DIS APPLICATION PROTOCOL TESTING USING A FORMAL DESCRIPTION TECHNIQUE

David T. Shen, Margaret L. Loper, Huat K. Ng
Institute for Simulation and Training
University of Central Florida
Orlando, Florida

ABSTRACT

The Distributed Interactive Simulation (DIS) Entity Information and Entity Interaction Draft Standard defines a communication protocol to interconnect simulators in a real-time environment. This protocol focuses on the information for describing the state of the simulated entities and their interactions during a battle simulation.

Many of the concepts in the DIS standard are derived from the Simulation Network (SIMNET) project. The SIMNET program has demonstrated the feasibility of interconnecting multiple autonomous simulators, primarily of ground based armor vehicles, via a communication network (LAN/WAN), such that the simulators could interact in real-time. DIS is based upon the foundations of SIMNET and will be enhanced to provide a standard for connecting existing and future simulators. However, a formal description of the DIS standard is yet to be developed, which would, in turn, speed its prototyping, development, and testing processes.

This paper describes the approach taken by the Institute for Simulation and Training (IST) to develop a formal description of DIS, to generate a prototype DIS protocol machine derived directly from the developed DIS formal description, and to test the prototype DIS protocol. IST used an International Organization for Standardization (ISO) standard Formal Description Technique (FDT) called Estelle for the formal specification, and an FDT prototyping tool known as the Portable Estelle Translator - Distributed Implementation Generator (PET-DINGO) to generate a DIS prototype. Furthermore IST developed a procedure to test the generated prototype.

The aim of this work was to develop a formal specification for DIS and to identify possible shortcomings and inconsistencies in the DIS standard. This task identified some areas in the current standard that need to be modified or clarified.

ABOUT THE AUTHORS

David Shen received his master degree in Computer Engineering from UCF in 1991. He is currently an Associate Engineer at the Institute for Simulation and Training (IST). For the past two years, he has done research related with computer communication protocols applied to the DIS application.

Margaret Loper received her master degree in Computer Engineering from UCF in 1991. She is currently a Research Associate at IST. She has technical responsibility for all IST research activities involving communication protocols and computer networks.

Huat Ng received his master degree in Electrical Engineering from UCF in 1989. He is currently an Associate Engineer at IST. He has been involved in software development that pertains to DIS application.

DIS APPLICATION PROTOCOL TESTING USING A FORMAL DESCRIPTION TECHNIQUE

David T. Shen, Margaret L. Loper, Huat K. Ng
Institute for Simulation and Training
University of Central Florida
Orlando, Florida

INTRODUCTION

The Institute for Simulation and Training, along with the U. S. ARMY Simulation Training and Instrumentation Command (STRICOM) and Defense Advance Research Project Agency (DARPA), has been promoting the interoperability of defense simulations through the DIS workshops. The goal of the workshops is to gather expertise from the simulation community and to advance the development of a standard to allow defense simulations to interact through networking. These types of simulations provide an environment for training inter-crew skills and provide an environment for evaluation of tactics, doctrines, and new equipment features.

The developed standard must be tested for validation. This test will create confidence in the standard's completeness and domain of applicability for implementers. This validation should begin with the initial stage of release.

The DIS draft standard is in its second year of development. The standard is being reviewed and is proposed to the IEEE in 1992 for adoption. Currently, the DIS standard covers only layer 7 of the Open System Interconnection (OSI) Reference Model [4]. Layer 7 is the Application Layer, which is the simulation entity information level. It does not cover the network architecture or communication protocols below the Application Layer. Several defense procurements have specified the DIS standard as the baseline for the interoperability requirements.

IST has experience in many areas of computer and simulator networking. In its effort to assess the suitability of existing OSI protocols for real-time simulations, IST has developed a formal specification of the DIS draft standard using the Estelle Formal Description Technique (FDT) and the Portable Estelle Translator - Distributed Implementation Generator (PET-DINGO) compiler.

Estelle is a standard FDT developed in 1989 by the International Organization for Standardization (ISO) to specify distributed, concurrent information processing systems using a Pascal like language. Estelle is not a programming language for implementation, but simply for specification, with its own syntax and semantics.

