A DYNAMIC TESTER FOR USE WITH OSI NETWORKS

Nasser Modiri, M.Sc., D.Phil., A.M.I.E.E., M.I.E.E.E.

EleTek Systems
1107 Second Ave, Suite 704
Redwood City, CA 94063, USA.

Abstract.

The widespread acceptance of Open Systems Interconnection (OSI) by most large computer manufacturers and procurement agencies means that many computer networks are now being installed that are based on a standard protocol defined by the International Standards Organisation (ISO). Although prior to the commissioning of the communication subsystem within a networked computer the protocols making up the subsystem are subjected to rigourous conformance tests, should a fault be suspected in an operational system it is also necessary to provide a facility to allow the operation of a system to be tested dynamically via the network. Such instruments are referred to as dynamic testers and they are particularly useful during the commissioning of a new network, for example, and also if a system connected to an operational network is suspected of being faulty.

1 INTRODUCTION

The dynamic tester to be described has been designed to operate as an intelligent client system for the dynamic testing of a network server such as a large file or mail server. It is a flexible tool that can be used to interactively perform specific conformance tests.

Essentially, associated with each test, there is a precompiled set of frames which can be transmitted to a system under test via the network. The response is then compared with the expected response.

The dynamic tester has many features similar to a conventional conformance tester [1,2,3,4,5,6] except that it is designed to operate over an operational network rather than, say, in a conformance laboratory. Also, it is designed to test a complete protocol suite rather than, say, an isolated protocol layer.

The paper comprises five sections. After this section, Section 2 gives the rationale behind the work, Section 3 describes the overall structure of the dynamic tester in which the test definition and run-time components are described, Section 4 describes some implementation aspects of the dynamic tester, and finally, Section 5 concludes the paper.

2 RATIONALE

Prior to acceptance of a system to be connected to an OSI network, it is likely that the system would have been subjected to rigourous conformance tests. It is, however, possible that some faults may only manifest themselves in an operational situation. There is also the possibility of an untested system being connected to the network.

The approach taken in designing the dynamic tester has been to provide a user-friendly interface (all operations are menu driven) and to minimise the level of expertise required by the operator. The selected protocols to be used in the OSI Environment (OSIE) - that is, the configuration profile - will have been previously defined, and hence the conformance requirements for a system connected to the network will be known. The conformance tests to be carried out, therefore, aim to establish that a system adheres firmly to the standards selected. The dynamic tester is designed to operate as a client which can be used to test a remote server.

It performs this function by transmitting previously compiled frames to the remote system (server) under test. Each frame is designed to test a specific step in an overall transaction involving the server; for example, the establishment of an association between, say, the File Transfer Access and Management (FTAM)[7] application entity in a file server and the tester. Associated with each test there is a frame which has previously been compiled representing the expected response.

This frame is then compared with the actual response frame received from the remote server. If the frames are the same, the test is assumed to be successful; if they are different, however, then the tester will identify the field(s) that is (are) in error, the layer to which it relates, and the actual symbolic name of the field as used in the appropriate standards document. In order to test all aspects of a particular server, a number of test scenarios are devised. Frames to be transmitted and frames expected to be received, relating to each test, are created prior to delivery of the tester. Since there may be many tests and the amount of data relating to a complete test suite might be large, the data relating to each test are first stored on disc. Subsequent analysis may then take place in an off-line mode rather than an on-line mode.

3 OVERALL STRUCTURE

The overall structure of the software associated with the dynamic tester is comprised of two main components the Test Definition component and the Run-time component.

Essentially, the Test Definition component is used to configure the tester prior to delivery whilst the Run-time component, as its name implies, is used to perform the actual dynamic test operations over, an operational network in the field. The functional overview of the dynamic tester is shown in Fig. 1.

A schematic diagram showing the structure of the hierarchical menu of each component is shown in Fig. 2. As can be seen from the figure, the Test Definition component comprises three activities - *Configure*, *Scenario Entry* and *Precompile*. Once these activities have been carried out, there are further utilities which are used to generate the Run-time version of the tester. Using this procedure it is possible to make available to a user only the Run-time version of the tester which has been specifically tailored to the OSI being monitored and managed.

3.1 Test Definition

The Test Definition software is used to configure a tester for use in a particular OSIE prior to delivery of the tester. It is comprised of three major components - *Configure*, *Scenario Entry* and *Precompile*. Each is now described separately.

Configure

The *Configure* component which is run first, is used to select the particular configuration profile that is being used with the OSIE. It is comprised of two main parts, namely *Profile Selection* and *Generate*. An outline of their functions is presented here.

