AN EDITOR FOR ISO's TEST NOTATION FOR TEST SUITE MANAGEMENT

Srinivas Eswara

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering at
Concordia University
Montréal, Québec, Canada

January, 1989

ISBN 0-315-49046-2

ABSTRACT

An Editor for ISO's Protocol Test Notation for Test Suite Management

Srinivas Eswara

A software tool is developed to interactively edit protocol test specifications written in a notation called the Tree and Tabular Combined Notation (TTCN). The TTCN is a language developed by the International Standards Organization (ISO) to specify protocol conformance tests abstractly. The tool performs syntax checks on the input specification and translates it to the machine processable form, TTCN-MP. A method of automatic translation of test sequences obtained from a test generation tool, called CONTEST-ESTL, to TTCN and the execution of TTCN test suites is presented.

ACKNOWLEDGEMENTS

To Dr. Behcet Sarikaya, I would like to express my deepest gratitude for guiding and supporting me during the entire period of preparation of this thesis. I also acknowledge the part Dr. Sarikaya played in inculcating in me a deep interest and thirst for further knowledge in a field that is taking shape and promises to be at the leading edge of technology in this Information Age.

I would also like to acknowledge the help I received from Robert Bratulic, Raymond Knopp and Pham Oanh.

To my family and friends, I can only begin to express my heartfelt gratitude for all they have done for me and hope they will continue to do so in the future!

DEDICATION

This work is dedicated to the memory of my father

Dr. E.N.B. Sarma.

Table of Contents

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Introduction to ISO and OSI

Computer networks started gaining considerable attention around the world with the success of networks such as the ARPANET [1] and CYCLADES [2]. The commercial applications of such networks soon became apparent and techniques such as packet switching, satellite technology and local network technology, combined with the ever decreasing cost of hardware made these networks even more attractive. It also became clear that to take advantage of the tremendous potential of internetworking, standards would be required to make different networks from different vendors communicate meaningfully. The International Organization for Standardization (ISO) formed a technical committee (TC 97) to look into this issue. In 1978, a basic layered architecture which would constitute the framework for the development of communication standards was adopted as the basis for the development of standards for Open Systems Interconnection within the ISO. This basic model came to be known as the OSI reference model and became an International Standard (ISO 7498) [3] in the spring of 1983. It was also accepted by the International Telegraph and Telephone Consultative Committee (CCITT) as Recommendation X.200.

1.1 : The OSI Model and Layered Architecture.

The general architecture of a communicating system is shown in Fig. 1.1. The user interfaces with the top layer, and as such, it must be sufficiently abstract and independent of the physical characteristics of the connecting medium, the means of routing the information from sender to receiver, etc. The bottom layer takes care of interfacing with the medium, controlling the links and routing issues and other imple-

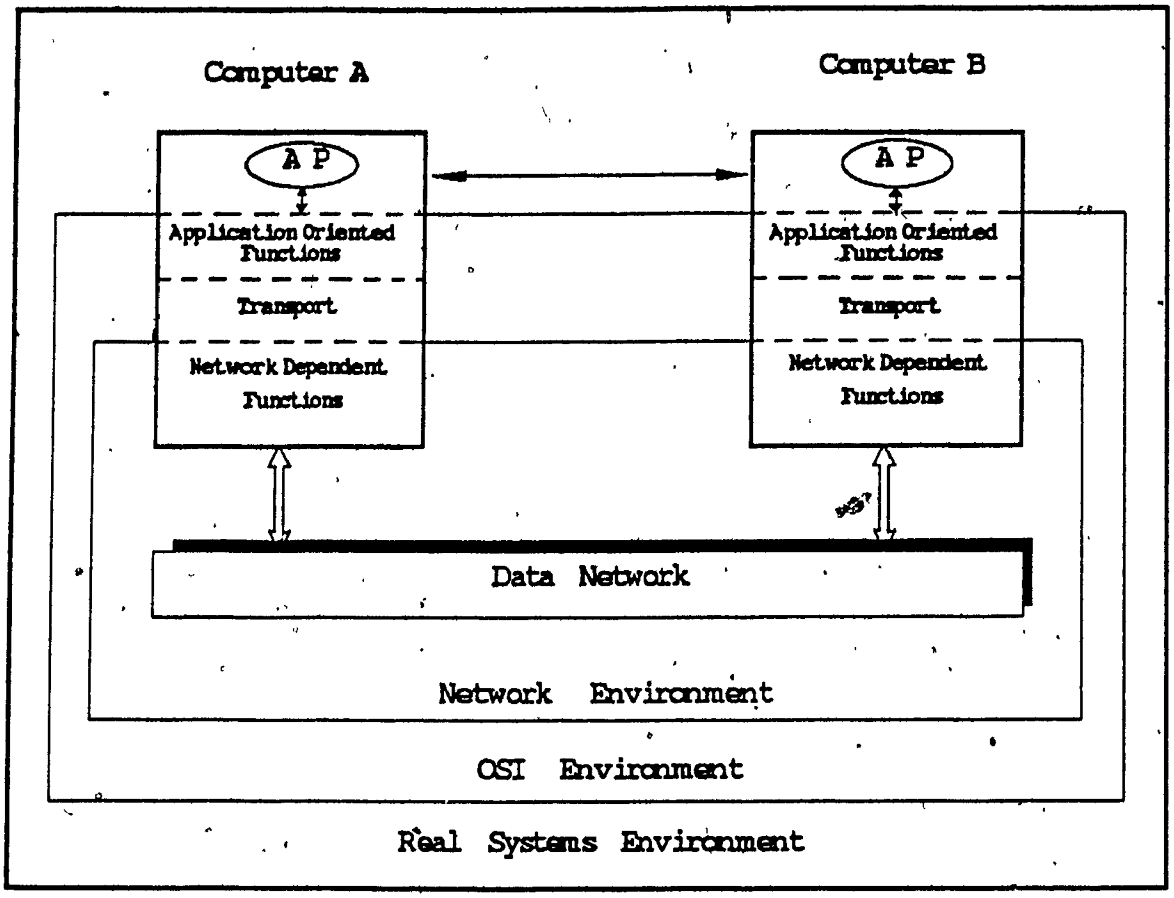

Computer A
Computer B
A P
A P
Application Oriented Functions
Application Oriented Functions
Transport
Transport
Network Dependent Functions
Network Dependent Functions
Data Network
Network Environment
OSI Environment
Real Systems Environment

Fig. 1.1. General Architecture of a Communicating System.

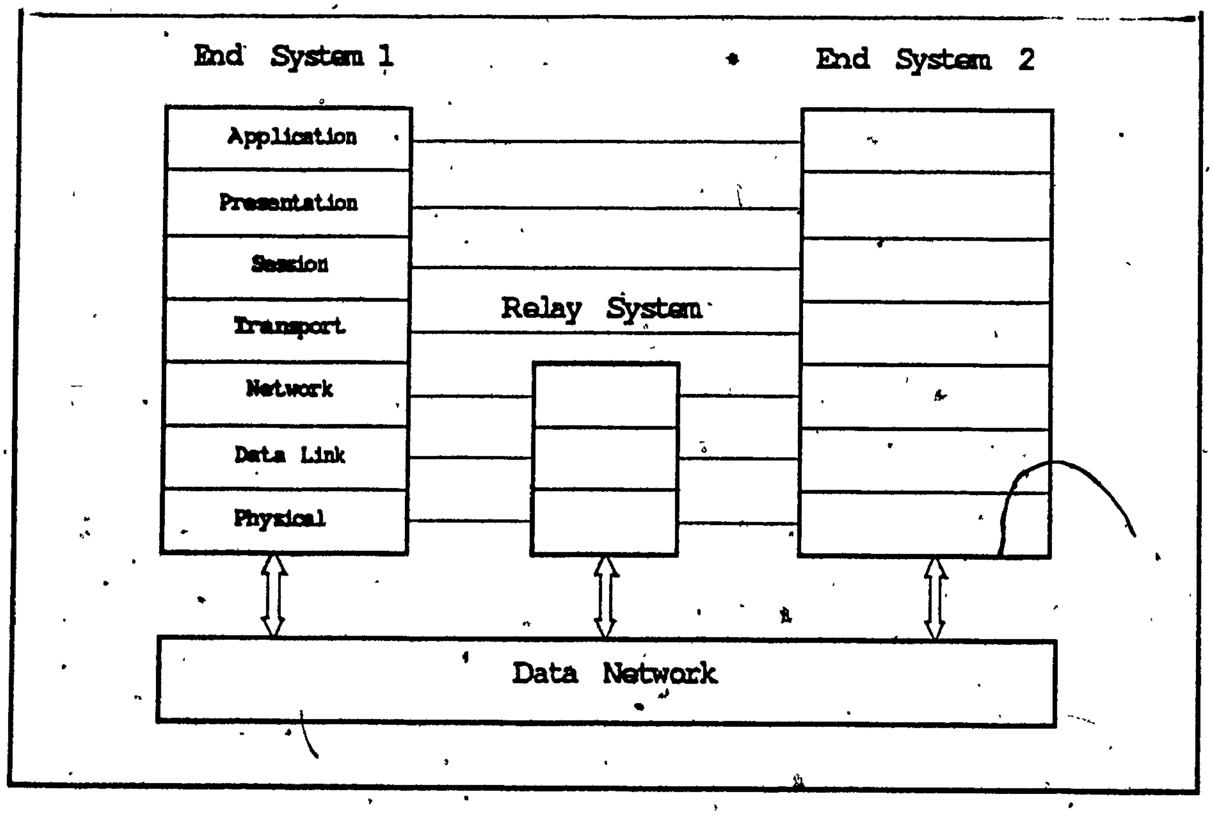


Fig. 1.2. OSI Model.

mentation dependent features. The transport layer serves as the bridge between the application oriented layers and the network oriented layers by providing an abstract service to the layer above. The application and network layers are further refined, to bring about the seven layer model as shown in Fig. 1.2. The boundaries between the layers have been selected as a result of the experience gained by previous standards activity.

1.1.1 : Protocols

Each layer performs a well defined function, operating according to a set of well defined rules. This is done by exchanging messages containing both data and control information with a corresponding (peer) layer in another system. These messages are referred to as *Protocol Data Units* (PDU) and the set of rules is referred to as a protocol.

The interface between each layer is well defined, with the layer above depending on the *services* provided by the layers beneath. Each layer can thus be implemented independent of the other layers, while providing a standard interface to the layer above. These layers exchange what are known as *Abstract Service Primitives* (ASP) as a means of specifying and using the services of the layer below. Conceptually, therefore, each layer communicates logically with a peer layer according to a defined protocol, but in reality, the (N)-PDUs are passed by means of the (N-1) services primitives provided by the next lower layer, coded as *(N-1) Service Data Units (SDUs).* (Refer to Fig. 1.3). Layer (N-1) encapsulates the (N-1)-SDU with its own *Protocol Control Information (PCI)* to form the (N-1)-PDU. A duality exists between services and protocols in that, if the conventional OSI model is rotated through 90 degrees, one finds that the communication between adjacent layers in the same system across a service boundary resembles the communication between peer layers (protocol) in the traditional sense.

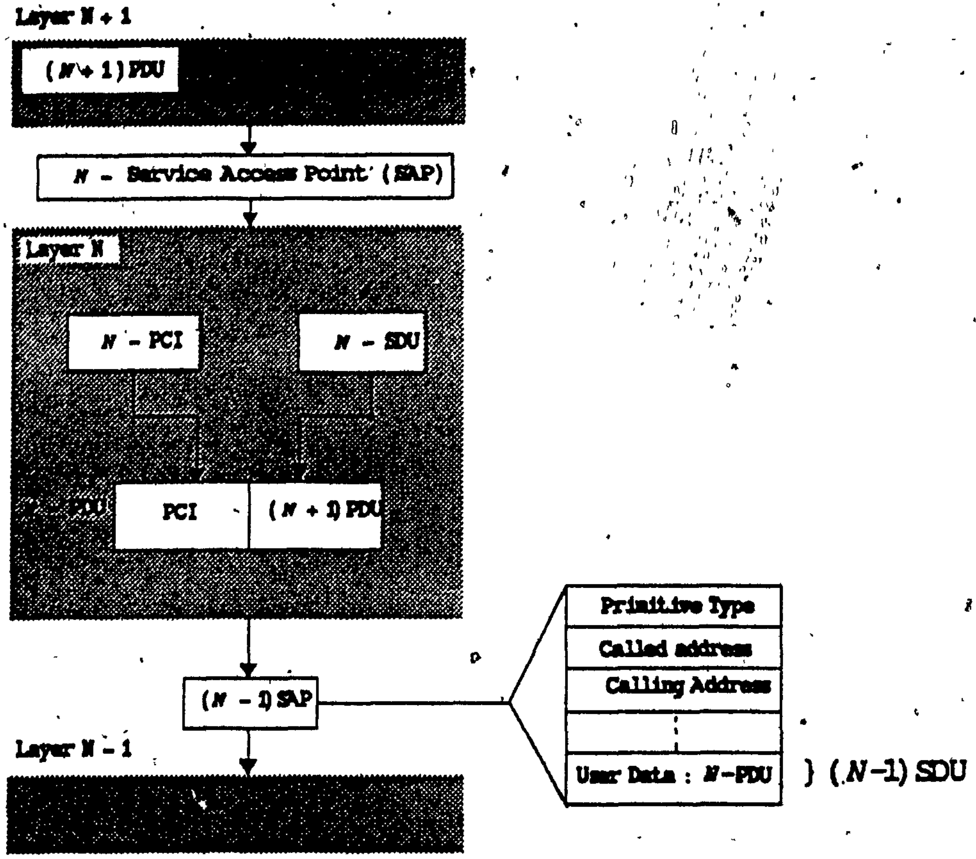


Fig. 1.3. SDUs, PDUs and ASPs.

1.1.2 : Logical Architecture

- **Application Oriented Layers**

These are the layers which are concerned with the protocols that allow two end user applications, to interwork. These applications are usually offered by the local operating system with distributed functionality, and whose mandate is to mask the networking details from the user to provide a clean interface, like any local service.

A. Application Layer

The application layer is primarily concerned with the semantics of the communication application. User applications reside in the application layer, although only the part of the process concerned with *interprocess communication* is in the real OSI system. Protocols such as file transfer, virtual terminal etc., all reside in this layer.

B. Presentation Layer

The purpose of the presentation layer is to shield the application processes from differences in data representation or syntax between the communicating entities.

C. Session Layer

The purpose of the session layer is to provide the structure for controlling the interactions between application processes. It does this by providing for full duplex or half duplex operation, establishment of synchronization points and the definition of special tokens to structure the information exchange.

D. Transport Layer

The purpose of the transport layer is to provide transparent transfer of data between end systems, even though the underlying network may be a series of concatenated disparate networks. This usually represents the boundary between the customer and the carrier in that it provides for optimal use of the network services and greater reliability than that supplied by the underlying services.

- **Network Oriented Layers**

E. Network Layer

The network layer shields the user from having to know the transfer technology and the details of routing in the case of concatenated networks. It also masks from the transport layer all the peculiarities of the physical medium transmission, be it optical fiber, cable, packet switched, satellite etc.

F. Data Link Layer

The data link layer provides for error free transmission of data *frames* for point to point and multipoint communications within the same network or local area network (LAN). Typical protocols are HDLC (High level Data Link Control) for point-to-point and multipoint communications and IEEE 802 for LANs. At this level, the protocol used depends very much on the characteristics of the transmission medium.

G. Physical Layer

The physical layer defines the mechanical, electrical and functional standards required of devices which access the physical medium.

1.2 : Scope and Contribution of thesis

The purpose of the standard committees is to allow implementors to unambiguously understand the requirements set forth for open systems internetworking. The claim by the vendors that their implementations conform to the necessary standards has to be investigated and with reasonable accuracy, be proven to be conforming. Thus we come to the main topic of this thesis, conformance testing and formal way to represent these tests. Chapter 2 defines the term conformance testing and describes the test methods and types of tests as defined in the standards, using example tests.

The standards presently describe the services and protocols which are required by open systems in an informal way, using English as the language of specification. The complexity of the systems involved soon made obvious the need to use formal techniques to specify these requirements, to accompany the English specification. So an ad hoc group on Formal Description Techniques (FDT) for communication systems was formed to investigate formal techniques suitable for protocol specification. In Chapter

3, the two FDTs which were developed and investigated by this sub group, *Estelle* and *Lotos* are explained briefly to serve as an introduction to later chapters, where Estelle is used in specifying test cases. The standardized test specification language, the *Tree and Tabular Combined Notation (TTCN),* is also introduced in chapter 3 and is the test specification language of interest in this thesis. Chapter 4 describes TTCN in detail, using an editor which was developed for this notation.

Chapter 5 explores the inner structure of the editor and its various functional units. In chapter 6, some example test specifications in TTCN are given and explained. The protocols under test are implementations of the application layer protocol, *File Transfer, Access and Management (FTAM),* and an implementation of the Class 4 transport protocol.