The PET-DINGO compiler, developed by the National Institute of Standards and Technology (NIST), is a protocol prototyping tool that accepts an Estelle specification and produces a runtime environment simulating the specified protocol behavior. The PET-DINGO was developed to familiarize the scientific community with the Estelle specification language and thus promote its use.

GENERAL DESCRIPTION

This section briefly describes the DIS standard, the Estelle FDT and the PET-DINGO compiler.

Distributed Interactive Simulation

The October 30, 1991 release of the DIS standard defines 10 types of Protocol Data Units (PDUs) to be used by networked simulators to represent the state of the simulation entities during a battle simulation [1]:

1) Entity State
2) Service Request
3) Repair Complete
4) Repair Response
5) Resupply Offer
6) Resupply Received
7) Resupply Cancel
8) Collision
9) Fire
10) Detonation

These 10 PDU types can support the entity behavior during repair, resupply, fire, collision, motion and entity appearance update, which are the vital components

of a simulation environment, and they are the core of this research task.

The Entity State PDU is used most often during a simulation exercise. It describes the status of its simulated entity's location, velocity, acceleration, ammunition, and articulated parts. It is transmitted when a dead reckoning model of an entity's state diverges from its own high fidelity model by a predetermined threshold. When an Entity State PDU is received by other simulators in the exercise, these simulators update their state information regarding the entity.

The Service Request, Repair Complete and Repair Response PDUs are used to represent a repair event involving two simulated entities. These PDUs carry the necessary information to determine the types of repair performed and the level of satisfaction of the repair. The repair PDUs are request/response type, meaning, some PDUs are used in response to the receipt of other PDUs.

The Service request, Resupply Offer, Resupply Received and Resupply Cancel are used to represent a resupply event involving two simulated entities. These PDUs contain the necessary information to determine the type of resupply needed. The resupply PDUs are also request/response type, which means that some PDUs are used in response to the receipt of other PDUs.

The Collision PDU is used by two simulated entities when involved in a collision. The information contained in this PDU is needed for collision damage assessment.

The Fire PDU and the Detonation PDU are used to inform a target entity of the weapon fire event and the associated detonation of the fired munition. The entity which fires a weapon models the munition's trajectory and informs the target of the point of impact. It is the responsibility of the targeted entity to assess its damage using the information in the Detonation PDU.

Estelle Formal Description Technique

The problem of specifying distributed systems is more difficult than that of specifying a sequential system. The difficulties are related to the necessity of describing various sequential components which may cooperate and execute in parallel. To attain reliability in production software, a protocol development should begin with a formal specification.

Estelle is a language for specifying distributed systems with a particular application in mind, namely that of communication protocols and services [2,5]. The semantics for Estelle have been formally defined and are aimed at describing structured communicating automata (states) whose internal actions are defined by Pascal programming language statements (with some restrictions and extensions).

The benefit in using an FDT, particularly Estelle, is to remove the ambiguities from the protocol description, traditionally defined in a combination of natural language and state tables. Another benefit is the availability of tools that use Estelle to generate rapid prototypes and test suites.

The three main components of the Estelle structure are the Module, the Interaction, and the Channel. The correct use of these components is essential for a specification.

The Modules have a number of input/output access points known as the Interaction Points. A module will be represented graphically as a rectangle and the interaction points will be represented by dots on its boundary. An active module includes in its transition part at least one transition. Each active module specifies its own states and the rules for state transition. The collection of the states, the possible state transitions, and the module variables is known as the Extended State Transition Model (ESTM).

The modules can be organized in a hierarchical (tree) structure, with parent/children relationships. Each module can have several embedded modules (children), and these modules can, in turn, include other embedded modules.

The Interactions are the messages sent and received by the modules. The interactions serve as the means of communicating information between modules. The modules can send an interaction at any time, yielding non-blocking communication. An interaction sent through an interaction point that is an end-point of a communication link directly arrives at the other end point of this link and is always accepted by the receiving module. Thus, only end-to-end communication between modules is possible.

Modules (parents or children) can establish communication with other modules using channels. Channels are defined as one-to-one connections between the modules through which the modules interact. Each channel essentially connects two interaction

points. Each channel is associated with an unbounded first-in-first-out queue for the incoming interactions as they arrive for processing. There are several restrictions imposed in the way channels may interconnect the modules. Channels also define the types of interactions that may pass through it.