The set of protocols to be used with the dynamic tester is first selected from a list of possible protocols using the *Profile Selection* component. An example of its use is shown in Fig. 3 (a..h). Essentially, the operator first selects the appropriate protocol suite needed to make up a tester for use in this environment, then, as each layer in the reference model is selected, all the protocol standards supported for the selected layer are listed. The user is thus able to select the appropriate protocol to be used for each layer simply by using a selection button on the mouse. Then as each protocol is selected, this information is added to the configuration file of the tester prior to compilation.

The operator can compile the configuration selected by selecting the *Compile* option as shown in Fig. 4. Hence, after compilation, just the object code necessary for the selected protocols of the *Precompile*, *Compile* and *Analysis* components is produced.

Once the set of protocols to be used in the environment has been selected, it is then necessary to generate a precise definition of the Protocol Data Units (PDUs) associated with each selected protocol. Many of the ISO standards have a number of optional features associated with them, for example with respect to the inclusion of a particular field within a PDU. Hence, having selected a configuration profile, it is then necessary for the user to select the inclusion (or otherwise) of those fields that are optional in each protocol making up the selected profile. This is accomplished using the interactive software tool, *Generate*, an example of which is shown in Fig. 5 (a..b). Essentially, as each protocol layer is reselected, then the list of PDUs associated with the previously selected protocol is presented. The user then selects each PDU in turn and, if any of the fields are optional, the user is prompted to specify whether the field should be included or not. Thus after this phase a precise template for each protocol in the suite will have been generated. These are then used in the subsequent *Precompile* phase and later during the *Compile* and *Analysis* phases of the Run-time component.

Scenario Entry

The overall test of an operational system is comprised of a number of sub-tests each of which is known as a scenario.

Typically, associated with each scenario is a sequence of previously compiled frames - together with their expected responses - each of which has been selected to test a specific aspect of the server. Defining scenarios for a test case of a System Under Test (SUT) (that is, a real open system) can be arrived at by looking at the conformance requirements relating to the ISO standards being used. As an example, assume it is required to devise a sequence of tests for a server based on the FTAM application entity. A typical test scenario that has been devised is shown in Table 1. The scenario shows the design to test the association establishment and termination components of FTAM. The layer PDUs making up the sent and expected frames associated with each test in a scenario are kept in a table known as the test definition table.

The *Scenario Entry* component, therefore, has been devised to allow an operator to interactively enter the contents of this table. As can be seen, this is a two dimensional table the first column of which specifies the name of the frames involved in each test. The second column denotes the direction of the frame (that is, sent (S) or expected (E)). The remaining eight columns then indicate the PDUs relating to each layer that make up the composite frames that will be generated for the test case. It is then the contents of the test definition tables that are passed to the *Precompile* component.

Precompile

The *Precompile* phase involves first looking up the PDUs that have been entered into the test definition tables to ascertain the composition of each sent and expected frame then, for each PDU, assigning the fields to either a default value or a value specified by the operator. The operator is first requested to select the application entity - FTAM or EM[8] - to be tested and a scenario number specifying the selected test. The PDUs in each test frame relating to the selected test are then listed and the operator is able to enter the appropriate value for each field in each PDU. As an example, assume it is required wish to precompile a test suite relating to an FTAM server. A typical sequence of events is as shown in Fig. 6.

First the user in part (a) selects FTAM and the test scenario number 001. In response, the user is then presented with the contents of the test definition table relating to this test scenario. This is shown in part (b) and is the same as that shown earlier in the *Scenario Entry* section. The test sequence is thus concerned with the establishment and termination of an FTAM association. Before this can be carried out, however, a transport connection between the tester and the server must first be established, the first two entries - sent and expected - performing this function. With the aid of the mouse the user is then able to proceed to define the fields in each PDU making up each frame and as an example, the windows in part (c) illustrate how the fields relating to the Transport Connection Request (CR) PDU (TPDU) [9] component of the first frame are presented. In practice these are then followed by the fields in the Unit Data Request (DT) Network PDU (NPDU) [10], the Un-

numbered Information Request (UI) Logical Link PDU (LPDU) [11], and the Data Request (DT) Medium Access Control (MAC) PDU [12]. As can be seen, the user is presented with a default value for each field and this can either be accepted (by pressing the return key on the keyboard) or a new value specified.

This procedure continues for each frame in the selected test scenario and, for each frame, a data file is produced containing the corresponding octet string. Only those fields that contain non-site-specific data values can be entered in this way but in practice these are the majority and only a small number of values - for example, the fully qualified address of the server under test - need be entered at run time.

An important component of the *Precompile* software is the set of so-called Precompile Protocol Engines (PPEs). Essentially there is a separate PPE for each protocol layer and its function is to produce the precompiled octet string for each layer PDU. The sent and expected frames for each test are thus a collection of the outputs from these PPEs.