Chapter 7 describes a test generation tool based on an FDT (Estelle) and the conversion of its text output into test sequences in the standard TTCN notation. The execution of TTCN and the feasibility of conversion of TTCN test cases into Estelle modules to facilitate test execution are discussed in this chapter.

Chapters 1 through 3 represent a tutorial introduction to the basic concepts used in later chapters. Chapters 4 through 7 describe the original contributions of this thesis. The main thrust of this work is to integrate the editor for TTCN with test generation tools and to derive an executable representation of the test cases in order to produce a well rounded tool to aid test designers and implementors. In this direction, implementation of the algorithms described in the later chapters combined with the capabilities of the editor will go a long way towards making the goal of conformance testing of all ISO protocols possible.

CHAPTER 2 CONFORMANCE TESTING

INTRODUCTION

This chapter looks at the formal definition of the term conformance testing and its role in the process of creating open systems. The various methods and architectures for performing these tests are introduced here, using example test sequences for clarification. The term test suite is introduced along with the recommended structure of a conformance test suite.

2.1 : Conformance Testing

Conformance Testing involves testing the capabilities and behaviour of an implementation against the conformance requirements given in the protocol standard. Conformance testing is limited to testing the implementation within its capabilities as stated in the Protocol Implementation Conformance Statement (PICS - a statement by the implementor of the protocol regarding the capabilities and the supported features of the product) and does not include assessment of the performance, robustness or reliability. The purpose of Conformance testing is to increase the confidence that different implementations are able to internetwork.

An ISO standards document describes the general concepts of conformance testing, the methodology , types of conformance tests and a test specification language called Tree and Tabular Combined Notation (TTCN), which is explained in Chapter 3 [4]. In the next sections, we take a look at the types of tests and the test architectures recommended in this standard.

2.2 : Types-of Tests.

There are four types of tests, classified according to the extent to which each tests for conformance. These are

-Basic Interconnection Tests: Basic interconnection tests are done to verify that the IUT has enough conformance to go ahead with further testing. These tests are used for detecting severe cases of non-conformance, ie., the IUT is not even capable of interconnecting with the tester, etc.

-Capability tests: There are a set of static conformance requirements defined in the standard which define the minimum set of capabilities required by an implementation. Proper formation of PDUs, services offered by the IUT etc., fall in this category [5]. These tests provide testing of the static requirements and the capabilities claimed in the Protocol Implementation Conformance Statement (PICS) provided by the vendor.

-Behaviour tests: these test the IUT, within its stated capabilities, over the full range of conformance requirements as specified in the standard.

-Conformance Resolution Tests: are used to test, as near to definitive as possible, the IUT for particular requirements.

2.3 : Abstract Test Methodology

The test architectures defined in the standard (ISO 9646-1 & 2) all refer to an abstract test methodology based on the OSI reference model. The term end-system as used from now on refers to a full or partial implementation of the OSI model in a system in which the communication request/response originates/terminates as opposed to the concept of a relay system, whose function is to bridge different networks or provide a related service such as routing. Abstract test methods are described in terms of what outputs from the entity under test in an end-system are observed and what inputs to it can be controlled. They also identify the points closest to the Implementation Under Test (IUT) at which control and observation can be exercised. These points are called Points of Control and Observation (PCO), which usually are the Service Access Points (SAPs) of the different layers.

The OSI protocol standards define allowed behaviour of a protocol entity in terms of the PDUs and ASPs above and below that entity. Thus the conceptual testing methodology (Fig. 2.1) shows the entity or IUT as a black box with two abstractions for control and observation above and below the IUT. The Lower Tester (LT) is the abstraction of the method of providing control and observability at the lower boundary of the IUT. This is usually the location of the actual test procedures which control the test operation. It could be either at the end-system itself or located remotely, in which case, the LT relies on the (N-1)-service, which is assumed to have been tested previously. It should be noted here that the LT is, in some cases, a reference implementation of that layer along with exception generators, etc., although it may not be clear that this is the case from looking at the conceptual model.

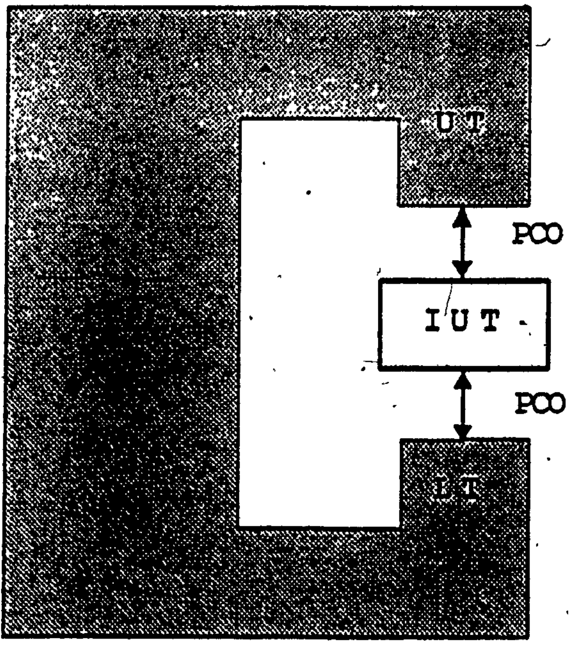


Fig. 2.1. Conceptual Testing Model.

The Upper Tester (UT) is the means of providing during test execution, control and observation at the upper service boundary of the IUT. The need for coordination between the upper and lower testers is apparent and the rules for such a cooperation are called the test coordination procedures.

Test architectures are also defined for IUTs where the interfaces to the layer are not visible externally and the testing procedures have to be designed considering the IUT as an embedded element in a multi-layer system. These methods are outside the scope of this thesis and all the methods defined here and later refer to single layer implementations.

2.3.1 : The Local Single Layer Test Method

The Local Single Layer test method (Fig. 2.2) assumes that the PCOs for the IUT are at the service boundaries above and below the layer. The test events are specified in terms of the (N)-ASPs , (N-1)ASPs and (N)-PDUs. The lower tester controls and observes the latter two test events. The requirements to be met by the test coordination procedures only have to be defined.

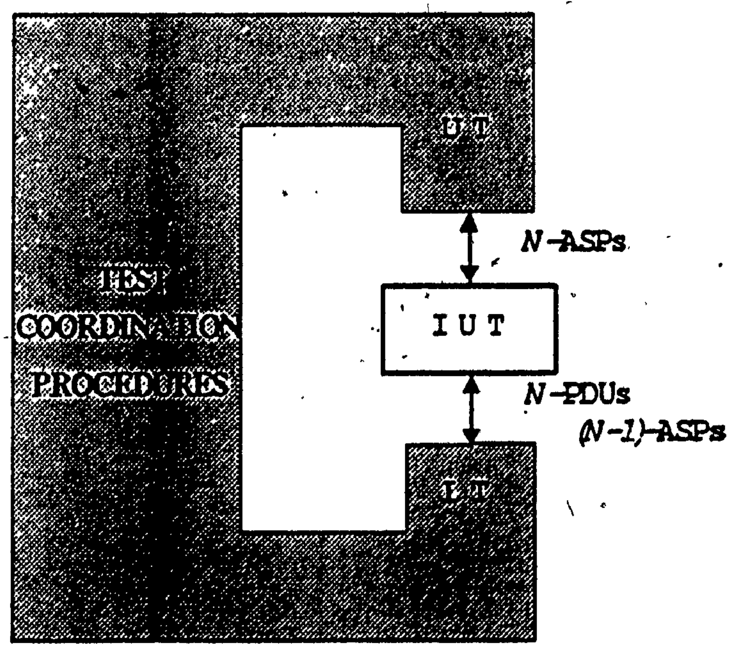


Fig. 2.2. Local Single Layer Architecture.

2.3.1.1 : An Example

Fig. 2.3 gives a time sequence diagram of the test events used in this example. The following is a test sequence intended to test basic connection refusal requirements of a transport implementation. *L* stands for the SAP at the lower tester and *U* stands for the SAP at the upper tester. This test assumes that the network connection has been established already and that it exists.

L sends NDataInd with the data part decoded as CR (Connect Request) PDU.

U should receive TConInd from IUT with the proper parameters.

U sends a TDiscReq --refusing the establishment request.

L should receive NDataReq with data part encoded as DR (Disconnection Request) PDU.

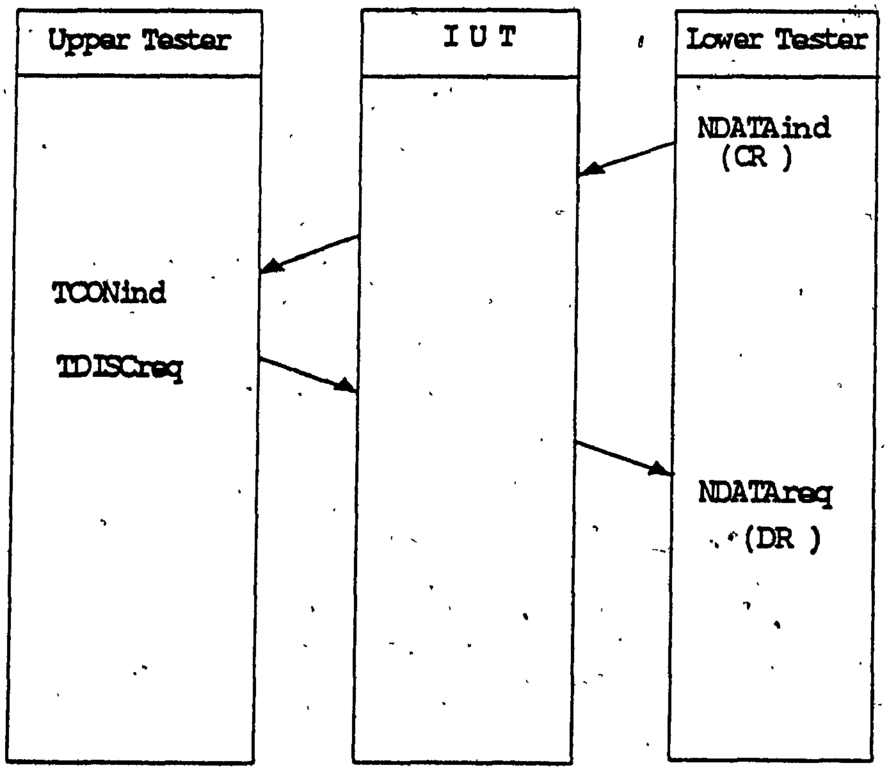


Fig. 2.3. Time Sequence Diagram.

2.3.2 : The Distributed Single Layer Test Method

The Distributed Single Layer abstract test method (Fig. 2.4) defines the PCOs as being the service boundaries above the IUT and above the (N-1)-service provider at the remote site. The test events are specified in terms of (N)-ASPs above the IUT and (N-1)-ASPs and (N)-PDUs remotely. These remote (N-1)-ASPs are referred to as (N-1)-ASP"s, to differentiate them from the local ASPs. This method overcomes the restriction of the Local method in that the test site can be remotely located, but the requirement of having an Upper Tester at the system under test is still present. Test coordination procedures are required, though only their requirements are needed to be defined.

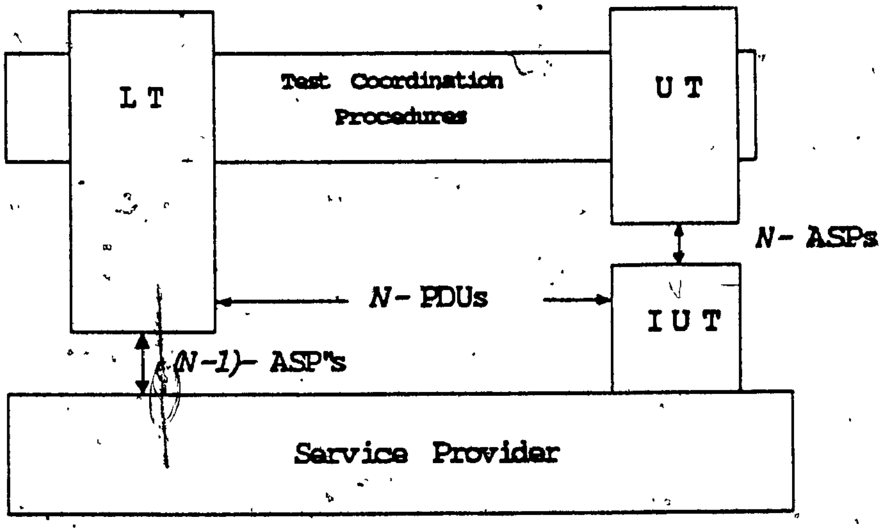


Fig. 2.4. Distributed Single Layer Architecture.

2.3.2.1 : An Example

The following is the same test case as the above modified for the distributed test architecture. It should be noted that the direction of indication and request to and from the Lower Tester (L) is reversed with respect to those in the local test architecture. The reason for this is the fact that there now exists a network between the IUT and the

Lower Tester.

L sends NDataReq with data part encoded as CR PDU.

U should receive TConInd with the proper parameters.

U sends TDisReq.

L must receive NDataInd with data part decoded as DR PDU.

2.3.3 : The Coordinated Single Layer Test Method

This is an enhanced version of the Distributed method in that the Upper Tester is standardized and a test management protocol is defined to realize the test coordination between upper and lower testers. These two need not be the same for all tests. The only PCO available is the one above the (N-1)-service provider at the remote SAP. The abstract model is shown in Fig. 2.5.

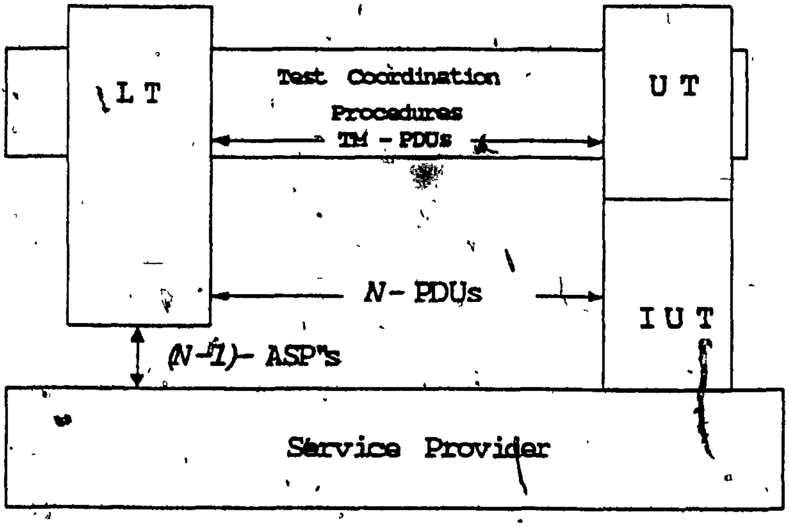


Fig. 2.5 Coordinated Single Layer Architecture.

2.3.3.1 : An Example

The following is the same sequence used in the Local method but modified for the Coordinated method. Here the initiation is done by the lower tester trying to establish a transport connection using TM-PDU (Test Management PDUs) for coordination between LT and UT. Each TM-PDU has a special significance understood by both Lower tester and Upper Tester.

L sends NdataReq with data encoded as CR PDU whose data is encoded as Disconnect TM PDU.

–This tells the Upper tester to refuse the connection requested.

L should receive NDataInd with data part encoded as DR PDU.

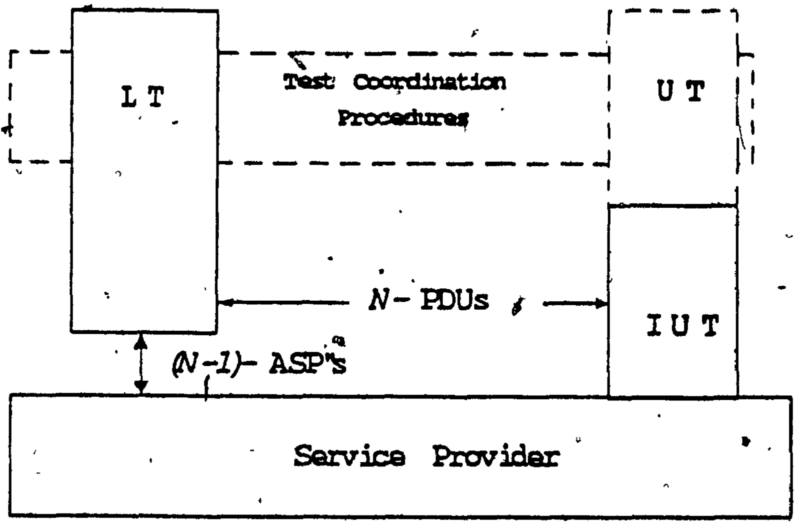


Fig. 2.6. Remote Single Layer Architecture.

2.3.4 : The Remote Single Layer Test Method

As can be seen from Fig. 2.6, the Remote Test method specifies the PCO as being above the (N-1)-Service provider at the SAP remote from the (N)-Entity under test. No access is assumed of the layer above the IUT. Test coordination procedures need to be informally specified only. Test laboratory and the client have to negotiate the method by which actions to be initiated by the IUT need to be specified and executed. The X.25 tests [6][7] which are being standardized are specified in terms of this method.