Each Estelle module definition is composed of a heading and a body that describes its behavior. A transition takes place in response to an internal or external event to the module and it may generate an interaction to the connected modules. The global properties of the Estelle FDT support a logical specification (modeling) of a communication protocol.

Portable Estelle Translator & Distributed Implementation Generator

The PET-DINGO prototyping tool, developed by NIST, is a twofold compiler that generates a static model and a dynamic model from an Estelle specification. The PET compiler [7] takes an Estelle specification and checks it for syntax, semantic or lexical error and then compiles it into an object code (static model). The DINGO compiler [8] takes the static model and generates a series of C++ source files associated with specified Estelle modules describing their behavior (dynamic model).

The PET-DINGO was written in the C++ programming language. It supports network communication using either Transmission Control Protocol/Internet Protocol (TCP/IP) or Remote Procedure Call (RPC); as a consequence, generated processes can run on diverse computers on a network. The PET-DINGO is designed to run on a Sun3 or Sun SPARC hardware platform. It also supports the X-window environment, which facilitates the user's interaction with the specified protocol processes.

Figure 1 shows the process to build and execute a runtime instance of an Estelle specification:

1) Compile the Estelle specification using the PET to generate an object file. At this stage, the decision whether to run the specification in a multi-host environment is made.

2) Compile the PET generated object file using DINGO to generate C++ source files and the Makefile.

3) Add user defined programs and set system parameters.

4) Modify the Makefile as needed by including user defined programs.

5) Use the Makefile to generate system executables from the C++ files, which are the Estelle modules.

6) Initialize the site-daemon (called site_serv) to manage the runtime processes and the network interface. The site-daemon has to be initialized in each host if the process is to run in a multi-host environment.

7) Call the Estelle specification top level module name for modules initialization.

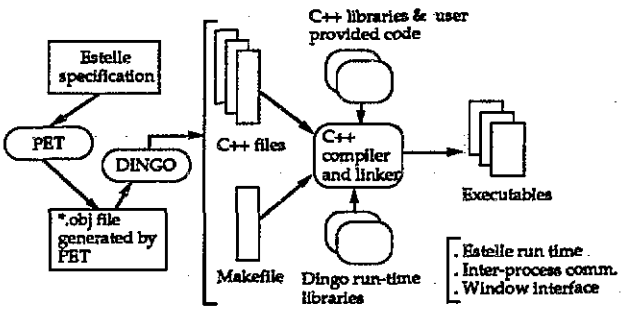

Figure 1- PET-DINGO Compilation procedure

Each Estelle module is a process that can be accessed through a window interface. A user specifies interactions by clicking the appropriate item of the window. The windows display modules' state, the value of the variables at any given time, and the queue of interactions associated with each channel connected to the module.

The module windows can be opened by clicking the name of the module on the root window. The embedded process windows can be opened by first opening their parent's window. The hierarchy of the windows follows the hierarchy of the specified Estelle modules.

SPECIFICATION PHASE

This section describes the approach taken to specify the DIS standard using the Estelle FDT. The model used for the DIS protocol with the associated assumptions and constraints is also described.

Approach

A formal specification of a protocol using the Estelle FDT is not unique, meaning there are several valid

ways to specify a particular protocol. IST has experienced several difficulties in specifying the DIS PDU interactions in Estelle because of the nature of the information passed from host to host and the type of interaction specified in the DIS standard.

The approach taken by IST to specify the DIS PDU interactions using Estelle is a model following an initiator/responder paradigm. This model uses a general view of a simulation entity from a driver's perspective, meaning that the model follows a hierarchical structure and the lower modules view the upper modules as the initiators of the processes by making the appropriate choices as to what actions to take.

Model

The model identifies the Repair and Resupply activities as the ones having a initiator/responder type of interaction. Other activities such as fire and collision are essentially non-replied interactions, which do not follow the initiator/responder scheme.

