	N14	N14	verify N14
106	N14	N16.1	I20/020
107	N16.1	N17.1	I24/O24
108	N17.1	N17.1	verify N17.1
109	N17.1	N18.1	I27.1/O27.1
110	N18.1	N18.1	verify N18.1
111	N18.1	N19	I29/O29
112	N19	N19	verify N19
113	N19	N20	I31/O31
114	N20	N20	verify N20
115	N20	N21.1	I33/O33
116	N21.1	N21.1	verify N21.1
117	N21.1	N22.3	I35/O35
118	N22.3	N22.3	verify N22.3
119	N22.3	N25.3	I36.3/O36.3
120	N25.3	N25.3	verify N25.3

Test Sequence for the Local Proxy (Continued)

Step	From State	To State	Input/Output
121	N25.3	N28.1	I42.2/O42.2
122	N28.1	N28.1	verify N28.1
123	N28.1	N29	I48.1/O48.1
124	N29	N12	I49/O49
125	N12	N13	I17/O25
126	N13	N17.1	I19/O19
127	N17.1	N18.1	I27.1/O27.1
128	N18.1	N19	I29/O29
129	N19	N20	I31/O31
130	N20	N19	I32/O32
131	N19	N21	I30/O30
132	N21	N21	verify N21
133	N21	N22.2	I34/O34
134	N22.2	N22.2	verifyN 22.2
135	N22.2	N25.2	I36.2/O36.2
136	N25.2	N25.2	verify N25.2
137	N25.2	N29	I42.1/O42.1
138	N29	N12	I49/O49
139	N12	N13	I17/O25
140	N13	N17.1	I19/O19
141	N17.1	N22.4	I26.1/O26.1
142	N22.4	N22.4	verify N22.4
143	N22.4	N23.1	I37.1/O37.1
144	N23.1	N23.1	verify N23.1
145	N23.1	N25.5	I38/O38
146	N25.5	N25.5	verify N25.5
147	N25.5	N28.2	I42.4/O42.4
148	N28.2	N28.2	verify N28.2
149	N28.2	N28.2	verify N28.2
150	N28.2	N28.2	verify N28.2
151	N28.2	N29	I48.2/O48.2
152	N29	N12	I49/O49
153	N12	N13	I17/O25
154	N13	N17.1	I19/O19
155	N17.1	N22.4	I26.1/O26.1
156	N22.4	N25.4	I36.4/O36.4
157	N25.4	N25.4	verify N25.4
158	N25.4	N26.1	I43.1/O43.1
159	N26.1	N26.1	verify N26.1
160	N26.1	N27.1	I45/O45

Test Sequence for the Local Proxy (Continued)

Step	From State	To State	Input/Output
161	N27.1	N27.1	verify N27.1
162	N27.1	N28.2	I47.1/O47.1
163	N28.2	N29	I48.2/O48.2
164	N29	N12	I49/O49
165	N12	N13	I17/O25
166	N13	N17.1	I19/O19
167	N17.1	N22.4	I26.1/O26.1
168	N22.4	N25.4	I36.4/O36.4
169	N25.4	N26.1	I43.1/O43.1
170	N26.1	N27.1	I45/O45
171	N27.1	N28.2	I46.1/O46.1
172	N28.2	N29	I48.2/O48.2
173	N29	N12	I49/O49
174	N12	N13	I17/O25
175	N13	N17.1	I19/O19
176	N17.1	N22.4	I26.1/O26.1
177	N22.4	N25.4	I36.4/O36.4
178	N25.4	N28.2	I42.3/O42.3
179	N28.2	N29	I48.2/O48.2
180	N29	N12	I49/O49
181	N12	N0	I16/O16
182	N0	N1	I0/O1
183	N1	N2	I1/O3
184	N2	N3	I3/O3
185	N3	N4	I5/O5
186	N4	N5	I6/O6
187	N5	N6	I7/O7
188	N6	N0	I8/O8
189	N0	N1	I0/O1
190	N1	N2	I1/O3
191	N2	N0	I2/O2
192	N0	N0	verify N0

INTENTIONALLY LEFT BLANK.