The example for this method is the same as the the one given for the coordinated method with the difference that, since the coordination procedures are informal, there will be no TM PDUs as such. Coordination between the tester at the remote site and the one at the site has to be done using procedures such as telephonic coordination.

2.4 : Test Suites and their Structure

An ISO test suite is made up of all the tests that are necessary to perform conformance testing or basic interconnection testing for an IUT or a protocol within an IUT, along with information about the order in which the tests are to be executed. These test suites are hierarchically structured into *test groups* and *test cases*. (Fig. 2.7). These along with a library of *test steps* form an ISO test suite. A test suite can either be structured as test groups and test cases under these groups or just a flat specification of all the test cases but not both. Tests which achieve related purposes are structured into sets called groups. These in turn can be divided into groups until we come to test cases which achieve a specific test purpose such as verifying whether the IUT can support the packet sizes as stated and so on. These test cases have references to test steps which are grouped into a library to be called from anywhere. They are used to modularize test cases. Test Cases can also make use of locally defined test steps. These test steps are invisible to other cases. A *test event* is the smallest test unit which can be specified, for example, the sending or receiving of an ASP. Verdicts of the test case are assigned at this level.

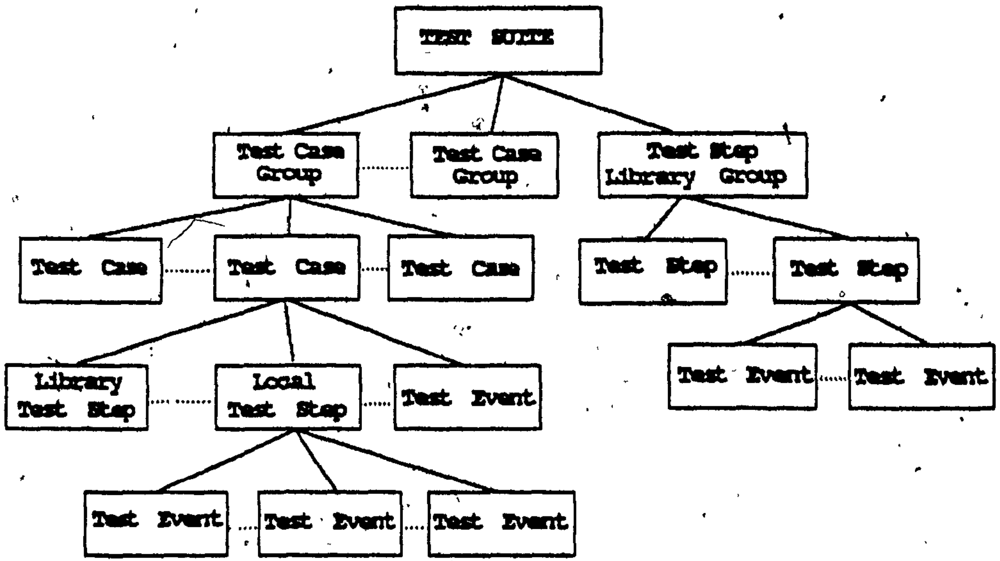


Fig. 2.7. Test Suite Structure.

The following example is given to clarify protocol testing in an OSI environment. The following structured sequence of tests in the test group would be specified in the same manner in TTCN. The purpose of the group is to test the response of the IUT to valid peer behaviour. This is further divided into three (sub)groups.

1. The connection establishment tests.

2. The data transfer tests.

3. The disconnection tests.

The first sub-group can be further divided into three groups.

1.1 Events sent to the IUT.

1.2 Events received from the IUT.

1.3 Interaction events between the IUT and peer.

Group 1.1 can be divided into test cases as given below.

1.1.1 Varying test events for each state of the IUT

1.1.2 Timing variations.

1.1.3 Varying the encodings

etc. This is a typical breakup of a test suite as recommended in [4].

There are three levels of defining test cases. They are :

- A generic test case is defined in the style of either the Remote or Distributed test methods. An initial state of the test body is defined and the test body is specified using, preferably, TTCN. No attempt is made to give implementation details or how to drive the test body into the required initial state, etc. Generic test cases produced before the abstract test cases are useful in the design process because they give a common structure to all possible abstract test suites, independent of the test method used. The abstract test suites derived from the generic test suite will have similar test purposes, though the means of achieving them may be different.
- An abstract test case is specified in terms of a particular test method and an additional requirement is that a *preamble* and *postamble* be specified for each test case to drive the test body into the required initial state and to take it back to a chosen stable state. The body of test case may be different from that of the corresponding generic test case if the test methods are different.
- An executable test case, derived from an abstract test case, is defined in an executable form to be used on a real system without further processing. For example, in an abstract test suite defined in TTCN, the test designer is allowed to use user-defined operations without having to specify how to realize them.

CHAPTER 3 FORMAL DESCRIPTION TECHNIQUES

INTRODUCTION

This chapter looks at the role of formal description techniques (as opposed to an informal English language description) in specifying the protocols and services required of systems in the OSI environment. The techniques of interest here are *Estelle, Lotos, ASN.1* and *TTCN.* A brief description of these specification languages will be given and their underlying models explained.

3.1 : Formal Description Techniques (FDT)

It was apparent from the inception of the Open Systems concept that formal techniques had to be used to describe these protocols and services so that implementors anywhere could develop correct and interworkable systems. The ISO / TC97 / SC 16 / WG1 ad hoc group on Formal Description Techniques was formed for this specific reason.

The following, which is taken from Annex E of an early version of the Reference Model, show that an FDT is necessary to provide :

an *unambiguous, clear* and *concise,* specification of a service, protocol, interface of the reference model;

a basis for determining the *completeness* of the specification;

a foundation for the *analysis* of the specification as to its correctness, efficiency, etc;

a basis for the *verification* that the specification meets all the requirements of the OSI;

a basis for determining the *conformance* of implementations to the OSI standard specification;

a basis for determining the *consistency* of OSI standards with each other;

a basis for developing *implementation* support.

In this chapter, we take a look at the FDTs which are undergoing the process of standardization, specifically, *Estelle* and *Lotos*. These FDTs are oriented towards specifying open systems but do not lend themselves easily to specifying test procedures. Hence, a new notation, the *Tree and Tabular Combined Notation (TTCN)* is presently being standardized and it is expected that test cases for implementations will be specified in TTCN test suites.

3.2 : Estelle

A subgroup of WG1, subgroup B, was formed to deal with the development of an FDT based on the Extended Finite State Machine concept. The underlying model is a finite state automaton which has memory and the state space of which is determined by a set of variables; a state is the set of of the different values which these variables can take. An important variable is the **STATE** variable which is the state of the finite automaton. The variables are also referred to as **context variables** to differentiate them from the STATE variable, also called the **major state.**

This model describes the system or protocol as a collection of entities or **modules.** Each module is a finite state machine capable of having memory and communicate with each other as well as the external environment over **channels.** These channels operate as First In First Out (FIFO) queues. The language of Estelle is based on Pascal with some extensions to facilitate protocol specification. **WHEN** clauses are used to specify the arrival of an input interaction and **FROM/TO** clauses are used to define the initial and final state of the machine. **OUTPUT** statements are used to specify the output of a **transition** and **PROVIDED** clauses are used to specify conditions required to fire that transition. Transitions with no WHEN clause are called **spontaneous** and are used to describe nondeterminism. **DELAY** clauses are used to specify timing requirements. The collection of the transitions and the relevant declarations make up the body of a module.

It is possible to semi-automatically generate executable code using translators developed for this purpose [8]. The present definition of Estelle has reached International Standard status. [9].

3.3 : Lotos

Lotos is an algebraic specification based on the **calculus of communicating systems (CCS)** of Milner [10][11]. CCS is used to define the observable behaviour of processes. Lotos incorporates a powerful abstract data type description language called **ACT ONE** [12]. ACT ONE is used to define interaction primitives as well as static data. The dynamic behaviour is expressed as a process algebra as a set of operations which include recursive procedural calls to instantiate multiple copies of the same process. The interaction type is the **rendezvous** type with value passing and value matching at the synchronization points. Further details can be found in [13].

3.4 : The Abstract Syntax Notation One (ASN.1)

ASN.1 [14] has been defined as an *abstract* syntax for the meaningful transfer of data between two application entities. This abstract syntax defines the structure of the PDUs to be transferred but the **concrete** syntax may be different on different machines. The presentation layer associated with each application entity performs a suitable transformation to and from this syntax if it happens to differ from the local syntax. ASN.1 also defines an **encoding** method to produce the PDU in a standard syntax form and can also be used to define the service primitives and PDUs of other layers.

A, definition in ASN.1 of a set of PDUs belonging to a particular protocol entity is defined as a **module.** The definition of the module body is placed between **BEGIN** and **END** statements and successive definitions are given using a tree structure with the leaves being one of the predefined types, either **primitive** (integer, boolean, bitstring, etc.) or **constructor** (integer, bitstring, sequence, ASCII character strings, etc.) types.

3.5 : Requirements for a Test Notation

To construct an abstract test suite, a test notation is needed to standardize the representation of a sequence of test steps or test cases. The necessity of introducing a new test notation instead of using existing formal description techniques may be questioned. The reasoning behind the introduction of TTCN as the standard test notation can be seen by considering the criteria for an FDT to be suitable as a test notation.

3.5.1 : Evaluation criteria for a test notation.

- Does the test notation have the right expressive power? This criterion includes such categories as emphasis on the main paths of the test, specifying timing events, specifying related verdicts to each paths of the test ,etc.
- Is the test notation equally applicable to all test architectures?
- Is the test notation easily readable, clear, understandable and simple? This raises questions such as how much complexity must go into the language and how much into the description.
- How can the test notation assist in deriving executable tests from abstract test specifications?
- Is the test notation suitable in other stages of the testing process?

- Is the test notation widely used and stable? Do experience and documentation exist? In this respect, Lotos and Estelle have a decided advantage over any new notation.
- Does the test notation support the validation of tests?
- How can the test notation assist in deriving tests for one architecture from tests for another architecture? For example, deriving the remote or distributed architectures from the local architecture. Of course, this may not be necessary in all cases.

Taking the above criteria into consideration, ISO has recommended TTCN as a preferred test notation. TTCN is a semi-formal notation with algorithmically defined semantics. The defined purpose of TTCN is :

a) to provide a common reference for assessing other test notations and to assist in examining the problems arising in test case and test suite design;

b) to provide a basis for the translation of test cases into other test notations.

3.6 : Syntax Forms of TTCN

TTCN is provided in two forms, graphical and machine processable (TTCN-GR and TTCN-MP). The TTCN-GR notation is defined using a number of table proformas. TTCN-GR is geared towards human readability and understanding. TTCN-MP is a linear form of TTCN-GR with the difference that tokens serve as delimiters between fields instead of tables. TTCN-MP is more suitable for machine processing, such as communicating test suites between different sites or for further automated processing of a test suite such as the translation of test cases into other test notations. Examples will be given in the next Chapter showing tables using the TTCN-GR notation and its equivalent TTCN-MP form to make the distinction clear.

3.7 : TTCN Test Suite Structure

As was seen in Chapter 2, a test suite for an OSI implementation is considered to be structured hierarchically into:

- Test Groups.
- Test Cases.
- Test Steps.
- Test Events.

This same structure is preserved in TTCN in that it allows the partitioning of the test suite into a hierarchical grouping by allowing global referencing of test cases in terms of the name of the suite, the component sub-groups and finally, the case itself. The test suite is also permitted to be completely flat with no groups, if so desired. Test Steps can be referenced in the same way to permit a library of steps to promote modularity in suite design.

3.7.1 : Components of a TTCN Test Suite

The following four sections make up a TTCN test suite :

- Suite Overview

The Suite Overview part contains information needed for overall understanding of the test suite and its purpose. All the test case, test step and default references have to be entered here. The structure of the reference determines the position of the particular case or step in the suite hierarchy.

- Declarations Part

The Declarations section contains information about the set of test events and all components which are used in the test suite. There are eleven different declarations which can be made in a test suite. The three sorts of test events defined here are: *abstract service primitives (ASPs)* which occur at the *PCOs* used by the UT and LT and timer events.

The other test suite components specified here are the user defined types, user defined operators, test suite parameters, global constants, global variables, PCOs, and PDU (Protocol Data Unit) data types. PDUs carry the information from peer layer to peer layer. The user defined types are for the types which do not fall in the predefined types in TTCN such as integer, boolean, bitstring, octet, etc.

- Dynamic Part

The dynamic behaviouf part of a test suite consists of the main body of the suite. The test cases, steps and default behaviour are defined here. The tree notation is used here to describe the sequence of events which can possibly occur in a test case or step or default. In TTCN-GR, indentation is used to show what events can occur at a particular instant of time. Events at the same level of indentation are alternative events at that instance of testing and events occurring one after another are shown line after line, successive events being indented once from left to right. For example

Events in sequence:

```
EVENT_A
  EVENT_B
    EVENT_C
```

Alternative events and events in sequence:

```
EVENT_A
  EVENT_B
  EVENT_B1
    EVENT_C
```

EVENT_B and EVENT_B1 are the set of possible events which can occur after EVENT_A. Since EVENT_C is given after EVENT_B1, it is understood that EVENT_C can occur only if, out of the two possible events at the second level of indentation, EVENT_B1 occurs.

- Constraints Part

This section specifies values for the ASPs, PDUs and their parameters which are used in the dynamic behaviour as the test events.

These four components will be explained in more detail in the next chapter along with the editor which was created to enable the test designer enter the test suite in the graphical format and automatically get the TTCN-MP form, which is required for further processing.

CHAPTER 4 THE EDITOR AND TTCN

INTRODUCTION

This chapter introduces the tool developed as an aid to the testing process and uses the TTCN language to explain the various features of the tool. The four parts of TTCN are shown to be supported completely by the tool and some example proformas are given.

4.1 : Survey of Language Editors

An editor, in computer terminology at least, is a system program that allows the user to interactively enter and modify his/her programs or text, as the case may be, providing maintenance and structure and relevant filing information. As programs get more and more complex, the major problem facing programmers is managing and maintaining large software systems. The concept of *passive* editors, which just accept anything the user types in without regard to the content or structure, is generally not found to be useful enough in the software development process. Lengthy compilation time also adds to the problem because the user has to re-edit if the compilation came up with any syntax errors.

Recent work has focussed on language oriented editors which are armed with a knowledge of the language's syntax, and in some cases, the associated semantics. This knowledge can be used to detect either local (context free) or global (context sensitive) errors as they are committed. [15] discusses the development of such a system called the Synthesizer generator. [16] discusses the guidelines for developing systems capable of generating language oriented editors (meta environments). WYSIWYG (What You See Is What You Get) editors also belong to this class, cutting down considerably on the time required for document preparation.

ITEX [17] is a commercial system under development by Swedish Telecom for the maintenance and execution of test suites specified in the TTCN notation. This system has been implemented in the LOOPS object oriented programming environment. It consists of a *syntax driven* editor for entering TTCN suites and a translator module for translation of the suite into various executable languages. Syntax driven means that syntax checking is performed as the user enters the various parts of TTCN. This checking can also be turned off. The system is going through the process of being ported to the UNIX and SUNVIEW environment.

4.2 : The Editor

The process of designing a test suite to test implementations which claim conformance to a particular standard is a very complex one with no recognized procedure for test generation, automatic or otherwise. There are many techniques available, each with its own disadvantages and advantages. Once the designer has decided upon a particular method to generate the test sequences, his/her next objective is to represent these tests in the standard notation to achieve wide acceptance and approval of the suite. Having a tool which can be useful at this step of the design process is very desirable indeed. This is the motivation behind the development of an editor which

- creates the tabular forms in a user friendly environment, on which the test suite specification could be entered
- creates the hard copies of the tables required
- converts the specification to TTCN-MP after syntactic and semantic checks
- takes a specification in TTCN-MP and allows the user to view the TTCN-GR

form of the specification.

The last two steps are required when one needs to exchange test specifications with other test sites because TTCN-MP is the transfer syntax of choice.

4.3 : Editor Structure

The editor, nick-named CONTEST-TTCN has been implemented entirely in the C language in the SUNVIEW environment on SUN workstations running the SUNOS 3.0 and up. It is menu driven with a simple user interface and requires a minimum of prior exposure to the environment in which the tool operates. Fig. 4.1 shows the structure of the tool. The function of each module is explained in detail in the next Chapter.