Figure 2 shows the model, which includes four basic modules, shown in Figure 2:

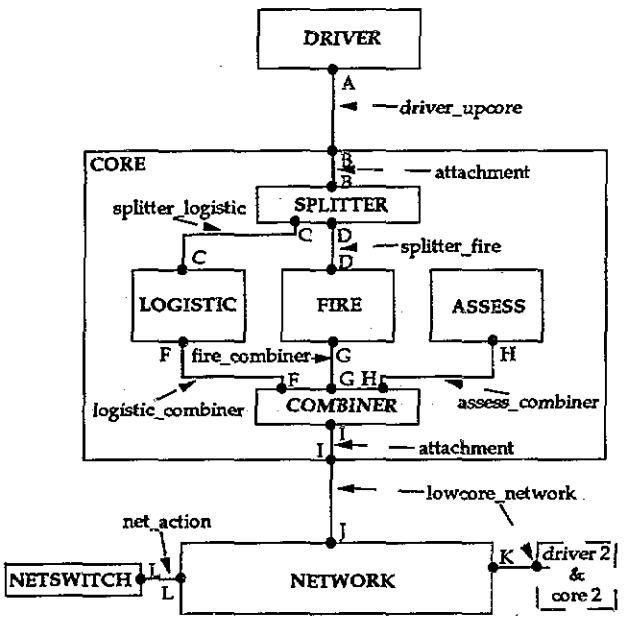

Figure 2 - Estelle Structure of DIS Protocol

Driver Module: interfaces the user to the protocols specified within the Core module. The Driver Module is responsible for starting a protocol process and communicating appropriate decisions.

Core Module: includes 5 embedded modules, namely the Splitter, Combiner, Logistic, Fire and Assess. It models the DIS PDU interactions.

Network: models the physical/logical linkage between the protocols (the entities). It is responsible for end-to-end transmission of protocol messages between two entities.

Network Switch: allows user control of the network, giving the user the ability to interrupt the message transmission. Its existence is purely for testing purposes.

Figure 2 shows the modules with their names in bold type, the associated interaction points in capital letters and the channel names indicated by arrows.

Within the Core module are the independent submodules of interest of this task:

Logistics Module: models both the repair and resupply (logistic) events of DIS. It can act as either initiator or responder, but not both. This module uses six types of PDUs (2 to 7 defined previously) to inform the peer entity of the action taken by the other. It incorporates reliability features by using some of the PDUs to acknowledge other PDUs. Such PDUs can be classified as an application level acknowledgement.

Fire Module: models the fire and detonation events. It uses the Fire PDU and Detonation PDU to convey related information to peers. The Fire PDU and the Detonation PDU do not require acknowledgement from their intended target.

Assess Module: responsible for updating the internal representation of a simulation. For instance, the Assess module represents updates to an entity's appearance during a simulation exercise when it receives an Entity State PDU, and it is also responsible for representing the fire event and damage assessment caused by a munition detonation or a collision. As far as the specification is concerned, the assessment means that the module transitions from an idle state to a particular type of assessment state and then back to the idle state.

There are two auxiliary sub-modules within the Core module:

Splitter Module: responsible for vectoring the user interactions (signals) to the appropriate protocols within peer sub-modules. The rationale is to allow a higher level of abstraction for the connection between the Driver module and the Core module. Without the Splitter module, one would have to be concerned with the various connections from the Driver module to the Core sub-modules.

Combiner Module: responsible for piping the outgoing interactions (PDUs) from protocols within peer sub-modules (i.e., Logistics, Fire or Assess sub-modules) to a single channel connected to the Network module. It is analogous to the Splitter module performing the reverse function. The rationale is to allow a higher level of abstraction for the connection between the Core sub-modules and the Network module. It is also responsible for conveying state variables among sibling modules.

Among all the referred modules, the Logistic, the Fire and the Assess modules are the only ones that are mapped into the DIS PDU interaction, other modules are intended for interface and testing purposes and are not part of the DIS specification.

The extended state transition model of the modules that maps to the DIS PDU standard are explained below:

Figure 3 unifies the existing repair/resupply related state transition models defined in the DIS standard with the entity movement and collision activities. The repair/resupply side from one entity responds to the repair/resupply side of another entity, which creates a reply/response type of interaction between two Logistic modules. A clock routine returns the system time which is used in the repair and resupply transition models.