A subdirectory is assigned to all the functionality required from a protocol layer and a layer PPE handles the management of the operations of the layer. In this way, development of the software can be devised in a straight forward way and more importantly, since the ISO standards are evolving, changing or adding new standards to the existing standards considered for the dynamic tester can be carried out more readily. Further, debugging of the dynamic tester components can be completed more quickly by running the PPEs independently from one another. In the *Precompile* stage for each PDU making up a subset frame, the PPEs display the elements of the PDU. Against each element typical default values are displayed and the user is then given the option of either accepting this value or entering a new value. After all the fields of a PDU have been entered, the PPE software for that layer produces the corresponding octet string. In addition to converting each field, the layer PPE also performs operations such as the computation of the number of octets in the PDU - the length indication - and the computation of the checksum for the precompiled octets.

Whenever possible maximum information is made available to the operator, so that he/she does not have to refer back to the standard documents, for example, to verify the encoding rules which define a PDU. Hence, the operator is able to use the dynamic tester to devise valid PDUs and consequently test scenarios with a minimum knowledge of the standards.

3.2 Run-time components

The Run-time component is comprised of four parts, as shown in Fig. 2. These are - *Compile*, *Concatenate*, *Test*, and *Analysis*. Each of these components now will be described with more details about the *Test* component given as a part of the Implementation Aspects of the dynamic tester.

Compile

The main function of the *Compile* stage is to allow a user to interactively specify any site-specific fields associated with the PDUs that make up each frame in the test scenarios. The input to the *Compile* stage is the set of data files that were generated at the *Precompile* phase - prior to delivery of the tester - and also the set of test definition tables that were entered during the *Scenario Entry* phase. The *Compile* component is selected by the user from the main menu shown in Fig. 2. The sequence of operations with the *Compile* component are similar to those of the *Precompile* component described earlier except that here the user is asked to enter only the site-specific fields associated with each PDU in the set of test frames.

As an example, consider the case where a user wishes to test an FTAM server. After selecting the *Compile* option, the user would then select FTAM followed by a scenario number. This is shown in part (a) of Fig. 7.

The first PDU of this test is the CR TPDU and, as can be seen in part (b) of this figure, the user is presented with the precompiled (that is, site-independent) fields and is prompted with the first site-specific element - that is, Source Reference. The user then either accepts the default value displayed or specifies another value. This procedure continues until all the fields in each PDU relating to this scenario have been defined - that is, all the PDUs in both the sent and expected frames. A second example is shown in Fig. 8 part (a..d).

This figure relates to a PDU of the FTAM entity itself and, as can be seen, fields are specified in an Abstract Syntax Notation One (ASN.1)[13,14] form. In the example the user is being prompted for the concrete value notation of the FTAM initialize request (INIRQ) PDU.

In the absence of a protocol stack in the dynamic tester - the octet strings created are transmitted directly onto the transmission medium - the user must know the fully qualified network address of the FTAM server under test. The user then enters each subaddress of the network address in the appropriate layer PDU as it is being compiled.

The defined fields making up each PDU are then processed by the appropriate layer Compile Protocol Engine (CPE). There is a separate CPE for each protocol layer and its function is to produce the corresponding octet string for the PDU. Thus the Transport Layer CPE will compute the length indication field and the checksum fields prior to compilation.

Similarly, for those protocol layers that are defined in ASN.1, the user will specify each field in this form and the corresponding CPE then compiles these into their concrete syntax form. Each compiled PDU is then stored in a separate file. The compiled PDUs making up each sent and expected frame must then be concatenated together to produce the octet strings that are transmitted and received.

Concatenate

Generally, a scenario comprises a number of sent and corresponding expected frames. These frames are made by concatenating the PDUs produced during the *Compile* phase.

Thus for the scenario test 001 of FTAM, eight data files would be produced containing a sequence of sent and expected frames as originally defined in the test definition table for this scenario. This *Compile - Concatenate* cycle is repeated for each scenario. The dynamic tester is then ready to start the actual test cycle.

Test

After a user has produced the compiled and concatenated set of frames relating to each scenario, the server may now be dynamically tested via the network. Clearly, there are two approaches that may be adopted: either each frame in a scenario is transmitted separately, or, the complete suite of frames relating to a scenario is transmitted contiguously. In practice, since the tests are to be carried out on an operational system, then the first approach may result in the server timing-out between tests. To overcome this problem, when a scenario is invoked, each frame is transmitted and a simple comparison - in terms of length and content - is made to determine whether the fields in the response frame are valid prior to sending the next frame in the scenario.

In addition, however, as each response frame is received is stored in a data file in the event of a more detailed analysis being required during the subsequent *Analysis* phase. Should a response frame be deemed to be incorrect, then the scenario is aborted and the *Analysis* phase is entered to determine the error.