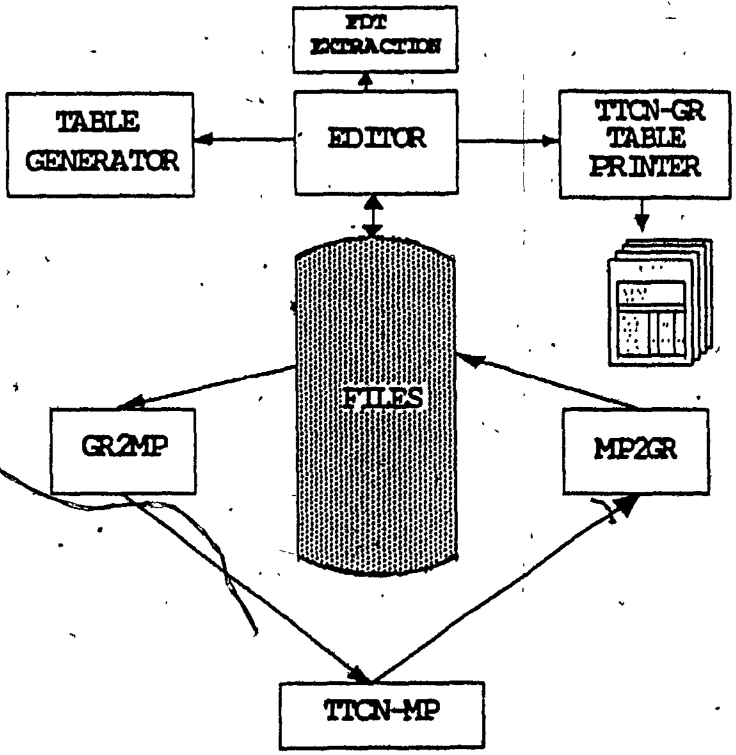


Fig. 4.1. Tool Structure.

4.4 : Components of TTCN

The different components of the language will be explained, taking example declarations and behaviour descriptions. The form of TTCN used here is TTCN-GR.

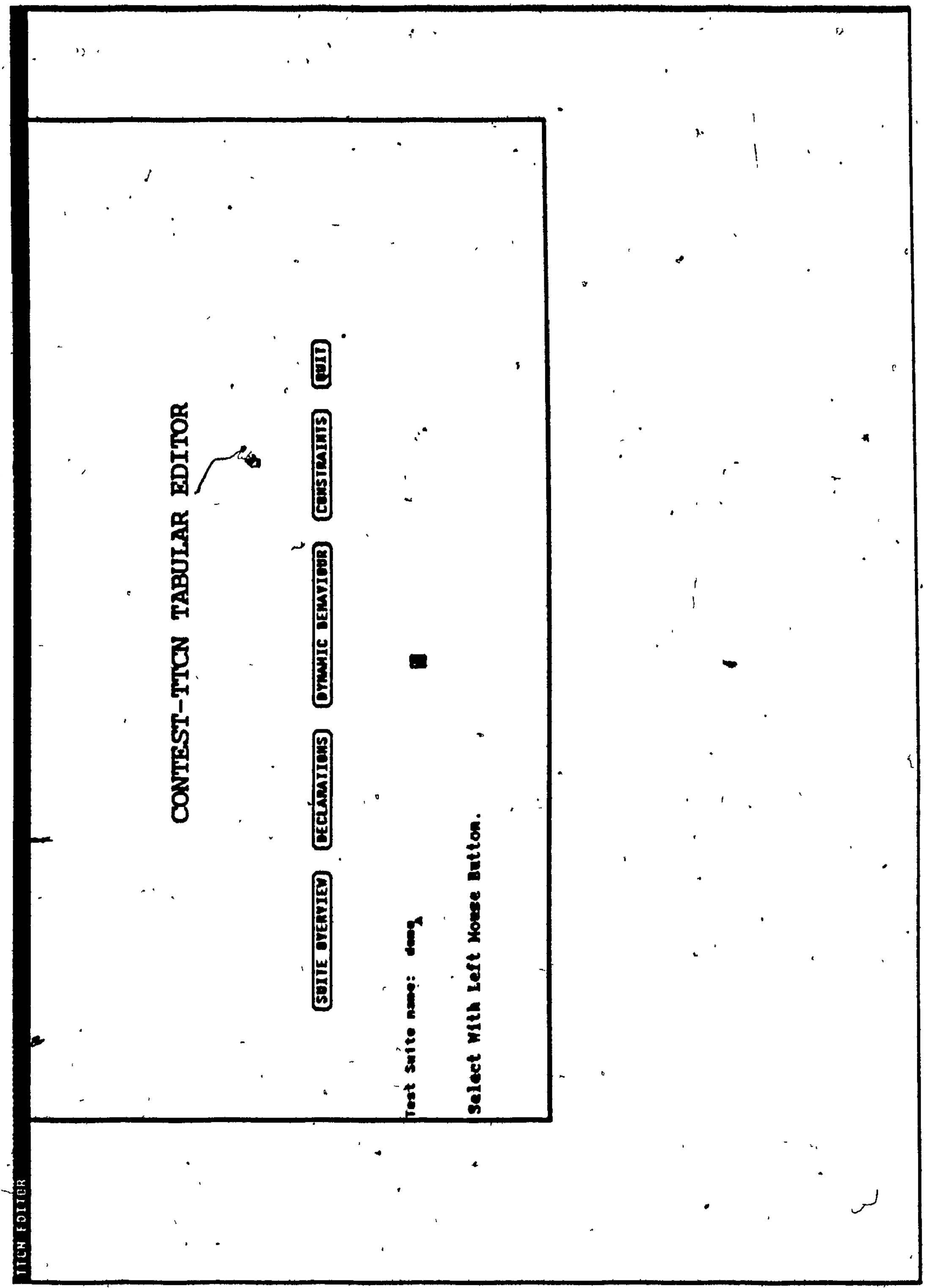
4.4.1 : Suite Overview

Fig. 4.2 shows the initial screen when the user invokes the editor. The screen representation of the suite overview table is shown in Fig. 4.3. This corresponds exactly to standard proforma as defined in [4]. Mouse location dictates which field or column is getting the input. Moving the mouse into the top panel, the user can enter the suite identifier, Protocol Implementation Conformance Statement (PICS) reference, the test method used, etc. Moving the mouse to the rightmost column below this panel, the user can enter the test case identifiers and the test case references in the next column. As can be seen from the figure, the menu which is called up when the mouse button is clicked anywhere on the frame contains an item called *proforma*. This is used to change the proforma for test steps and defaults to enter the information for these two components of the Dynamic Behaviour.

Selecting the *hardcopy* item on the same menu produces the table shown in Fig. 4.4, which is the standard proforma for the suite overview part of a test suite.

4.4.2 : Declarations

This section contains information about the set of test events and all the components which are used in the test suite. There are ten different declarations which can be made in a test suite. These are explained one by one using examples.

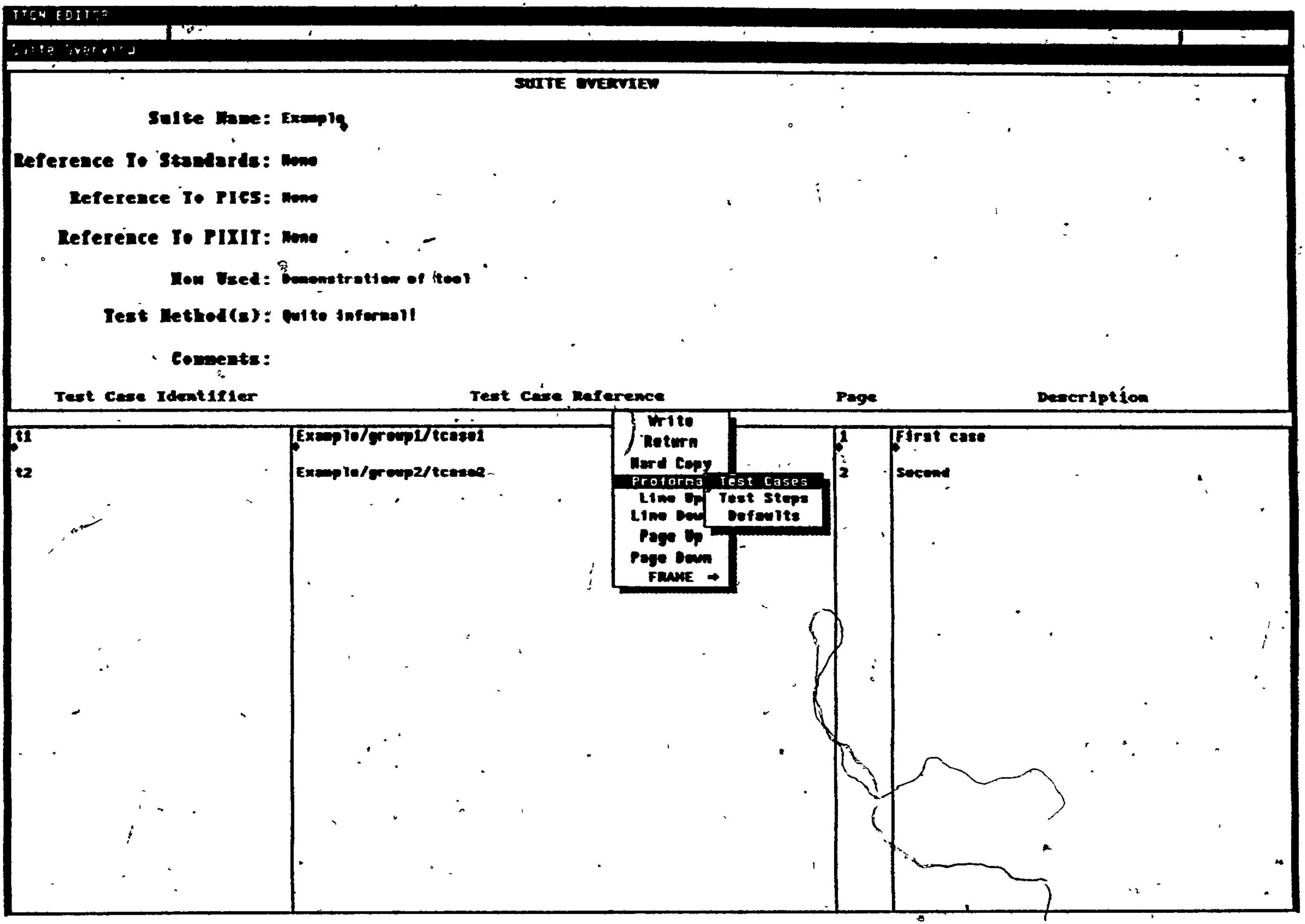


CONTEST-TTCN TABULAR EDITOR

Test Suite name: demo

Select With Left Mouse Button.

Fig. 4.2. Initial Screen of the tool.



TTCN EDITOR

Suite Overview

SUITE OVERVIEW

Suite Name: Example

Reference To Standards: None

Reference To PICS: None

Reference To PIXIT: None

How Used: Demonstration of tool

Test Method(s): Quite informal!

Comments:

Test Case Identifier	Test Case Reference	Page	Description
t1	Example/group1/tcase1	1	First case
t2	Example/group2/tcase2	2	Second

Fig. 4.3. Screen representation - Suite Overview.

SUITE OVERVIEW	
Suite Name:	Example
Reference to Standards	None
Reference to PICS	None
Reference to PIXIT	None
How Used:	Demonstration of tool
Test Methods	Quite informal!
Comments	

Test Case Identifier	Test Case Reference	Page	Description
t1	Example/group1/case1	1	First case
t2	Example/group2/case2	2	Second

Test Step Identifier	Test Step Reference	Page	Description

Default Identifier	Default Reference	Page	Description

Fig. 4.4. Suite Overview Example Table.

Abbreviations

These are used to define strings which are frequently used in the dynamic behaviour as a short identifier or to simulate sending or receiving PDUs as test events instead of ASPs. In the example table, (Fig. 4.5), CR-TPDU may be used instead of NDataInd [UserData~CR] where ~ represents "decoded as" in TTCN.

ABBREVIATION DECLARATIONS		
Abbreviation	Expansion	Comments
CR-TPDU	?NDataInd [UserData~CR]	CR-TPDU may be used in the Behaviour
CC_TPDU	?NDataInd [UserData~CC]	Tables

Fig. 4.5. Abbreviations Example Table.

Operators

Operators which are specific to a test suite may be introduced by the user if the predefined arithmetic, relational and boolean operators do not satisfy all the designer's needs. In order to do so, the following information needs to be provided :

- a name for the operation

- the signature of the operation which is made up of

  a list of the input types

  a name for each input argument

  the type of the result.

- a textual description of the operation, i.e., only an abstract functional description without any implementation details.

Side effects are not allowed in operations, i.e., the input arguments are unchanged by the operation. Fig. 4.6 is an example operation definition of *substr* which takes an ASCII string as input along with two integer arguments, *start* and *length* and produces a string of length *length* starting from *start* of the original input string.

OPERATION DEFINITION	
Operation Name	: substr
Argument Types	: IA5string, integer, integer
Arguments	: source, start, length
Result Type	. IA5string
DESCRIPTION	
substr(source, start, length) is the string of length length starting from index start of source	

Fig. 4.6. Operators Example Table.

<u>User Types</u>

TTCN supports a number of predefined (base) types such as **integer, boolean, bitstring** etc., and also provides a mechanism to allow the user to define types specific to a test suite. These types can be defined in two ways :

- TTCN tabular method
- ASN.1 method

An example TTCN type definition is shown in Fig. 4.7. The column labeled **Base Type** may be left blank if the type being defined implicitly refers to the base type. This column is used only when the type is a subset of a previously defined type. The **Definition** column can also refer to an ASN.1 type if available in the standard. The first entry in the example declaration defines a type called Transport Classes. There is no base type entered because the definition consists of an enumerated type with the values explicitly defined. Fig. 4.8 shows the table for an ASN.1 type definition for a type used in the FTAM (File Transfer Access and Management) protocol.

USER TYPE DEFINITIONS			
Name	Base Type	Definition	Comments
Transport_Classes		(Class0, Class1, Class2)	Some Classes
Seq_num	INTEGER	(0..128)	

Fig. 4.7. TTCN User Types Example Table.

USER ASN.1 TYPE DEFINITION
Type Name : Access-Request
ASN.1 Definition or Reference
Access-Request ::=[APPLICATION 13] IMPLICIT BIT STRING { read (0), insert(1), replace(2), erase(3), extend(4), read-attribute(5), change-attribute(6), delete-file(7) }

Fig. 4.8. ASN.1 User Types Example Table.

Test Suite Parameters

The implementor of the protocol under test has to fill in two documents which are meant to provide information helpful for conformance testing, such as the features supported, constants used as parameters, timer values etc. These documents are the Protocol Implementation Conformance Statement (PICS) and Protocol Implementors eXtra Information for Testing (PIXIT). This section of TTCN is used to declare the constants derived from PICS/PIXIT. These constants are referred to as the test suite parameters. Fig. 4.9 shows the tabular form for this declaration. Examples of test suite parameters are values for timers, values for credit, etc.

TEST SUITE PARAMETERS			
Name	Type	PICS/PIXIT Ref.	Comments
R_TIME	INTEGER	ISO xxxx	value for a timer
WIN_SIZE	INTEGER	ISO xxxx	Window Size for protocol

Fig. 4.9. Test Suite Parameters Example Table.

Constants

This section is used to declare those constants not derived from PICS/PIXIT, and which are constant throughout the suite. The name, type and value of each constant are given in the proforma shown in Fig. 4.10.

GLOBAL CONSTANTS			
Name	Type	Value	Comments
WAIT_FOR_CC	INTEGER	20	Time Tester will wait to establish connection Implementation dependent

Fig. 4.10. Constants Example Table.

Points of Control and Observation (PCO)

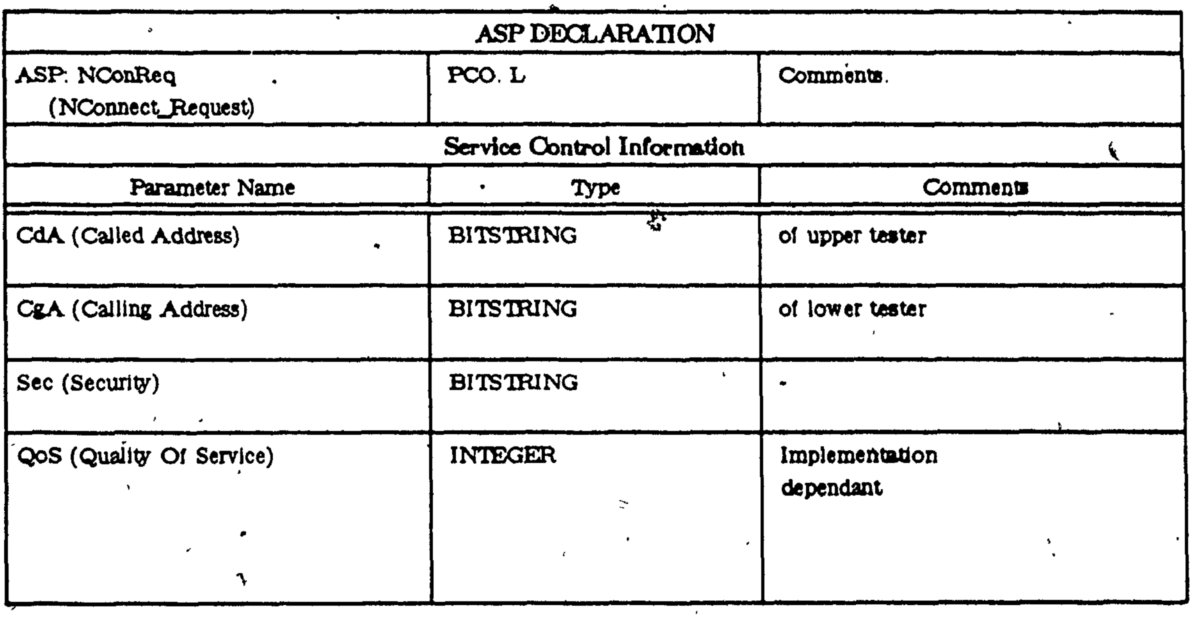
This section describes the set of points at which the IUT displays controllability and observability. An explanation of the location of these PCOs with relation to the testing environment is also to be given in textual form. The number of PCOs depends on the abstract test architecture used, as was seen in Chapter 2. The PCO model is based on two First In First Out queues, one for control (stimulus) and one for observation. This information will be provided in the proforma shown in Fig. 4.11.