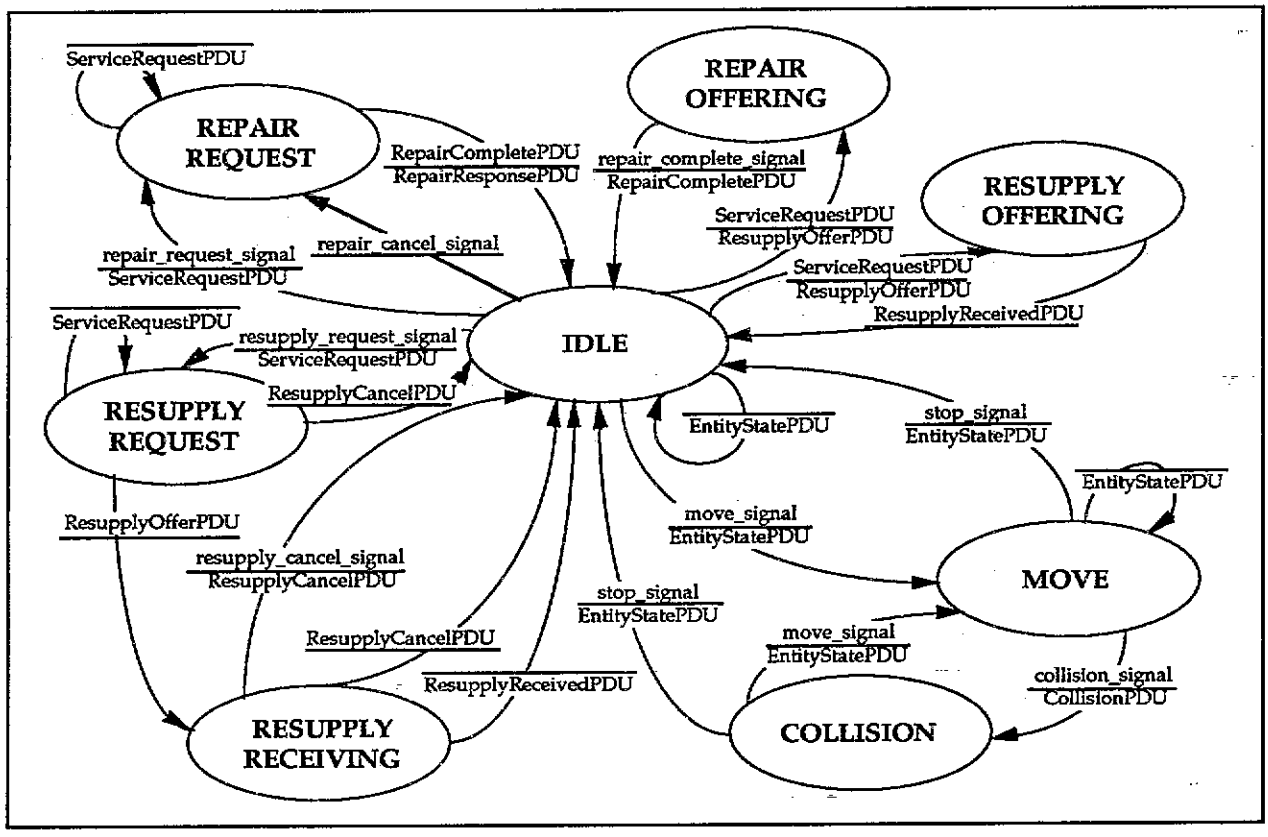

Figure 3 - Extended State Transition Model for Logistic Module

Because no collision Extended State Transition Model (ESTM) is specified in DIS, the SIMNET model is assumed. An entity involved in a collision sends a Collision PDU to the other entity, communicating the collision. The receiver of the Collision PDU replies to the first entity with another Collision PDU, as an acknowledgement of the collision.

The Fire module, shown in Figure 4, is composed of two states: IDLE and FIRE. An entity transitions to the FIRE state on the Driver's signal representing a weapon fire and returns to IDLE state after a delay simulating the time required for flight and detonation of the fired munition.

Because no fire ESTM is specified in DIS, the SIMNET model is assumed. The initiator sends a Fire PDU immediately followed by a Detonation PDU. These PDUs are conveyed to the Assessment module which simulates the internal processing of the receiving simulator.