In the event of the latter, however, then the association and network connections are both closed.

Analysis

After the *Test* phase, the frames received from the SUT in response to each transmitted frame will be stored in a data file. If all the test in each scenario have been carried out successfully, then all is well and there is no need to proceed further. If a test has failed, however, then the actual received frames can be analysed to ascertain the fault by entering the *Analysis* phase.

To use the *Analysis* component the user selects *Analysis* option from the main menu together with the name of the server - FTAM or EM. In response the user is presented with a summary of the previous *Test* phase. An example is shown in Fig. 9. As can be seen, first a summary of the success or failure of each scenario is presented and, in the event of a scenario failure, a summary is also presented of the success or otherwise of each test frame in the scenario. Thus in the example scenario 003 was aborted because frame 002 was invalid, Table 2 shows the frames and PDUs that comprise the 003 test of FTAM.

The user must now determine which field(s) and in which PDU(s) are invalid. To help the user, the contents of the failed frame are automatically passed and a summary is presented in a third window. In addition, the contents of the Protocol Control Information (PCI) of the invalid PDU are presented in a fourth window and those fields that are different from those expected are highlighted. Clearly, if the parsing operation fails, then only those fields that have been successfully parsed are presented.

4 DEVELOPMENT ENVIRONMENT AND TESTING

As has been described, the dynamic tester is not designed to test a single protocol layer associated with communication subsystem but rather to dynamically test complete protocol stack including a selected application entity; for example, a file server that is based on FTAM. The dynamic tester is also based on a Sun Workstation® [15] and all software is written in C [16] running under the UNIX™ operating system. The physical interface between the Sun and the network will, of course, be network dependent. Two boards for use with a Technical Office Protocol (TOP)[17] and Manufacturing Automation Protocol (MAP)[18] network. These are the MVME 374 Ethernet™ Controller board and the MVME 372 advanced MAP Controller board respectively.

A schematic diagram showing the actual test system is given in Fig. 10 part (a). The interface board in the tester does not contain the full protocol software. Rather it simply contains software to transmit the compiled frames associated with a test scenario onto the transmission medium and, as each response frame is received, to pass the response frames to the host processor for analysis/verification.

The development of the tester was carried out in a number of steps. First precompiled frames and their associated precompiled responses were used. The responses also has deliberately introduced errors associated with them and the software was able to detect an identify those fields that were in error and also any incorrect sequence errors.

Secondly, the FTAM server associated with the ISODE [19] software running on top of Transmission Control Protocol (TCP) was used to test a real system.

A schematic is shown in Fig. 10 part (b). As can be seen, the ISODE software comprises only layer 4 upwards but this formed a suitable vehicle for carrying out tests on the FTAM layer entity in particular.

The actual tester software occupies 1.8 Mbytes of main memory to test a TOP (or MAP) FTAM-based file server.

5 CONCLUSIONS.

Conventional conformance test tools [20,21] are extremely complex and often take a considerable amount of time to set up. In contrast the dynamic tester described in this paper has been designed to perform a reasonably detailed level of tests on a system that is connected to an operational network. The aim is for the tester to be used during the commissioning phase of a new network and also as a part of the general instrumentation available to a network manager of an OSI network.

6 ACKNOWLEDGEMENTS.

The work reported in this paper has been partially funded by The Networking Centre Ltd. of Hemel Hempstead, England. The author would like to acknowledge this support and also the many useful discussions with them about the facility offered by the tester.

7 REFERENCES.

[1] BICC Data Networks, "ISOLAN Test Unit", BICC Data Networks, 1987.
[2] VANCE, "ATS 1000, Network Analysis and Test Systems", VANCE Systems Inc.
[3] National Computing Centre, "Transport Layer Tester", COMMS-AID, The Networking Centre, 1987.
[4] RETIX, "RETIX NETBIOS Test/Validation Tool", RETIX The International Company, 1989.
[5] RETIX, "RETIX Session Test/Validation Tool", RETIX The International Company, 1989.
[6] RETIX, "RETIX Transport Test/Validation Tool", RETIX The International Company, 1989.
[7] ISO 8571/1-4 - 1988 FTAM Service/Protocol Specification.
[8] CCITT - 1988, CCITT: Message Handling Systems and Service Overview, International Telegraph and Telephone Consultative Committee Recommendation. X.400.
[9] ISO 8072/3 - 1987 Transport Service/Protocol/Specification
[10] ISO 8348/2 - 1987 Network Service/Protocol/Specification
[11] ISO 8802.2 - 1987 Logic Link Control Service/Protocol/Specification
[12] IEEE - 1985 Token Bus Access Method IEEE 802.4.
[13] ISO 8824 - 1987, OSI: Specification of Abstract Syntax Notation One (ASN.1).
[14] ISO 8825 (1987), OSI: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).
[15] Sun-3 Architecture, "A Sun Technical Report", Sun Microsystems, Inc. 1986.
[16] B. W. Kernighan and D. M. Ritchie, "The C Programming Language", Prentice-Hall, 1987.
[17] S. A. Farowich, "Communicating in the technical office", IEEE Spectrum, pp. 63-67, Apr. 1986.
[18] M. A. Kaminski, "Protocols for communicating in the factory", IEEE Spectrum, pp. 56-62, Apr. 1986.
[19] T. R. Marshall, "The ISO Development Environment", Vol. 1-4, The Wollongong Group, 1988.
[20] R. Matthews, K. Muralidhar and S. Sparks, "MAP 2.1 Conformance Testing Tools", IEEE Trans.On Software Eng., Vol. 14, No 3, pp.363-374, Mar. 1988.
[21] R. Matthews, K. Muralidhar and M. K. Schumacher, "Conformance Testing: Operational aspects, tools and experiences", Proc. 3rd Int. Workshop Protocol Specification, Testing and Verification. Amsterdam, The Netherlands: North-Holland, 1986.