PCO DECLARATIONS	
Name	Role
LT	Above the (N-1)-Service Provider at the SAP remote from the (N)-entity under test.
UT	SAP above the (N)-entity under test.

Fig. 4.11. PCOs Example Table.

Abstract Service Primitives

ASPs are abstractly defined in the service standards. The set of ASPs which may occur at the PCOs declared before are declared here, along with comments about their use. Like the User Type definitions, one can declare ASPs using the TTCN definition or the ASN.1 definition. Fig. 4.12 shows the screen representation of the ASP table. The menu has an item **Goto** which allows the user to goto the next declaration or declare a new ASP if required. The Parameter Name column contains the list of parameters of the service primitive with the type of these parameters being declared in the next column. These types can be complex types but must have been defined before in the User Types if not belonging to one of the predefined types. Fig. 4.13 shows the hard copy of the table produced by the **Hard Copy** item on the menu. The ASN.1 declaration of service primitives parallels that of User Types.

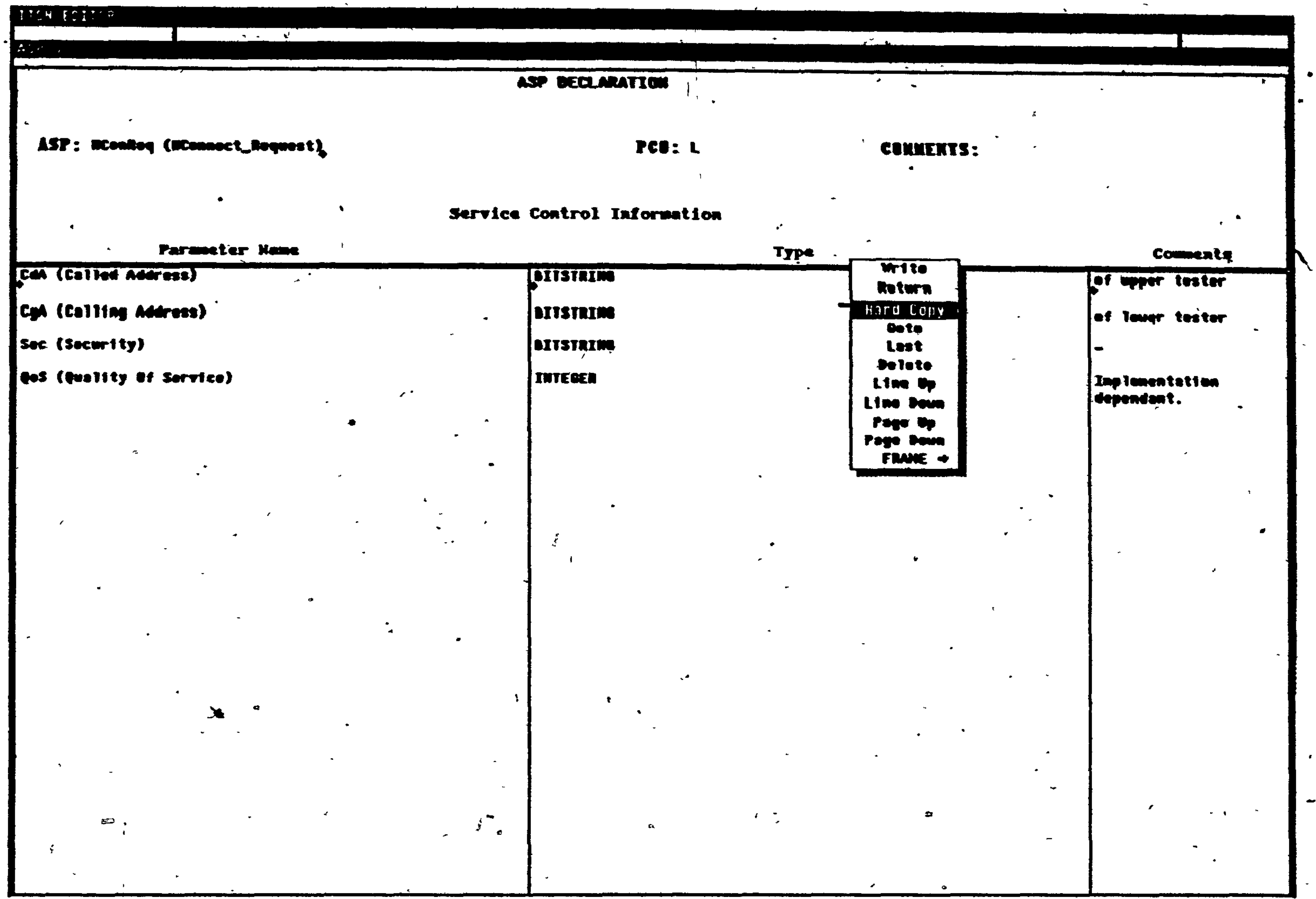


ASP DECLARATION		
ASP: NConReq (NConnect_Request)	PCO. L	Comments.
Service Control Information		
Parameter Name	Type	Comments
CdA (Called Address)	BITSTRING	of upper tester
CgA (Calling Address)	BITSTRING	of lower tester
Sec (Security)	BITSTRING	-
QoS (Quality Of Service)	INTEGER	Implementation dependant

Fig. 4.12. Screen Representation - ASPs.

Protocol Data Units

This section of TTCN is used to declare data types used in a test suite, which are usually Protocol Data Units. User Type definitions are also used to define data types but PDUs are given a separate proforma because of their special significance. The



ASP DECLARATION

ASP: NConReq (NConnect_Request) PCO: L COMMENTS:

Service Control Information

Parameter Name	Type	Comments
CdA (Called Address)	BITSTRING	of upper tester
CgA (Calling Address)	BITSTRING	of lower tester
Sec (Security)	BITSTRING	-
QoS (Quality Of Service)	INTEGER	Implementation dependant.

Fig. 4.13. ASPs Example Table.

PDU DECLARATION		
PDU Name: INIrq (F-INITIALIZErequest)	Comments: Initialise request	
ProtocolControl Information		
Field Name	Type	Comments
protocol_id	Protocol-Version	This type is defined
presentation-context-management	BOOLEAN	
service-level	Service-Level	
service-class	Service-Class	
attribute-groups	Attribute-Groups	
calling-address	Application-Entity-Title	
called-address	Application-Entity-Title	
checkpoint-window	INTEGER	

Fig. 4.14. TTCN PDU definition Example Table.

ASN.1 DATA TYPE DECLARATION
PDU Name : INIrq (F-INITIALIZErequest)
ASN.1 Definition or Reference

```
F-INITIALIZE-request ::=SEQUENCE {

  protocol-id   Protocol-Version
        DEFAULT {  major version-1 , minor  revision-1 },

  presentation-context-management [1] IMPLICIT BOOLEAN
        DEFAULT FALSE,

  service-level Service-Level DEFAULT reliable,

  service-class Service-Class DEFAULT management-class, --should be transfer-class--

  attribute-groups  Attribute-Groups  DEFAULT { },

  calling-address Application-Entity-Title,

  called-address Application-Entity-Title,

  checkpoint-window [8] IMPLICIT INTEGER DEFAULT 1
  }
```

Fig. 4.15. ASN.1 PDU definition Example Table.

TTCN tabular declaration allows the user to define the PDU field by field as shown in Fig. 4.14. The example shows a PDU declaration for the F-INITIALIZE-request PDU of the FTAM protocol. It has a number of fields whose types are assumed to have been declared before using either ASN.1 User Types or TTCN User Types. Fig. 4.15 shows the same PDU declaration in ASN.1.

Global Variables

Variables which are common to both the dynamic part and constraints part are declared in this proforma. These may be used to store values obtained from ASP or PDU test events in the behaviour description or as regular variables as in any programming language. An example table is given in Fig. 4.16. The **Values** column is used to initialize the values of these variables.

GLOBAL VARIABLES			
Name	Type	Value	Comments
result	INTEGER	1	This stores returned values of test steps
pass	INTEGER	1	These may also be declared as constants
fail	INTEGER	2	
inconc	INTEGER	0	

Fig. 4.16. Global Variables Example Table.

Timers

This section is used to define timer types by virtue of their duration. A timer of the defined type may then be used in the behaviour part and there is no restriction on the number of timers which may be made use of. The duration is to be given using seconds, milliseconds, microseconds and seconds as possible time units. Fig. 4.17 gives the proforma and some example timer types.

TIMER DECLARATIONS		
Timer Type Name	Duration	Comments
Retransmission-Timer	R_TIME sec	Obtained from Test Suite Parameters

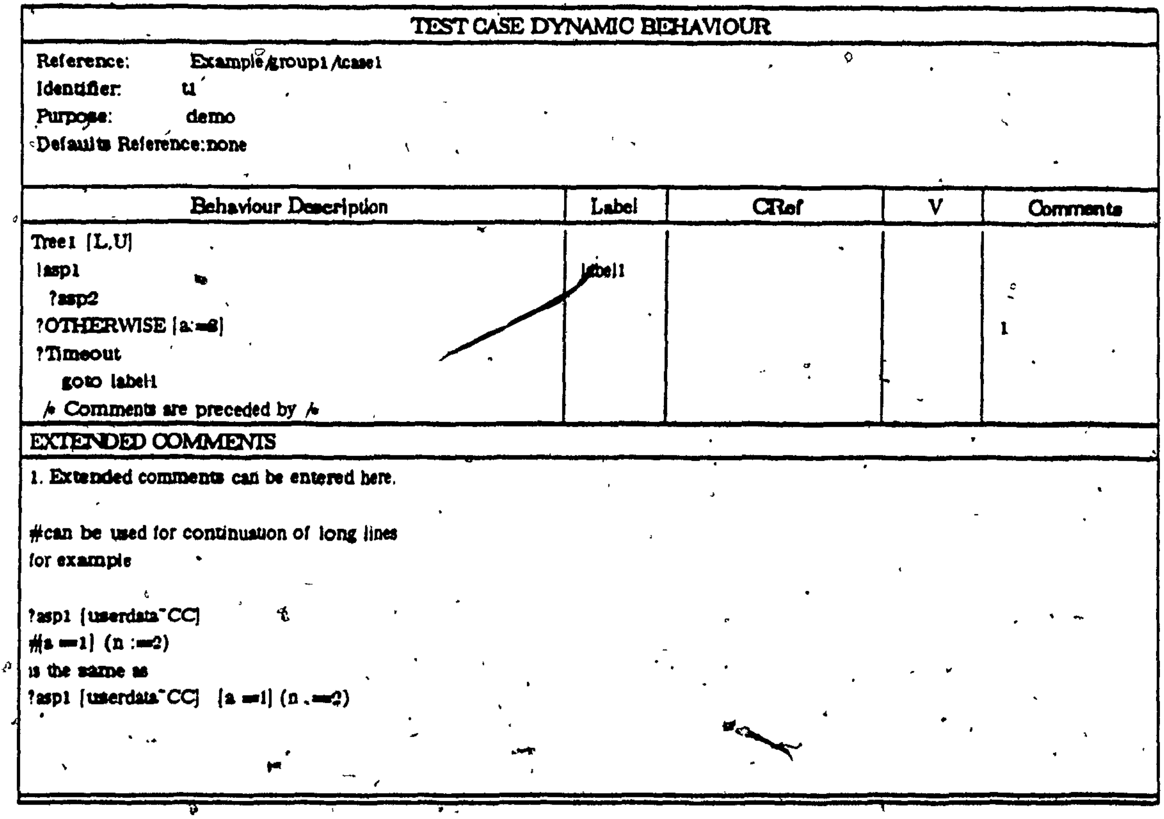
Fig. 4.17. Timers Example Table.

4.4.3 : Dynamic Behaviour

This section defines the sequence of test events which are needed to accomplish a specific test purpose. These test cases may be hierarchically defined as a tree structure or can be flat. The tool allows the user to enter the cases/steps/defaults in any order but keeps in memory the structure of the suite as defined by the appropriate references. The user can interact with this structure, which can be displayed as a tree on the screen, to browse through the suite.

There are three sub-sections in dynamic behaviour :

- Test Cases
- Test Steps
- Default Behaviours



TEST CASE DYNAMIC BEHAVIOUR

Reference: Example/group1/case1
Identifier: t1
Purpose: demo
Defaults Reference:none

Behaviour Description	Label	CRef	V	Comments
Tree1 [L,U]				
!asp1	label1			
?asp2				
?OTHERWISE [a:=8]				1
?Timeout				
goto label1				
/* Comments are preceded by /*				

EXTENDED COMMENTS

1. Extended comments can be entered here.

#can be used for continuation of long lines
for example

?asp1 [userdata^CC]
#[a ==1] (n :=2)
is the same as
?asp1 [userdata^CC] [a ==1] (n :=2)

Fig. 4.18. Test Case Example Table.

Test Steps and Defaults are used to modularize and to factor out recurring possibilities of events, respectively, in a test case. In other words, test cases are used to specify the interesting paths through the test tree, leaving the behaviour which is common, such as Disconnection or Network Reset, which may occur any time, to the default behaviour declaration.

Fig. 4.18 shows the standard proforma for a test case description. Each test case in a test suite is defined in a separate table in TTCN-GR. The **Test Case Reference** field is used to specify the location of the test case in the suite and is of the form *Suite_Identifier / Group Identifier / ..... / Test Case Identifier*. The grouping can be of any depth. The **Test Case Identifier** field is used to give the test case a short unique name because the Reference tends to be long. The **Test Purpose** field gives an informal description of the test purpose and the **Defaults reference** field identifies the default behaviour description for this test case.

While the top fields of the proforma provide general information about the test case, the dynamic description of the case is given in the columns that appear below.

- **Behaviour Description Column**

This column describes the test case/step behaviour in terms of the **Tree Notation** which is explained below. A behaviour description may consist of a number of trees with the first tree entered being the main starting point for the test case. This tree can refer to the library of test steps or may refer to locally defined test steps which are invisible to the other test cases.

4.4.3.1 : The Tree Notation.

The tree notation is characterized by using indentation to represent events in time increasing order. Thus events at the same level of indentation are alternatives and events in increasing indentation represent successive events. For example, suppose the following events can occur during a test whose purpose is to establish a connection, exchange some data and disconnect. It is not enough to just specify the required and expected events but also to specify all other events which may occur and are not under the control of the tester.

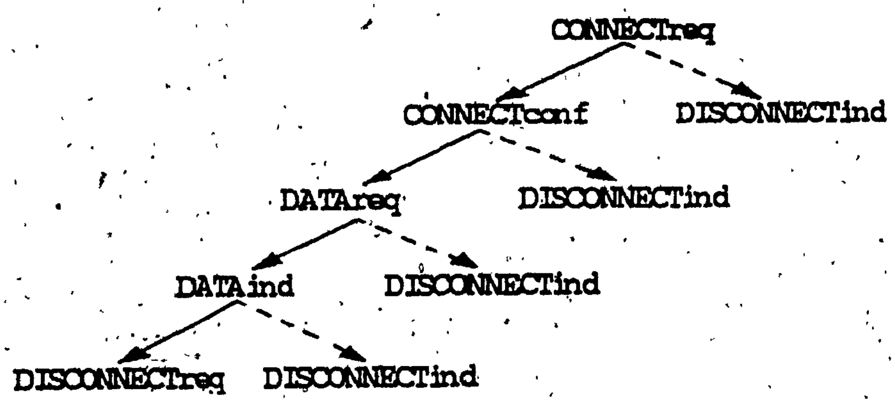
The main sequence:

- *CONNECTrequest, CONNECTconfirm, DATAreq, DATAind, DISCONNECTreq.*

The following are the alternative sequences which may occur because the service provider may be unable to continue with the service required.

- *CONNECTrequest, CONNECTconfirm, DATAreq, DATAind, DISCONNECTind.*
- *CONNECTrequest, CONNECTconfirm, DATAreq, DISCONNECTind.*
- *CONNECTrequest, CONNECTconfirm, DISCONNECTind.*
- *CONNECTrequest, DISCONNECTind.*

This sequence of alternative behaviours can be viewed as all the possible paths of the tree shown below, starting at the root, *CONNECTreq*.