NO. OF COPIES ORGANIZATION

2 DEFENSE TECHNICAL INFORMATION CENTER
DTIC DDA
8725 JOHN J KINGMAN RD
STE 0944
FT BELVOIR VA 22060-6218

1 HQDA
DAMO FDQ
D SCHMIDT
400 ARMY PENTAGON
WASHINGTON DC 20310-0460

1 OSD
OUSD(A&T)/ODDDR&E(R)
R J TREW
THE PENTAGON
WASHINGTON DC 20301-7100

1 DPTY CG FOR RDE HQ
US ARMY MATERIEL CMD
AMCRD
MG CALDWELL
5001 EISENHOWER AVE
ALEXANDRIA VA 22333-0001

1 INST FOR ADVNCD TCHNLGY
THE UNIV OF TEXAS AT AUSTIN
PO BOX 202797
AUSTIN TX 78720-2797

1 DARPA
B KASPAR
3701 N FAIRFAX DR
ARLINGTON VA 22203-1714

1 NAVAL SURFACE WARFARE CTR
CODE B07 J PENNELLA
17320 DAHLGREN RD
BLDG 1470 RM 1101
DAHLGREN VA 22448-5100

1 US MILITARY ACADEMY
MATH SCI CTR OF EXCELLENCE
DEPT OF MATHEMATICAL SCI
MAJ M D PHILLIPS
THAYER HALL
WEST POINT NY 10996-1786

NO. OF COPIES ORGANIZATION

1 DIRECTOR
US ARMY RESEARCH LAB
AMSRL DD
J J ROCCHIO
2800 POWDER MILL RD
ADELPHI MD 20783-1145

1 DIRECTOR
US ARMY RESEARCH LAB
AMSRL CS AS (RECORDS MGMT)
2800 POWDER MILL RD
ADELPHI MD 20783-1145

3 DIRECTOR
US ARMY RESEARCH LAB
AMSRL CI LL
2800 POWDER MILL RD
ADELPHI MD 20783-1145

ABERDEEN PROVING GROUND

4 DIR USARL
AMSRL CI LP (305)

NO. OF
COPIES ORGANIZATION

1 DIRECTOR
US ARMY RESEARCH LAB
AMSRL IS
J GANTT
2800 POWDER MILL RD
ADELPHI MD 20783-1145

1 DIRECTOR
US ARMY RESEARCH LAB
AMSRL IS T
J GOWENS
2800 POWDER MILL RD
ADELPHI MD 20783-1145

ABERDEEN PROVING GROUND

23 DIR USARL
AMSRL IS T
S CHAMBERLAIN
AMSRL IS TP
B COOPER (11 CPS)
C RETTER
F BRUNDICK
L MARVEL
AMSRL IS CI
B BROOME
H CATON
G HARTWIG (4 CPS)
M LOPEZ
A BRODEEN

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1999	Final, June–October 1998

4. TITLE AND SUBTITLE
Refining VHDL Specifications Through Conformance Testing: Case Study of an Adaptive Computing Architecture

5. FUNDING NUMBERS

6. AUTHOR(S)
Ali Y. Duale,* Bruce D. MClure,** and M. Ümit Uyar*

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-IS-TP
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-2010

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
*Electrical Engineering Department, City College of the City University of New York, New York, NY 10031
**Defence Science and Technology Organization, P.O. Box 1500, Salisbury, SA 5108, Australia

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The aim of this work is to enable a more rigorously tested product by integrating protocol specification and conformance test sequence generation. Such an integration will allow for the removal of costly mistakes from a specification at an early stage of the development process before they propagate into different implementations, possibly combined with other errors. This integrated approach has been applied to the VHDL specification of a military-oriented protocol prototype called the "Local Proxy." Based on the results of the conformance test generation process, the Local Proxy specification has been refined by uncovering various missing actions, removing redundancies, and restructuring the specification to improve its testability.

14. SUBJECT TERMS
VHDL, test generation, FSM, EFSM

15. NUMBER OF PAGES
35

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-TR-2010 (Duale) Date of Report June 1999

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name E-mail Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)