Frame Name	S/E	FPDU	APDU	PPDU	SPDU	TPDU	NPDU	LPDU	MPDU
Connection req	S					CR	DT	UI	DT
Connection conf	E					CC	DT	UI	DT
Initialize req	S	INIRQ	AARQ	CP	CN	DT	DT	UI	DT
Initialize resp	E	INIRP	AARE	CPA	AC	DT	DT	UI	DT
Terminate req	S	TERRQ	RLRQ	RS	FN	DT	DT	UI	DT
Terminate resp	E	TERRP	RLRE	RSA	DN	DT	DT	UI	DT
Disconnect req	S					DR	DT	UI	DT
Disconnect conf	E					DC	DT	UI	DT

Table 1: *FTAM 001 Test Definition table.*

Frame Name	S/E	FPDU	APDU	PPDU	SPDU	TPDU	NPDU	LPDU	MPDU
Connection req	S					CR	DT	UI	DT
Connection conf	E					CC	DT	UI	DT
Initialize req	S	INIRQ	AARQ	CP	CN	DT	DT	UI	DT
Initialize resp	E	INIRP	AARE	CPA	AC	DT	DT	UI	DT
Select req	S	SELRQ		DT	DT	DT	DT	UI	DT
Select resp	E	SELRP		DT	DT	DT	DT	UI	DT
Deselect req	S	DESRQ		DT	DT	DT	DT	UI	DT
Deselect resp	E	DESRP		DT	DT	DT	DT	UI	DT
Terminate req	S	TERRQ	RLRQ	RS	FN	DT	DT	UI	DT
Terminate resp	E	TERRP	RLRE	RSA	DN	DT	DT	UI	DT
Disconnect req	S					DR	DT	UI	DT
Disconnect conf	E					DC	DT	UI	DT