The tree notation expresses this sequence of events as :

```
TREE [L]
 L ! CONNECTrequest
   L ? CONNECTconfirm
     L ! DATAreq
       L ? DATAind
         L ! DISCONNECTreq
         L ? DISCONNECTind
       L ? DISCONNECTind
     L ? DISCONNECTind
   L ? DISCONNECTind
```

Here, *L* stands for the PCO at which the lower tester exercises the test. The symbols *?* and *!* stand for receive and send respectively. So, *L ! CONNECTrequest* means that the tester transmits the CONNECTrequest primitive at the PCO L at that point in the test. **TREE** [L] is the identifier for this behaviour tree and L stands for the formal PCO used. The tree identifier can also have formal parameters to pass values, although these formal parameters and PCOs are useful only in a tree describing a test step. Test Case trees are invisible outside the test case behaviour table, whereas test step trees can be defined to be global. Test Steps defined in the Test Step library are global whereas Test Steps defined within a Test Case as a sub-tree are not.

4.4.3.2 : Behaviour Description Syntax and Semantics.

Each line in the Behaviour Description column is referred to as an *Event Line*. This can be any one of the following :

- An event ::=Send | Receive | Otherwise | Timeout | Elapse
- A statement ::=Goto | Attach | Repeat
- Pseudo Event ::=Boolean Expression | Arithmetic Expression | Timer Operation.

Here " | " stands for alternate choices.

Event :

This type of test event includes the basic test procedure - the sending or receiving of an ASP or PDU. These events always occur at a particular PCO which is specified as in the example. If the test suite is defined using only one PCO, then the PCO need not be specified at all (for example, in test suites employing the remote test method). If the event is a send event, then one can use the **Encoded As** statement to enforce the encoding of the SDU parameter of the ASP being sent.

For example :

L ! NDATAReq (UserData ^ CC_TPDU)

This example specifies that the tester at the PCO L is to transmit the ASP NDATAReq whose UserData parameter is to be encoded (^) as the CC_TPDU (Connection Confirm Transport PDU) which is assumed to have been defined before. Constraints can further be placed on this encoding, as will be discussed later.

There can be only one *encoded as* expression by itself for a PDU or an ASP on an event line. There could be nested *encoded as* expressions though, specifying encoding requirements on the fields of PDUs which are target for the first encoding statement. For example :

L ! NDATAReq (UserData ^ CC_TPDU (UDATA ^ DISC_TMPDU))

Here, the *UserData* parameter of the ASP *NDATAReq* is to be encoded as *CC_TPDU* whose *UDATA* field, in turn, is to be encoded as the Disconnection Test Management PDU used for Coordinated Test Architectures.

L ? NDATAInd [UserData ~ CC_TPDU]

This event shows that the tester at PCO L will receive the ASP NDATAInd whose UserData decodes (~) to the CC_TPDU.

A notation is provided called *Implicit Send* in order to specify action to be taken by undefined (non-standard) Upper Testers. An example is

<IUT ! DR >

This represents initiation from the IUT side. This is useful to specify test cases in the *Remote* test architecture where no assumption is made about the Upper Tester. The event specified is the event to be transmitted by the IUT and not the Upper Tester. It is assumed that the Upper Tester can stimulate the IUT in such a way as to cause this action.

The **Otherwise** statement is used to specify the reception of unforeseen or don't care events. Otherwise is used to specify that the upper tester or lower tester will accept any event which has not been given previously as an alternative. Otherwise need not be the last alternative.

For example :

PCO1 ? A *pass*

PCO1 ? B *pass*

PCO1 ? Otherwise *fail*

This states that the tester at PCO PCO1 will accept A or B but for all other events, the verdict is fail.

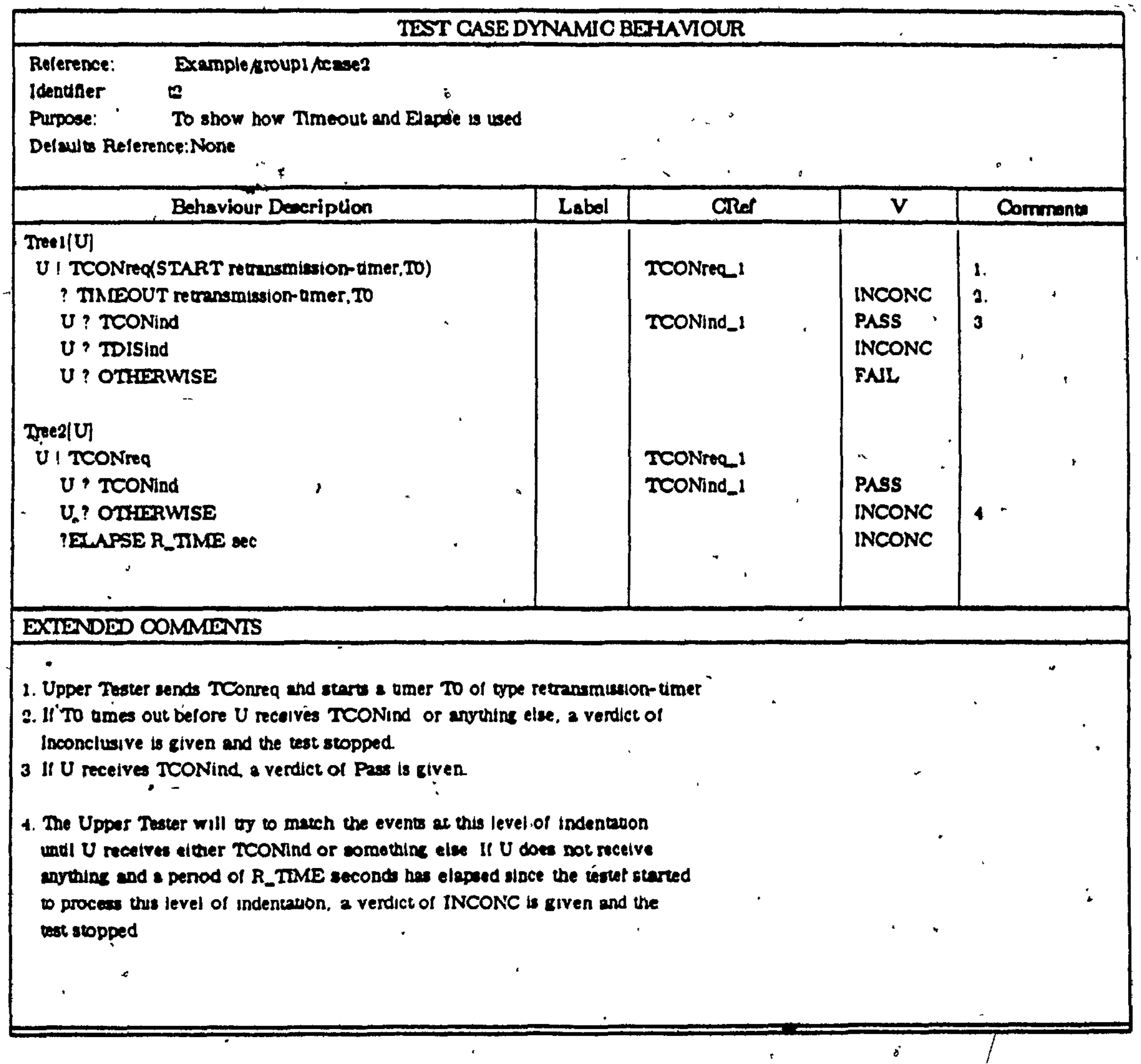
The two statements **Timeout** and **Elapse** are essentially timing events. The timeout event specifies the action to be taken on the expiry of a given or default timer. The Elapse event is used to specify the length of time a tester will be in a particular set of alternatives before taking some other action. Fig. 4.19 shows examples using Timeout and Elapse events.

Statements :

There are three types of statements which are events by themselves. These are :

- **GOTO :**

This statement is used to unconditionally jump to another part of a test tree which is labeled using the second column of the test case proforma. The referenced event has to be the first of the set of alternatives, if any. *Tree_1* in Fig. 4.20 shows the use of GOTO. The Boolean guard tests the value of variable *loop_var*. If the value is less than 128, the GOTO statement is enabled and execution continues from the statement labeled *loop*. If the Boolean expression returns FALSE, the alternative event(s) are next in line for possible execution.



TEST CASE DYNAMIC BEHAVIOUR

Reference: Example/group1/tcase2
Identifier t2
Purpose: To show how Timeout and Elapse is used
Defaults Reference:None

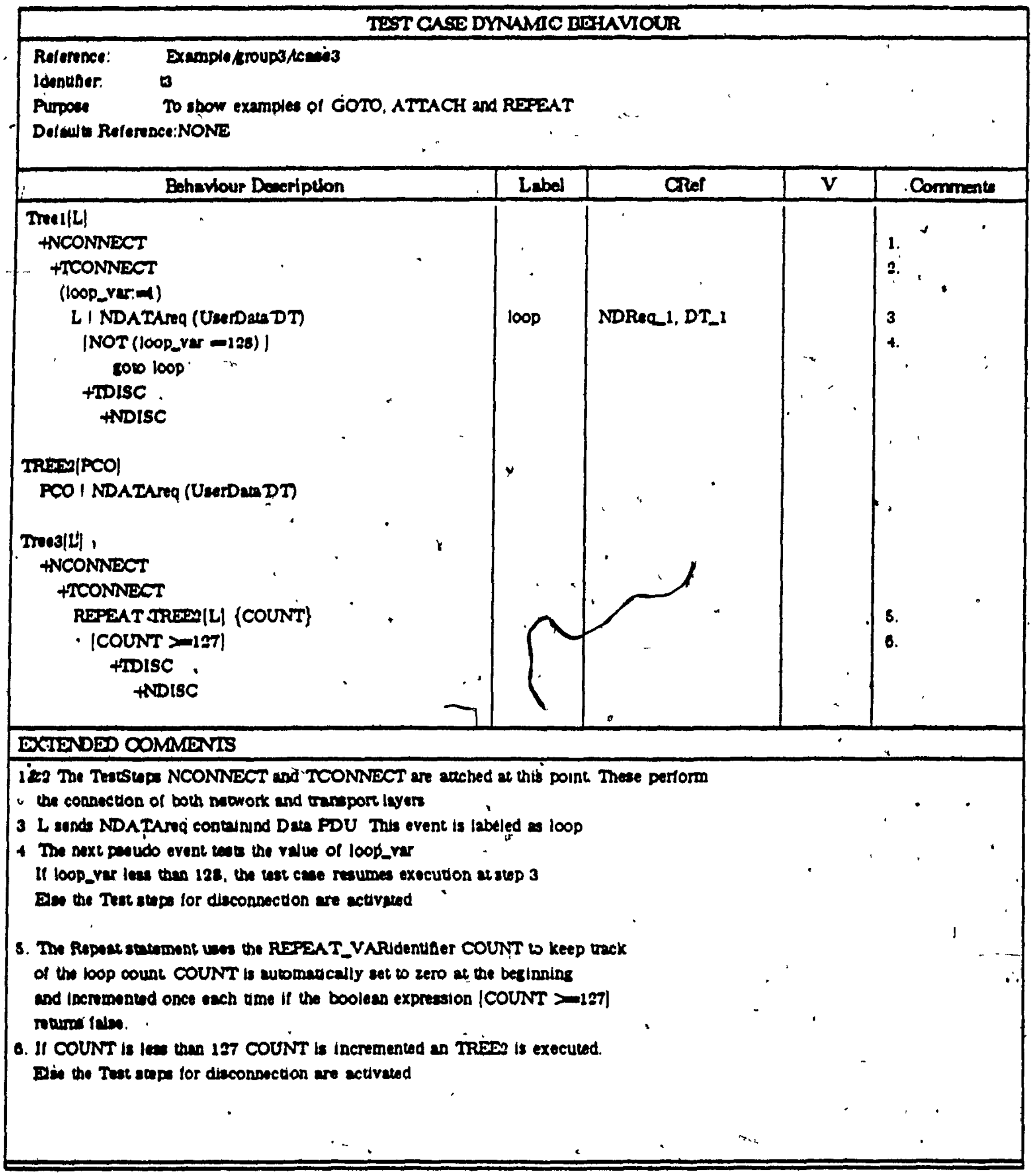
Behaviour Description	Label	CRef	V	Comments
Tree1[U]				
U ! TCONreq(START retransmission-timer,T0)		TCONreq_1		1.
? TIMEOUT retransmission-timer,T0			INCONC	2.
U ? TCONind		TCONind_1	PASS	3
U ? TDISind			INCONC	
U ? OTHERWISE			FAIL	
Tree2[U]				
U ! TCONreq		TCONreq_1		
U ? TCONind		TCONind_1	PASS	
U ? OTHERWISE			INCONC	4
?ELAPSE R_TIME sec			INCONC	

EXTENDED COMMENTS

1. Upper Tester sends TConreq and starts a timer T0 of type retransmission-timer
2. If T0 times out before U receives TCONind or anything else, a verdict of Inconclusive is given and the test stopped.
3 If U receives TCONind, a verdict of Pass is given.

4. The Upper Tester will try to match the events at this level of indentation until U receives either TCONind or something else If U does not receive anything and a period of R_TIME seconds has elapsed since the testet started to process this level of indentation, a verdict of INCONC is given and the test stopped

Fig. 4.19. Timeout and Elapse Examples.



TEST CASE DYNAMIC BEHAVIOUR

Reference: Example/group3/case3
Identifier: t3
Purpose To show examples of GOTO, ATTACH and REPEAT
Defaults Reference:NONE

Behaviour Description	Label	CRef	V	Comments
Tree1[L]				
+NCONNECT				1.
+TCONNECT				2.
(loop_var:=1)				
L ! NDATAreq (UserData DT)	loop	NDReq_1, DT_1		3
[NOT (loop_var =128)]				4.
goto loop				
+TDISC				
+NDISC				
TREE2[PCO]				
PCO ! NDATAreq (UserData DT)				
Tree3[L]				
+NCONNECT				
+TCONNECT				
REPEAT TREE2[L] {COUNT}				5.
[COUNT >=127]				6.
+TDISC				
+NDISC				

EXTENDED COMMENTS

1&2 The TestSteps NCONNECT and TCONNECT are attched at this point. These perform
the connection of both network and transport layers
3 L sends NDATAreq containind Data PDU This event is labeled as loop
4 The next pseudo event tests the value of loop_var
If loop_var less than 128, the test case resumes execution at step 3
Else the Test steps for disconnection are activated

5. The Repeat statement uses the REPEAT_VARidentifier COUNT to keep track
of the loop count. COUNT is automatically set to zero at the beginning
and incremented once each time if the boolean expression [COUNT >=127]
returns false.
6. If COUNT is less than 127 COUNT is incremented an TREE2 is executed.
Else the Test steps for disconnection are activated

Fig. 4.20. GOTO, ATTACH and REPEAT Examples.

• **ATTACH :**

This statement is used to modularize test trees by acting as procedure calls. The Attach statement (which is symbolized by "+") is used to specify that a particular test step is to be attached at that point of the tree. The test step referenced may be either a locally defined one or a library test step.

Tree_2 in Fig. 4.20 shows the use of ATTACH. In the example shown, ATTACH is used to modularize the preamble to most transport tests, viz., the establishment of a network connection, followed by transport connection, etc.

• **REPEAT :**

The Repeat statement is used to specify iteration over a test step a required number of times, the minimum being once. Boolean guards may be used to come out of the loop. There is a predefined *REPEAT_VARIdentifier* which may be used as a counter through the looping. This global variable is read only and is set to zero at every REPEAT statement.

REPEAT statements are executed at least once since the Boolean guards are tested only after one iteration through the loop. *Tree_3* in Fig. 4. 20 shows the use of the REPEAT construct. *TREE_2* will be traversed once before returning control back to *Tree_3*. If variable COUNT happens to be 128, the REPEAT statement stops iteration and control passes to the next level of indentation. Note that REPEAT will always succeed at least once and so it is redundant to place any alternatives to a REPEAT statement.

Pseudo Events :

• **Boolean & Arithmetic Expressions :**

A pseudo event consists of boolean and arithmetic expressions and timer operations. These may be specified on a line by themselves or may follow regular events. Boolean expressions are used as guards for events. Arithmetic expressions are performed only if the event which precedes the expression can occur and any boolean expression specified is satisfied. For example

L ! NDATAreq [X >1] (a :=2)

The meaning of this line is that NDATAreq is sent and a is set to 2 only if X is greater than 1.

- Timer Operations :

Timer operations consist of Start, Cancel and Read statements. START operation indicates that the specified timer should be started, or if already running, be restarted. CANCEL timer is used to inactivate a timer and READTIMER operation stores the current value of a timer into a global variable.