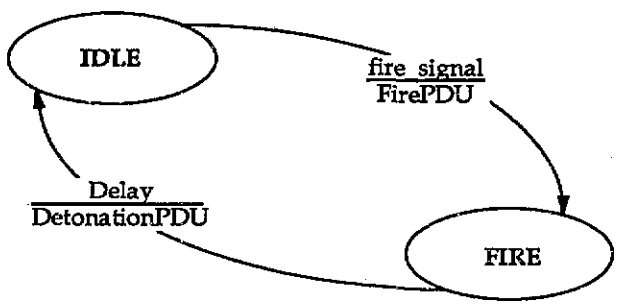

Figure 4 - Extended State Transition Model for Fire Module

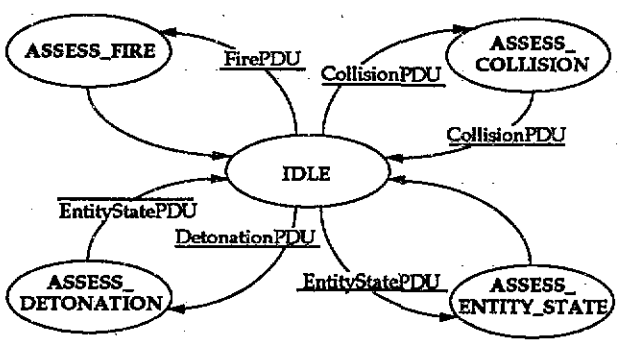

Figure 5 - Extended State Transition Model for Assess Module

The ASSESS module in Figure 5 represents the possible assessments a simulator can perform: fire assessment, detonation assessment, collision assessment, and entity state assessment.

Because no assessment ESTM is specified in DIS, the SIMNET model is assumed. It serves the purpose of isolating the information update event from the usual simulation related events, allowing concurrent processing for the specified model.

Assumptions

There are several assumptions made to facilitate the model's creation:

1) *The repair activities and resupply activities* (logistic) do not take place concurrently. This allows all four state transition models defined in the standard associated with these activities to be combined in a single transition model.

2) An entity cannot repair or resupply while in motion. This assumption allows the inclusion of collision and movement events in the logistic module.

3) The logistic activities, the fire activities and the PDU assessment activities can be processed concurrently. This is represented in the core module through three independent sub-modules.

4) The resupply activity, once initiated, takes 15 seconds to complete.

5) The interval between consecutive Entity State PDUs is 1 second for a moving entity.

6) The interval between a weapon fire and the detonation of the fired munition is 2 seconds.

7) The assessment related activities are instantaneous, which is represented by the immediate transition from any type of assessment state to idle state in the Assess module.

8) The IDs used by the entities for identifying the collision events are consecutive integers starting at 1.

9) All interactions are received by the Assess module except for the repair and resupply related interactions.

Constraints

The following constraints are imposed by the Estelle specification:

1) There are only two entities modeled in the module for testing purpose.

2) The module interactions do not carry variable structure or concrete values.

TESTING PHASE

As mentioned earlier, the DIS Standard is in its infancy. Efforts must be channeled toward writing a test plan that will discover whether the DIS protocol is complete and valid. The DIS Standard specifies procedures and formats for the exchange of information between heterogeneous simulators. Based on these specifications, a test plan can be written.

Protocol tests can be divided into three different types: Valid, Inopportune, and Invalid Message Tests [1]. Valid Tests are those where the tester sends messages at times and in sequences that are expected or normal for the Implementation Under Test's (IUT's) state. Inopportune Tests are those where the tester sends messages at times when they should not occur or are out of sequence. Invalid Message Tests determine if the IUT correctly handles receipt of messages that are incorrectly encoded, have illegal fields, or have parameters outside their legal bounds. Since the DIS specification in this report does not include the actual bit structure of the PDUs, Invalid Message Tests are inappropriate and will not be included in the test plan.

The DIS validation is an interactive process. Experience gained during the testing phase is used to enhance the quality of the specification and testing process. If the test plan uncovers an ambiguity or incompleteness in the specification, the specification can be modified and re-tested, leading to further refinement.