Table 2: *FTAM 003 Test Definition table.*

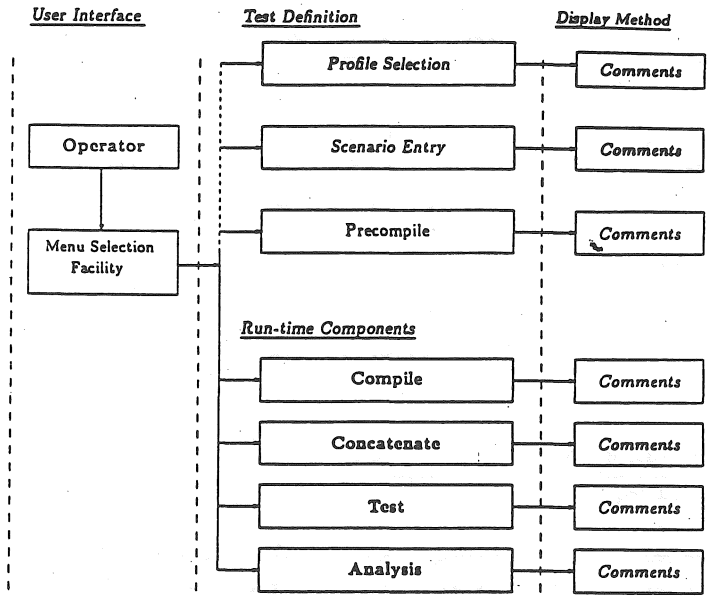


Figure 1: *Functional Overview of the Dynamic Tester.*

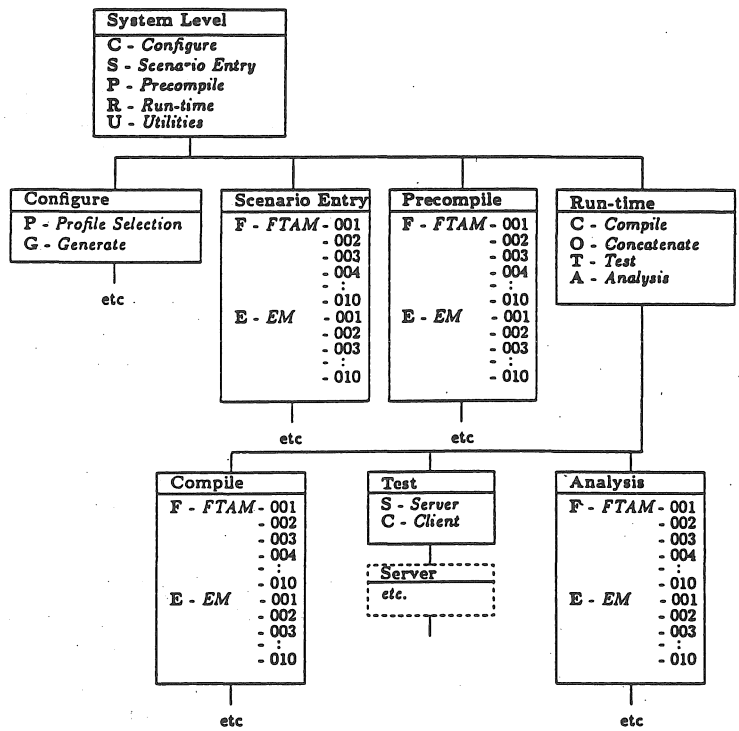


Figure 2: *Hierarchical menu of the Dynamic Tester.*

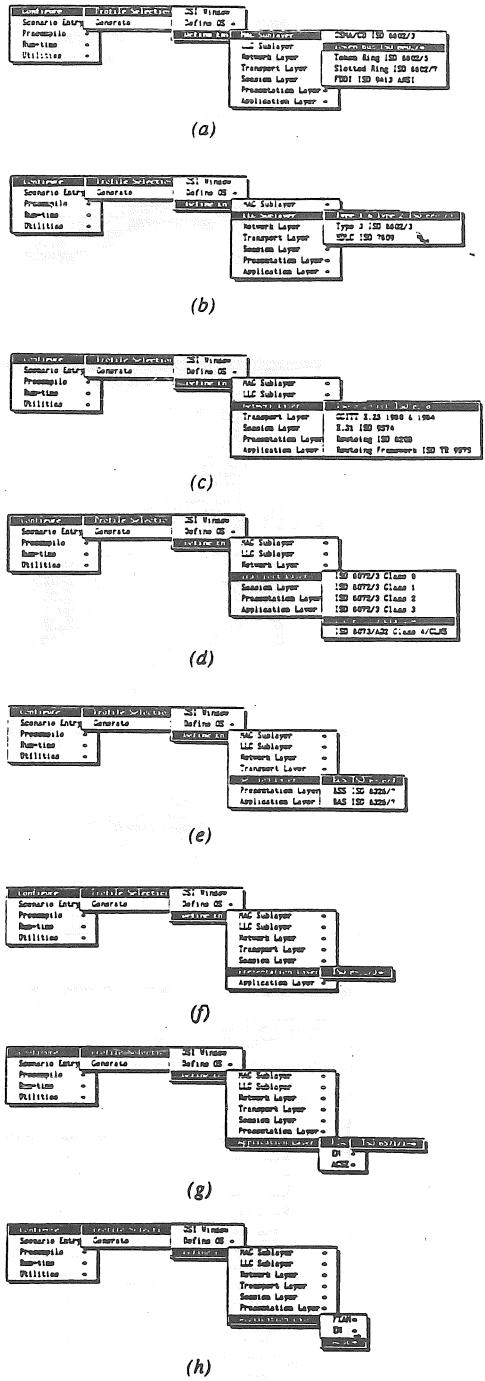
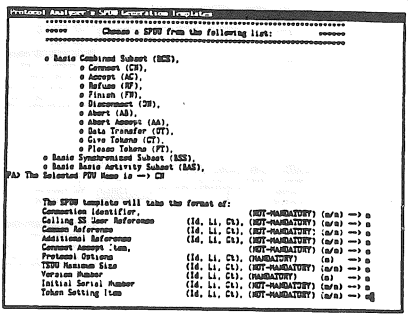


Figure 3: *Profile Selection: (a) MAC Sublayer; (b) LLC Sublayer; (c) Network Layer; (d) Transport Layer; (e) Session Layer; (f) Presentation Layer; (g) ACSE Application Entity; (h) FTAM Application Entity.*