These operations would be used to test for the timing requirements on an IUT in the following way. In the example, RET_Timer is the Timer Type whose value is equal to the time the LT should wait for an acknowledgement before timing out and either retransmitting or quitting.

```
        Behaviour                                              Label
L ! NDataReq (UserData ^ CR) (START RET_Timer, t1)              Start
   L ? NDataInd [UserData ^ CC]
      ( CANCEL START RET_Timer, t1 )
   ?TIMEOUT START RET_Timer, t1
      GOTO Start
```

where START operation is followed by the Timer Type RET_Timer and t1 is a specific instance (variable) of that type. On the reception of CC before the expiry of t1, the timer is cancelled. If t1 expires, CR is transmitted and t1 is restarted.

- **Labels Column**

The column **Label** is used, appropriately enough, to label event lines as targets.

for the GOTO statement. GOTO statements must target only the first of a set of alternatives.

- **Constraints Reference Column**

The next column, **Constraint Reference (CRef)** is used to specify constraints on the ASPs and PDUs in the event lines. There are two ways of specifying constraints : the full form and the short form. A full form constraint identifies explicitly the ASP or PDU on which the constraint is placed. As an example:

Behaviour Description	*Constraints Reference*
L ? NDATAind [UserData ~CR_TPDU]	*NDATAind[DT1], CR_TPDU[CR1]*

DT1 and CR1 are specific instances of NDATAInd and CR_TPDU, respectively with specified values. This line states that the tester at L will accept an NDATAInd which has parameters identical to the ones given in DT1 and whose UserData parameter decodes to CR_TPDU, whose fields are identical to instance CR1.

The short form implies the ASP or the PDU the constraint is placed on.

L ? NDATAind [UserData ~CR_TPDU] *DT1, CR1*

The requirement is that the constraint references have to be in the same order the targets appear in the event line.

Constraints may be parametrized to allow the passing of values from the behaviour description. The next section describes how constraints are defined.

- **Verdicts Column**

The **Verdicts** column is used to record the result of having traversed a particular path upto the leaf. A verdict may be either preliminary or final. A global variable **$R** is available for each test case, initialized to 'None'. Possible preliminary results are Pass, Fail and Inconclusive. The preliminary verdict is given for the test case for some aspect of the test purpose being achieved. It is placed on those events which do not represent the final leaf of the testing tree. The final verdict given to a case must be consistent with the value of $R. For example if "$R" is fail, then a final verdict of pass is considered inconsistent.

4.4.3.3 : CONTEST-TTCN Behaviour Tables

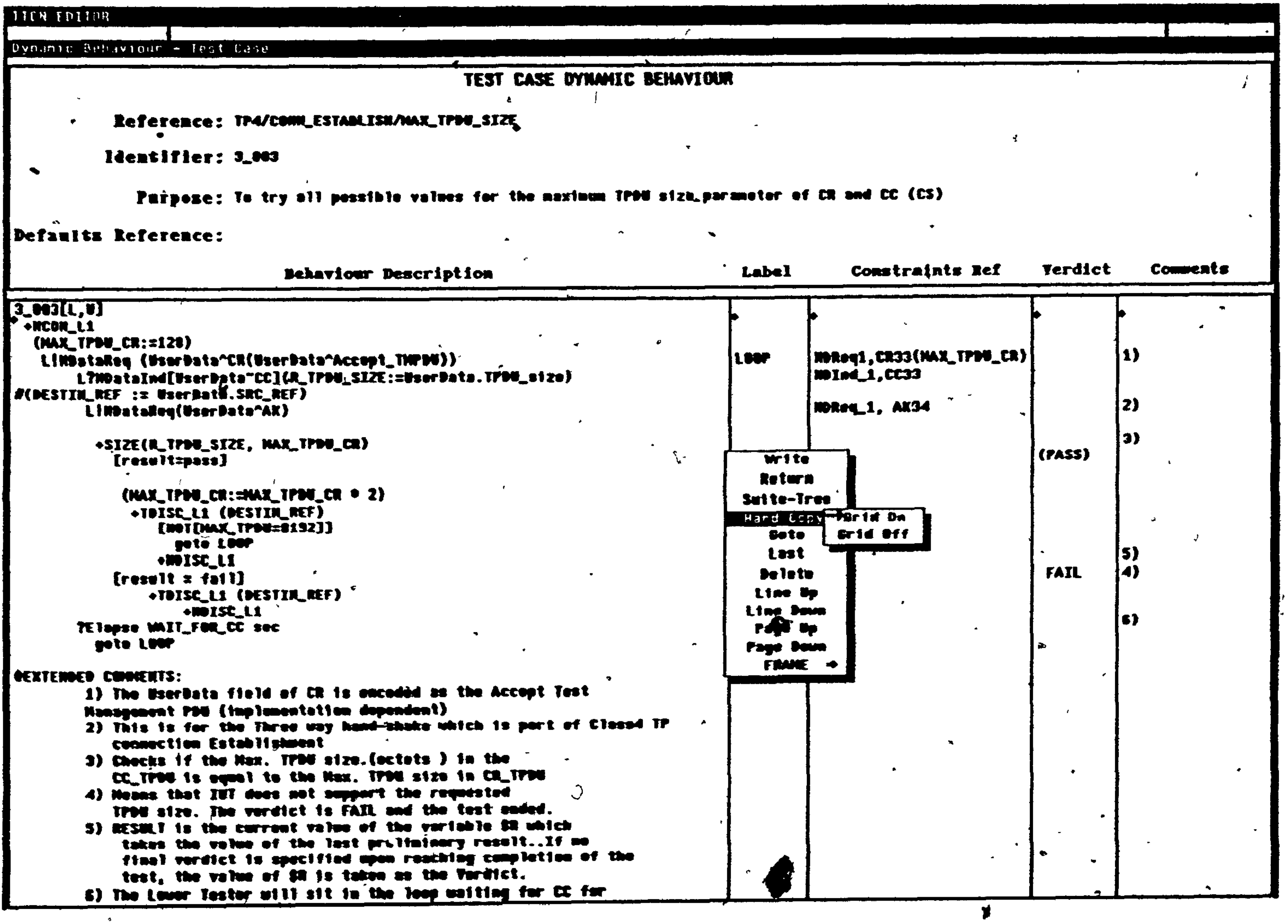
Fig. 4.21 shows the way the screen appears when the user is entering the test case. The menu shown comes up when the user depresses the mouse button on the frame. The menu items of interest here are

Suite-Tree : This item brings up a window which shows the structure of the suite as a tree with the suite identifier as the root with children representing groups and test cases. The user can select the test case of interest with the mouse and browse through the suite. Fig. 4.22 shows the structure of a typical suite. The test case which has been selected can be seen to be the one on the screen.

Goto : This item can be used to jump to a specified test case by giving the number or to create a new test case.

Fig. 4.23 shows the menu which comes up when the user depresses the mouse button in the behaviour description column. The user can select the ASP, PDU or abbreviation required to "stuff" it at the required point. This eliminates spelling errors and proves handy when defining large test suites containing many ASPs, etc. **V-Hair** draws a vertical bar following the mouse in the first column . This is useful to align events which are at a distance from each other.

Test Step and Default declarations are similar to test case declarations in terms of proforma and syntax. A test step does not have a *Purpose,* it has an *Objective.* The user describes the function of the particular test step in this field.. The order in which the user enters the test cases, test steps or defaults is irrelevant because the editor takes care of ordering on the basis of the reference structure.



TEST CASE DYNAMIC BEHAVIOUR

Reference: TP4/CONN_ESTABLISH/MAX_TPDU_SIZE

Identifier: 3_003

Purpose: To try all possible values for the maximum TPDU size parameter of CR and CC (CS)

Defaults Reference:

```
Behaviour Description                                     Label   Constraints Ref             Verdict  Comments
3_003[L,U]
 +NCON_L1
  (MAX_TPDU_CR:=128)
   L!NDataReq (UserData^CR(UserData^Accept_TMPDU))        LOOP    NDReq1,CR33(MAX_TPDU_CR)             1)
        L?NDataInd[UserData^CC](R_TPDU_SIZE:=UserData.TPDU_size)  NDInd_1,CC33
#(DESTIN_REF := UserData.SRC_REF)
        L!NDataReq(UserData^AK)                                   NDReq_1, AK34                        2)

          +SIZE(R_TPDU_SIZE, MAX_TPDU_CR)                                                               3)
            [result=pass]                                                                     (PASS)

            (MAX_TPDU_CR:=MAX_TPDU_CR * 2)
             +TDISC_L1 (DESTIN_REF)
               [NOT[MAX_TPDU=8192]]
                 goto LOOP
               +NDISC_L1                                                                               5)
            [result = fail]                                                                   FAIL     4)
                +TDISC_L1 (DESTIN_REF)
                  +NDISC_L1                                                                            6)
      ?Elapse WAIT_FOR_CC sec
        goto LOOP
```

@EXTENDED COMMENTS:

1) The UserData field of CR is encoded as the Accept Test Management PDU (implementation dependent)

2) This is for the Three way hand-shake which is part of Closed TP connection Establishment

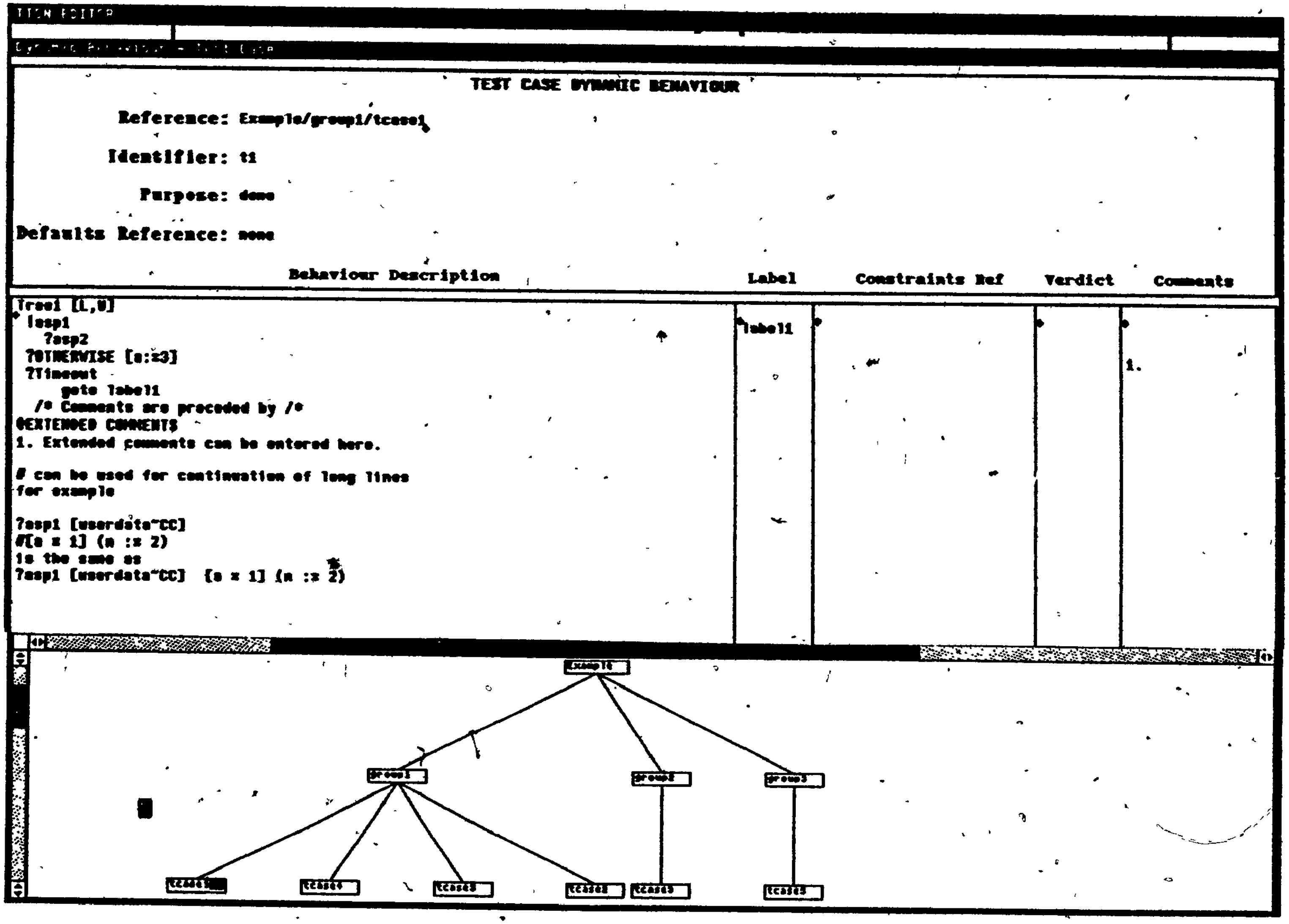
3) Checks if the Max. TPDU size.(octets) in the CC_TPDU is equal to the Max. TPDU size in CR_TPDU

4) Means that IUT does not support the requested TPDU size. The verdict is FAIL and the test ended.

5) RESULT is the current value of the variable $R which takes the value of the last preliminary result..If no final verdict is specified upon reaching completion of the test, the value of $R is taken as the Verdict.

6) The Lower Tester will sit in the loop waiting for CC for

Fig. 4.21. Screen Representation with Frame Menu.



TEST CASE DYNAMIC BEHAVIOUR

Reference: Example/group1/tcase1

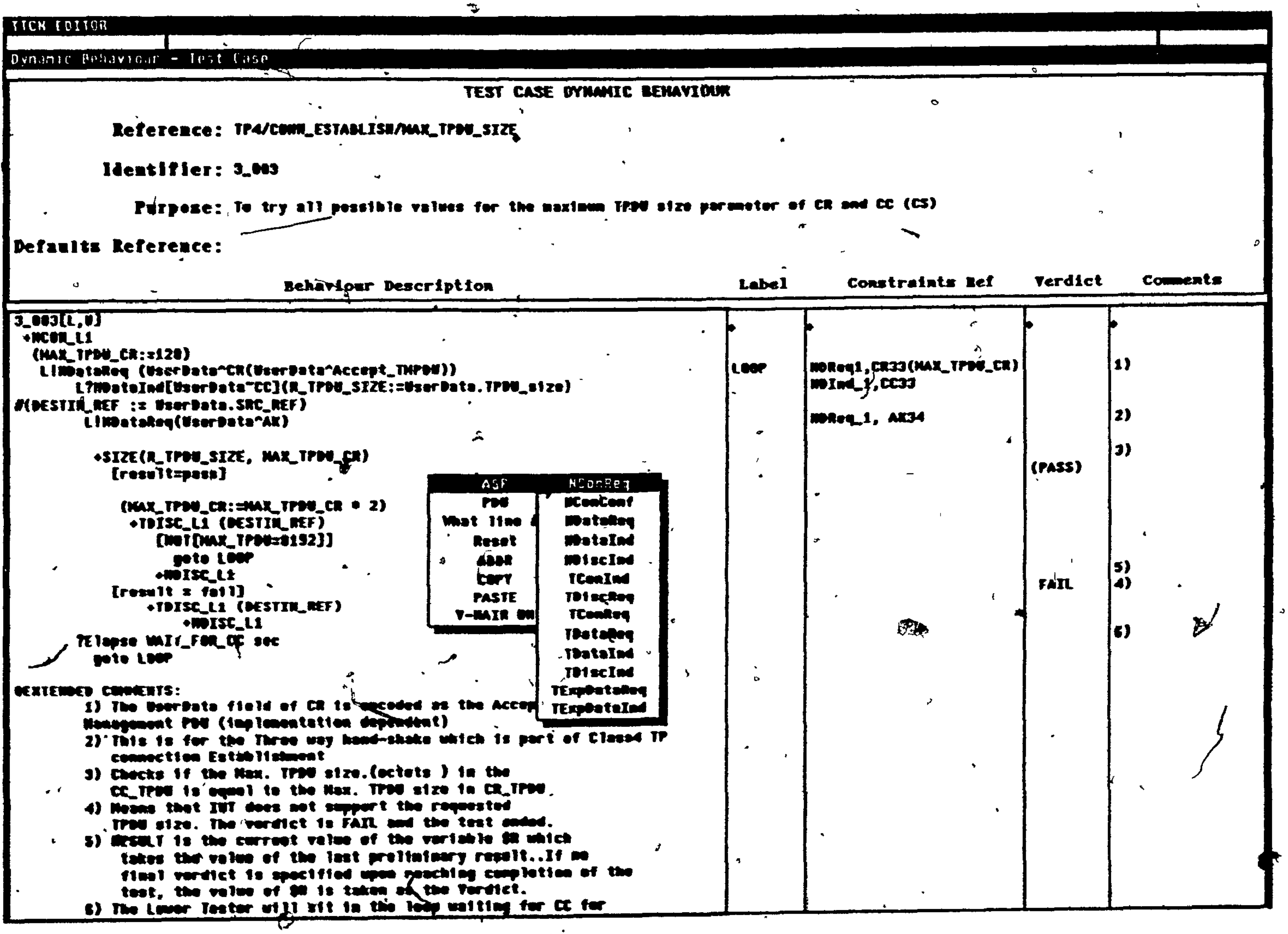
Identifier: t1

Purpose: done

Defaults Reference: none

```
                Behaviour Description                   Label    Constraints Ref   Verdict   Comments
Tree1 [L,U]
 !asp1                                                  label1
   ?asp2
 ?OTHERWISE [s:=3]                                                                           1.
 ?Timeout
     goto label1
  /* Comments are preceded by /*
@EXTENDED COMMENTS
1. Extended comments can be entered here.

# can be used for continuation of long lines
for example

?asp1 [userdata~CC]
#[s = 1] (n := 2)
is the same as
?asp1 [userdata~CC]  [s = 1] (n := 2)
```

Fig. 4.22. Screen Representation with Suite Tree.