The hardware used in the IST testing environment consists of two Sun SPARC Workstations connected by Ethernet. The systems used Sun Open Window, a graphical user interface based on the X-window environment. The experiments were conducted in a multi-host environment, and both computers shared the files generated by PET-DINGO.

Testing Procedure

Each test step follows this procedure:

- Initialize the IUT and manipulate it into the desired state.
- Apply the specified input.
- Observe and validate the output.
- Verify the new state is as expected.

There are two choices available when executing the runtime PET-DINGO files of an Estelle specification: continuous mode or single-step mode. Running the DIS specification in single-step mode is the more appropriate choice. This allows the tester to observe the changed states and messages passed between modules at his or her own pace.

Valid Testing - The test plan is written in a tabular format. It consists of Valid test scripts. Four tables describe the DIS state transitions. These are:

- Logistic / Logistic - Resupply Service
- Logistic / Logistic - Repair Service
- Logistic / Assess
- Fire / Assess

Each table is separated into the Initiator column and the Responder column. Both Initiator and Responder are separated again into four other columns: Input, Output, Current State and Next State. In this fashion, the tables follow a natural sequence of state transitions, which facilitate the observation of the transitions and messages that are conveyed between two modules. The Logistic/Logistic - Resupply Service used in the test plan is shown in Table 1.

Table1 shows the Initial transitions and the corresponding responding transitions for the resupply services available in the DIS protocol. Resupply services may include resupplying for fuel or munitions. Using Table 1 and following the transitions, a list of the Initial/Respond transitions can be obtained and checked against the DIS standard.

For example, an entity may request resupplies by sending a Service Request PDU. The resupply receiver, will change state, transitioning from Idle to the Requesting State and will respond by sending a Resupply Offer PDU. The supplier entity will then transition from Idle to the Offering State. This set of transitions can be traced by observing the first row of Table 1. Executing the DIS specification in single-step mode allows the tester to verify the steps and observe the messages in the queues.

Table 1 - Logistic / Logistic - Resupply Service

Logistic (Initial) / Logistic (Respond)	Initial Transition				Respond Transition			
	Input	Output	Current State	Next State	Input	Output	Current State	Next State
Request Service / Offer Supplies	Request Resupply	Service Request PDU	Idle	Requesting	Service Request PDU	Resupply Offer PDU	Idle	Offering
Offer Supplies / Receive Offer	Service Request PDU	Resupply Offer PDU	Idle	Offering	Resupply Offer PDU	-	Requesting	Receiving
Accept Service / Resupply Complete	-	Resupply Received PDU	Receiving	Idle	Resupply Received PDU	-	Offering	Idle
Reject Offer / Resupply Cancelled (by Receiver)	Resupply Cancel	Resupply Cancel PDU	Receiving	Idle	Resupply Cancel PDU	-	Offering	Idle
Resupply Cancelled (by Supplier) / Transfered Cancel	Resupply Cancel	Resupply Cancel PDU	Offering	Idle	Resupply Cancel PDU	-	Receiving	Idle
Repeat Request / Offer Supplies	-	Service Request PDU	Requesting	Requesting	Service Request PDU	Resupply Offer PDU	Idle	Offering
Resupply Refuse / Cancel Request	Resupply Cancel	Resupply Cancel PDU	Idle	Idle	Resupply Cancel PDU	-	Requesting	Idle

When the Resupply Offer PDU is transmitted by the supplier, the resupply receiver receives the PDU and transitions from the Requesting State to Receiving State. This set of transitions are described in the second row of Table 1.

As can be seen from the above scenario, a set of initial and response transitions can be observed by following the test script in Table 1.

Inopportune Testing - The inopportune tests identify shortcomings in the protocol due to network failure and tests the protocols response in such situations.

For the inopportune tests, IST has developed test cases dealing with recovery from PDU loss. In all cases, the protocol recovered by a timeout mechanism. In test cases identified, DIS would simply discard the PDUs that do not apply to the state of the transition model.

For example, in row 1 of Table 1, if the Service Request PDU sent by the requester is lost, the responder remains in the Idle state. The requester would repeat the request by sending a Service Request PDU every 5 seconds until either it receives a response for its request or it gives up the request. A lost Service Request PDU has no great consequences.