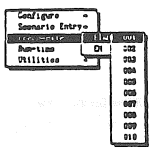
Figure 4: *Compile the selected profile.*

(a)

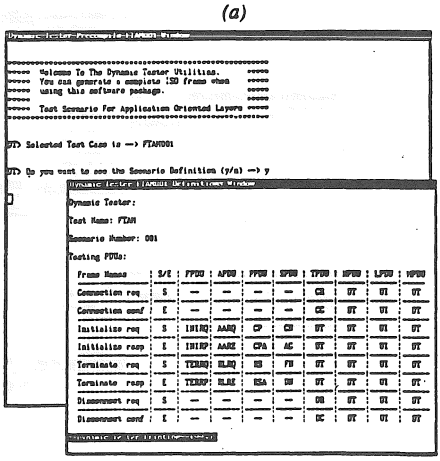


(b)

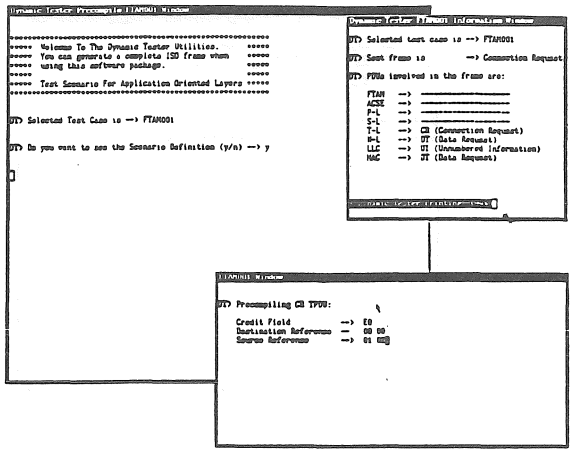
Figure 5: *Generate phase: (a) select Session Layer; (b) generate CN SPDU template.*



(a)

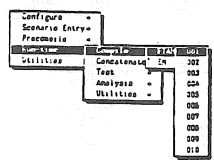


(b)

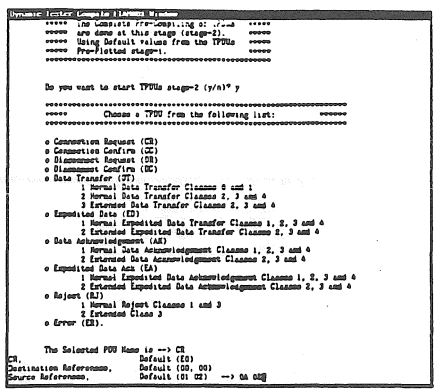


(c)

Figure 6: Precompile phase: (a) Select FTAM 001 scenario; (b) Test Definition table; (c) defining all the CR TPDU fields.

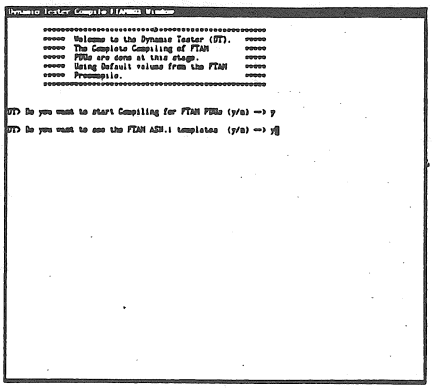


(a)

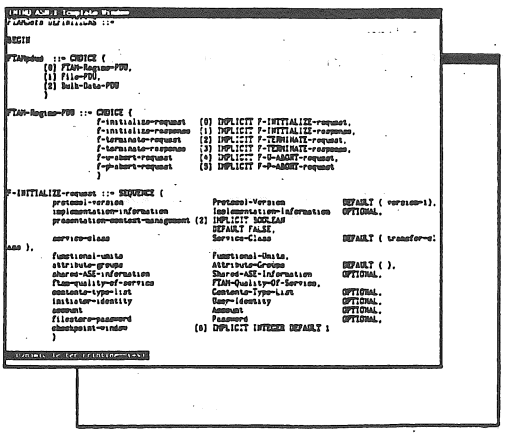


(b)

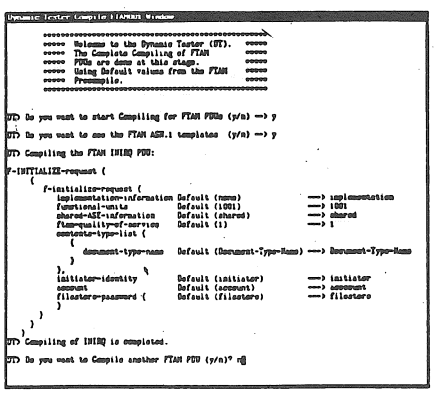
Figure 7: Compile phase: (a) Select FTAM 001 Scenario; (b) Define site specific fields of the CR TPDU.