Testing Results

The table presented in the previous section shows a detailed testing procedure for the DIS Protocols. The table is arranged into an Initiator/Responder analysis. All the initiated transitions were verified with the corresponding responder transitions. The experience gained during the testing phase was used to enhance the quality of the specification and the testing process. If an ambiguity in the specification was observed, the specification would be suitably modified.

As a result of the testing phase, two inconsistencies were found in the Logistic Module. In the first case, the Cancel Request transition in the resupply receiver module did not occur. This was because no interaction (Resupply Cancel PDU) originated from the supplier module to allow the receiver's Cancel Request transition to take place. The specification was modified to correct this inconsistency. With the addition made to the DIS specification, the testing procedures were changed to reflect this modification (Resupply Refuse transition in Figure 6).

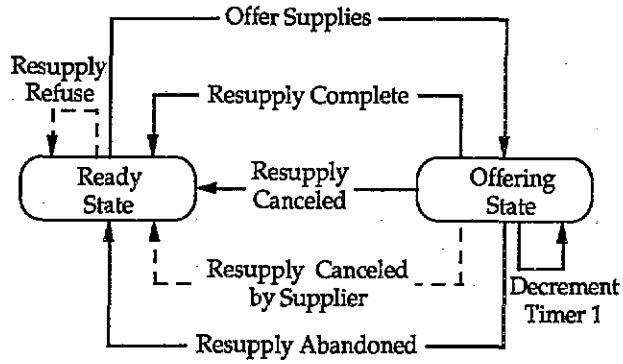

Figure 6 - Original DIS Standard Resupply Supplier State Transition Model with Corrections

For the second case, the Transfer Cancel transition in the receiver resupply module would not occur. As with the first case, the reason was that there was no interaction (Resupply Cancel PDU) originated from the supplier module to allow the receiver's Transfer Cancel transition to take place. Again, the specification was modified (Resupply Canceled by Supplier transition in Figure 6) to handle this deficiency.

The Fire and Assess state transition models were checked based on a number of assumptions on the standard and no ambiguities were found.

The DIS standard has provisions for the loss of packets in the case of the repair/resupply activities due to network failures. Because of this, the Inopportune test has shown those cases were handled properly. In the current DIS standard, a timeout mechanism is incorporated and has been verified to be adequate.

CONCLUSION

This work was successful in specifying the DIS standard within the context of a formal description technique. However, because of the nature of the current standard, several assumptions and restrictions were imposed on the entity responses defined in the PDUs. Without such assumptions and restrictions, this work would be cumbersome and unnecessarily extensive.

The current standard is not fault tolerant, i.e., the protocol can present misbehavior caused by network failures. However, the protocol works well in an environment of low network failure rate.

REFERENCES

[1] Bertine, Herbert V.; W. B. Elsner; P. K. Verma; K.T. Tewani, "Overview of Protocol Testing Programs, Methodologies, and Standards", *AT&T Technical Journal*, Jan/Feb 1990

[2] Budkowski, S.; P. Dembinski, "An Introduction to Estelle: A Specification Language for Distributed Systems", *Computer Networks and ISDN Systems* 14, 1987

[3] *Military Standard, "Protocol Data Units for Entity Information and Entity Interaction in a Distributed Interactive Simulation (Final Draft)"*, Institute for Simulation and Training IST-PD-91-1, October 30,1991

[4] International Organization for Standardization, *Open Systems Interconnection Reference Mode*, ISO7498, 1984

[5] International Organization for Standardization, *Estelle: A Formal Description Technique Based on an Extended State Transition Model*, ISO9047, August 15, 1989

[6] Pope, A; R. L. Schaffer, *The SIMNET Network and Protocols*, BBN Systems and Technologies, Report No. 7627, June 1991

[7] Sijelmassi, R.; B. Strausser, "The Portable Estelle Translator: An Overview and User Guide", U. S. Department of Commerce, National Institute of Standards and Technology, Technical Report NCSL/SNA-91/1, January 91

[8] Sijelmassi, R.; B. Strausser, "The Distributed Implementation Generator: An Overview and User Guide", U. S. Department of Commerce, National Institute of Standards and Technology, Technical Report NCSL/SNA-91/3, January 91