Dynamic Tester Compile FTAM001 Window
Welcome to the Dynamic Tester (DT).
The Complete Compiling of FTAM
PDUs are done at this stage.
Using Default values from the FTAM
Precompile.
DT> Do you want to start Compiling for FTAM PDUs (y/n) --> y
DT> Do you want to see the FTAM ASN.1 templates (y/n) --> y

(a)


FTAMpdus ::= CHOICE {
[0] FTAM-Regime-PDU,
[1] File-PDU,
[2] Bulk-Data-PDU
}
FTAM-Regime-PDU ::= CHOICE {
f-initialize-request [0] IMPLICIT F-INITIALIZE-request,
f-initialize-response [1] IMPLICIT F-INITIALIZE-response,
f-terminate-request [2] IMPLICIT F-TERMINATE-request,
f-terminate-response [3] IMPLICIT F-TERMINATE-response,
f-u-abort-request [4] IMPLICIT F-U-ABORT-request,
f-p-abort-request [5] IMPLICIT F-P-ABORT-request
}
F-INITIALIZE-request ::= SEQUENCE {
protocol-version Protocol-Version DEFAULT { version-1},
implementation-information Implementation-Information OPTIONAL,
presentation-context-management [2] IMPLICIT BOOLEAN
DEFAULT FALSE,
service-class Service-Class DEFAULT { transfer-c
ass },
functional-units Functional-Units,
attribute-groups Attribute-Groups DEFAULT { },
shared-ASE-information Shared-ASE-information OPTIONAL,
ftam-quality-of-service FTAM-Quality-Of-Service,
contents-type-list Contents-Type-List OPTIONAL,
initiator-identity User-Identity OPTIONAL,
account Account OPTIONAL,
filestore-password Password OPTIONAL,
checkpoint-window [0] IMPLICIT INTEGER DEFAULT 1
}

(b)


Dynamic Tester Compile FTAM001 Window
Welcome to the Dynamic Tester (DT).
The Complete Compiling of FTAM
PDUs are done at this stage.
Using Default values from the FTAM
Precompile.
DT> Do you want to start Compiling for FTAM PDUs (y/n) --> y
DT> Do you want to see the FTAM ASN.1 templates (y/n) --> y
DT> Compiling the FTAM INIRQ PDU:
F-INITIALIZE-request {
f-initialize-request {
implementation-information Default (name) --> implementation
functional-units Default (1001) --> 1001
shared-ASE-information Default (shared) --> shared
ftam-quality-of-service Default (1) --> 1
contents-type-list {
{
document-type-name Default (Document-Type-Name) --> Document-Type-Name
}
},
initiator-identity Default (initiator) --> initiator
account Default (account) --> account
filestore-password { Default (filestore) --> filestore
}
}
}
}
DT> Compiling of INIRQ is completed.
DT> Do you want to Compile another FTAM PDU (y/n)? n

(c)

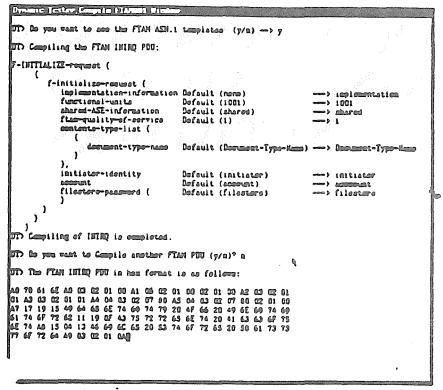


Figure 8: Compile phase: (a) Compiling FTAM INIRQ PDU; (b) FTAM INIRQ PDU definition; (c) Define site-specific fields; (d) Concrete syntax of INIRQ PDU.

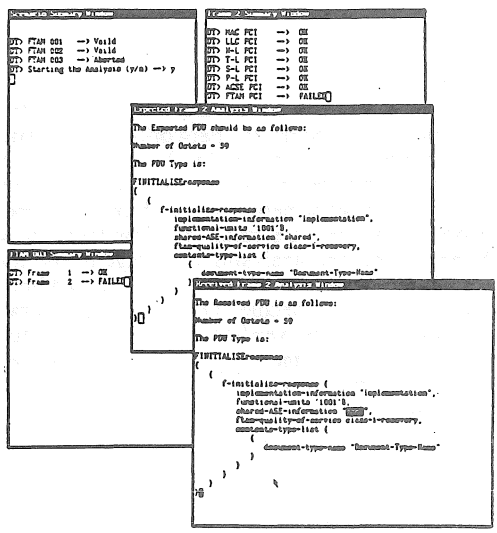


Figure 9: Analysis result of scenario FTAM 003.