TEST CASE DYNAMIC BEHAVIOUR

Reference: TP4/CONN_ESTABLISH/MAX_TPDU_SIZE

Identifier: 3_003

Purpose: To try all possible values for the maximum TPDU size parameter of CR and CC (CS)

Defaults Reference:

```
Behaviour Description                                          Label  Constraints Ref          Verdict  Comments

3_003[L,U]
 +NCON_L1
  (MAX_TPDU_CR:=128)
   L!NDataReq (UserData^CR(UserData^Accept_THPDU))            LOOP   NDReq1,CR33(MAX_TPDU_CR)           1)
       L?NDataInd[UserData^CC](R_TPDU_SIZE:=UserData.TPDU_size)       NDInd_1,CC33
#(DESTIN_REF := UserData.SRC_REF)
        L!NDataReq(UserData^AK)                                       NDReq_1, AK34                      2)

         +SIZE(R_TPDU_SIZE, MAX_TPDU_CR)                                                                 3)
           [result=pass]                                                                     (PASS)

            (MAX_TPDU_CR::=MAX_TPDU_CR * 2)
             +TDISC_L1 (DESTIN_REF)
               [NOT[MAX_TPDU=8192]]
                 goto LOOP
               +NDISC_L1                                                                                 5)
           [result = fail]                                                                   FAIL        4)
              +TDISC_L1 (DESTIN_REF)
                 +NDISC_L1                                                                               6)
       TElapse WAIT_FOR_CC sec
        goto LOOP

@EXTENDED COMMENTS:
        1) The UserData field of CR is encoded as the Accep
        Management PDU (implementation dependent)
        2) This is for the Three way hand-shake which is part of Class4 TP
           connection Establishment
        3) Checks if the Max. TPDU size.(octets ) in the
           CC_TPDU is equal to the Max. TPDU size in CR_TPDU.
        4) Means that IUT does not support the requested
           TPDU size. The verdict is FAIL and the test ended.
        5) RESULT is the current value of the variable $R which
            takes the value of the last preliminary result..If no
            final verdict is specified upon reaching completion of the
            test, the value of $R is taken as the Verdict.
        6) The Lower Tester will sit in the loop waiting for CC for
```

Fig. 4.23. Screen Representation with Behaviour Column menu.

The next and last section of TTCN is the Constraints declaration part.

4.4.4 : Constraints Declaration

This section is probably the most voluminous part of any TTCN test suite because constraints have be defined for every occurrence of ASPs and PDUs in the dynamic behaviour. Methods have been provided to parametrize the constraints, define sub constraints, define generic constraints and generic fields, all aimed at compactness. Constraints are similar for both ASPs and PDUs so only PDU constraint declarations will be explained.

- Basic TTCN Constraints

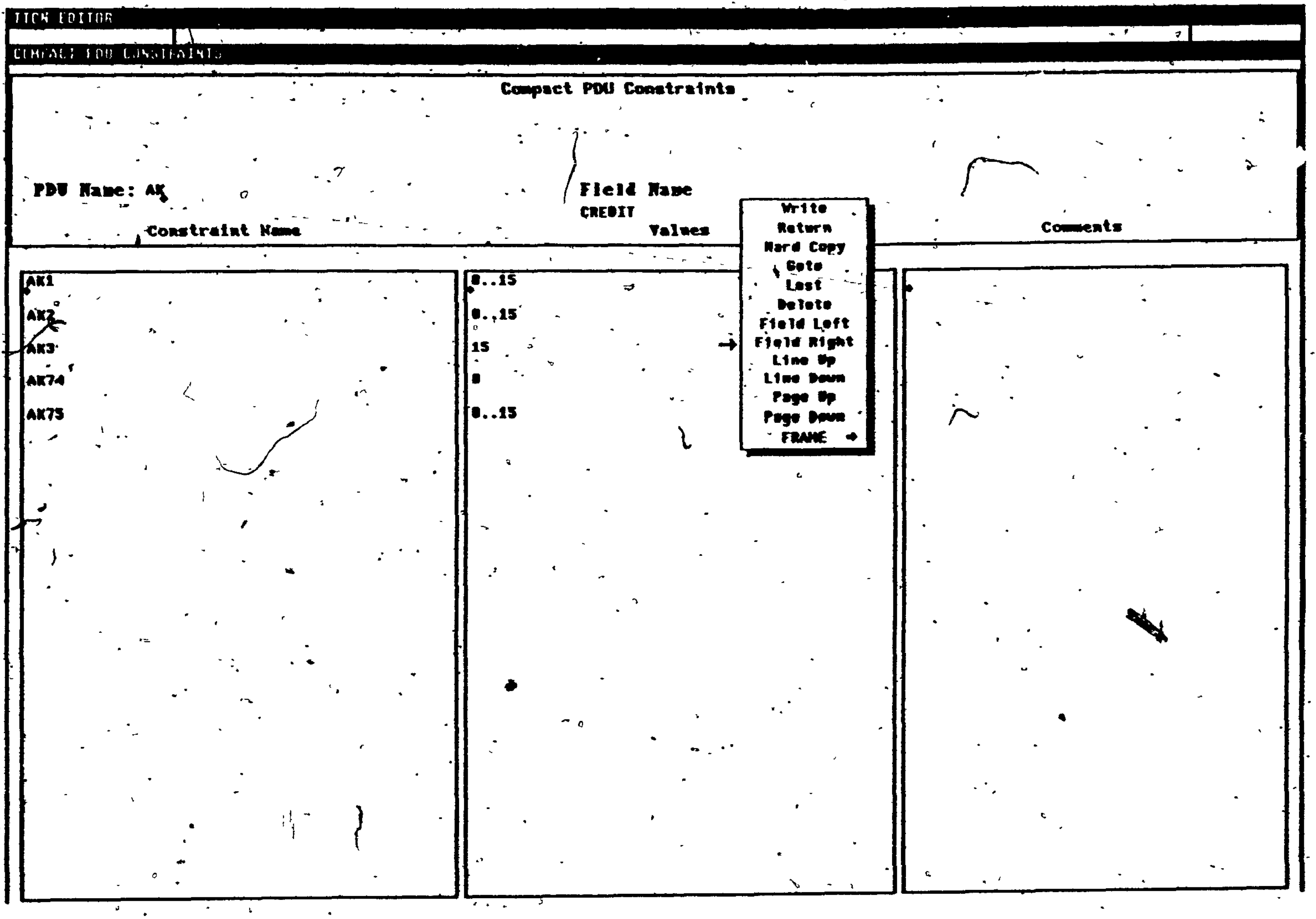
The basic method of defining a constraint is to take a PDU defined in the Declarations part and specify a set of values for all its fields. Each set is given a name and this name, along with the PDU identifier, is referred to in the Dynamic behaviour. Fig. 4.24 shows the proforma for such a constraint definition. Each constraint on a PDU requires a new proforma.

PDU CONSTRAINT		
PDU Name: CR	Constraint Name. CR_1	
Field Name	Value	Comments
Called_Address	IUT	Predefined OctetString
Calling_Address	LOWER_T	The Base Constraint
		must define values
		for all fields

Fig. 4.24. TTCN PDU Constraint Example Table.

- Compact Constraints

The next method is called the compact constraint declaration. In this proforma, all the constraints on a particular PDU are defined using only one proforma. This is suitable for PDUs with a small number of fields. The editor allows upto thirty fields to be



Compact PDU Constraints

PDU Name: AK

Field Name
CREDIT

Constraint Name	Values	Comments
AK1	0..15	
AK2	0..15	
AK3	15	
AK74	0	
AK75	0..15	

Fig. 4.25. Screen Representation - Compact Constraints.

defined in this manner. Fig. 4.25 shows the screen representation of a compact PDU constraint. AK1 is the name of the first constraint on the PDU AK. Credit is the first field on which these constraints are placed. A value of "?" implies that any single legal character will be accepted and a value "*" implies that a string of legal characters will be accepted. The menu shows two items called *Field Left* and *Field Right*. These items are used to define new fields or browse through the fields. Only one field is shown at a time on the screen. Selecting the *Hard-Copy* item produces the table in Fig. 4.26. Five fields are given at a time in one table.

PDU CONSTRAINT LIST

PDU Name: AK

Constraint Name	Field Name					Comments
	CREDIT	DST_REF	YR_TU_NR	Variable_Part	LWE	
AK1	15	DST1	NIL	NIL	NIL	
AK2	15	DST1	NIL	NIL	LWE1	
AK3	15	DST1	0	CHKSUM1	NIL	
AK74	0	DST1	NIL	NIL	NIL	
AK75	15	DST1	NIL	SUB_SEQ_NO	LWE1	

Fig. 4.26. Compact PDU Constraint Example Table.

- Generic Value and Field Constraints

The next method is to define generic constraints which may be parametrized. A number of fields in a PDU may be collapsed into one generic name and a number of values defined for this generic name for those fields. For example, in Fig. 4.27, fields SRC_REF and DST_REF both refer to IUT_REF for constraint CR2. CR2 is parametrized by IUT_REF. IUT_REF is the identifier for a generic value list. Fig. 4.28 shows the generic constraint for IUT_REF with two constraints C1 and C2 defined for fields SRC_REF and DST_REF, respectively. Now, one can refer to CR_PDU in the dynamic behaviour by

CR_PDU[CR_2(C1)]

referring to generic constraint C1 placed on fields SRC_REF and DST_REF for the constraint CR_2.

PDU CONSTRAINT LIST						
PDU Name CR						
Constraint Name	Field Name					Comments
	SRC_REF	DST_REF	DATA			
CR_1	1	0	DATA_FIELD			
CR_2(IUT_REF)	IUT_REF	IUT_REF	DATA_FIELD			

Fig. 4.27. PDU Constraint Example Table.

GENERIC VALUE LIST						
PDU Name CR_PDU				Generic Name IUT_REF		
Constraint Name	Field Name					Comments
	SRC_REF	DST_REF				
C1	1	4				
C2	12	0				

Fig. 4.28. Generic Value Constraint Example Table.

Fig. 4.29 shows a table showing a constraint for CR_PDU with fields REFERENCE and DATA. REFERENCE is not an actual field in the PDU but refers to a generic field constraint declaration shown in Fig. 4.30. So when one refers to constraint CR_1, the field values are 1 for SRC_REF and 0 for DST_REF.

PDU CONSTRAINT LIST						
PDU Name: CR						
Constraint Name	Field Name					Comments
	REFERENCE	DATA				
CR_1	REF_1	D1				
CR_3	REF_2	D2				

Fig. 4.29. PDU Constraint Example Table.

GENERIC FIELD LIST						
Generic Field Name: REFERENCE						
Constraint Name	Field Name					Comments
	SRC_REF	DST_REF				
REF_1	1	0				
REF_2	45	12				

Fig. 4.30. Generic Field Example Table.

d. ASN.1 Constraints

ASPs and PDUs which have been declared using the ASN.1 notation in the Declarations section may have constraints placed on them using the ASN.1 method. Fig. 4.31 shows a typical constraint table. The constraint is defined on the FTAM (File Transfer Access and Management) PDU *FINITIALIZEresponse.*

ASN.1 VALUE CONSTRAINT DECLARATION	
PDU Name : F_INITIALIZEresponse	Constraint Name: FINIT_1
ASN.1 Value	

```
FINIT_1 ::=[1] IMPLICIT SEQUENCE {
            stateResult  StateResult
            actionResult  ActionResult
            :
            :
            checkpointWindow  [8] IMPLICIT INTEGER 1
            }
```

Fig. 4.31. ASN.1 Constraint Example Table.

There is a mechanism called REPLACE by which one may define only some values required for a specific constraint. For example, on constraint FINIT_1, one may define FINIT_2 (Fig. 4.32) using REPLACE for only those fields whose values are different from the base constraint. One may also parametrize ASN.1 constraints using the **PUT** and **GET** keywords. For example :

L ? NDATAind[UserData~PDU_1]　　　　*PDU_1[CON1(PUT VAR1)]*

L ? NDATAind[UserData~PDU_1]   *PDU_1[CON1(GET VAR1)]*

The first event specifies that the event will match only if PDU_1 received is the same as PDU_1 encoded using the constraint CON_1 with the formal parameter replaced by VAR1. VAR1 is defined as a global variable of the same type as the parameter in CON1? The second event will match if the UserData is decoded to be PDU_1 and the incoming parameter value assigned to VAR1 for later use. PUT is the default operation if none is specified.

<table>
<tr><th colspan="2">ASN.1 VALUE CONSTRAINT DECLARATION</th></tr>
<tr><td>PDU Name : F_INITIALIZEresponse</td><td>Constraint Name: FINIT_2</td></tr>
<tr><th colspan="2">ASN.1 Value</th></tr>
<tr><td colspan="2">FINIT_2 ::=FINIT_1 {
REPLACE checkpointWindow BY [8] IMPLICIT INTEGER 8
}</td></tr>
</table>

Fig. 4.32. ASN.1 Constraint with REPLACE Example Table.

4.5 : TTCN-MP

TTCN-MP is the machine processable form of TTCN. It is a linear representation of the information contained in TTCN-GR. Keywords are used as delimiters instead of columns and tables. Keywords start with the "$" sign. Another difference is that indentation of the events in the Behaviour Description part is denoted explicitly in TTCN-MP. For example,

```
L ! ASP1
   L ? ASP2
```

This would be represented in TTCN-MP as

$BehaviourLine $Line [0] L ! ASP1 $End_BehaviourLine

$BehaviourLine $Line [4] L ? ASP2 $End_BehaviourLine Here, the number in the square brackets is the level of indentation, which is equal to the number of blank spaces before the event.

The following is the TTCN-MP equivalent of the ASP declaration given in Fig. 4.13.

```
$Begin_TTCN_ASPdcl
$ASPid NConReq (NConnect_Request)
$PCOs L

$SCI

$ASP_PARdcl
$ASP_PARid CdA (Called Address)
$ASP_PARtype BITSTRING
$End_ASP_PARdcl

$ASP_PARdcl
$ASP_PARid CgA (Calling Address)
$ASP_PARtype BITSTRING
$End_ASP_PARdcl

$ASP_PARdcl
$ASP_PARid Sec (Security)
$ASP_PARtype BITSTRING
$End_ASP_PARdcl

$ASP_PARdcl
$ASP_PARid QoS (Quality Of Service)
$ASP_PARtype BITSTRING
$End_ASP_PARdcl

$End_SCI

$End_TTCN_ASPdcl
```

Here, SCI stands for Service Control Information, i.e., the list of parameters and their types of the ASP. Each parameter declaration is enclosed between the keywords *$ASP_PARdcl* and *$End_ASP_PARdcl*. The Service Control Information is enclosed between the keywords *$SCI* and *$End_SCI*. The entire ASP declaration is enclosed between *$Begin_TTCN_ASPdcl* and *$End_TTCN_ASPdcl*. This is for TTCN ASP declarations. ASN.1 ASP declarations are given after and the entire ASP declaration section is enclosed between *$ASPdcls* and *$End_ASPdcls*. A similar pattern is followed for all declarations.

4.6 : TTCN Semantics

TTCN semantics are defined using an algorithmic approach. The execution of a behaviour tree is explained by a set of rules given as a series of simple steps. The algorithm takes each set of events, at the same level of indentation and gives the semantics for that event to match. For example, if an event is qualified by a boolean expression, an assignment and a constraints reference, the event will match and the assignment executed only if both the boolean expression and the constraint hold. The execution of a test case terminates if an explicit final verdict is returned from the algorithm.

The alternatives A_1 .... A_m contained in a level are processed in the order of their appearance. Since the semantics applied are assumed to be snapshot semantics, (the status of any of the events cannot change during the event matching process), the processing of a set of alternatives is assumed to be instantaneous.

CHAPTER 5 MORE ABOUT THE EDITOR

INTRODUCTION

This Chapter describes the construction of the editor in detail along with the data structures used and the tools that are the building blocks of the program.

5.1 General Structure

The general structure of the tool is once again given in Fig. 5.1 for easy reference. The normal flow of operation is from top to bottom. The test designer invokes the editor to enter the test suite and after completion, obtains the MP form of the suite after necessary syntactic and semantic checks using the module GR2MP. By semantic errors, it is meant the use of ASPs, PDUs or Test Steps which have not been defined in the test suite. Tabular printouts of the test suite may be obtained during the editing process of individual tables.

If the MP form of the test suite is already available, the module named MP2GR is invoked which creates the necessary files in the form required by the editor to display and modify, if required. The modules above the dashed line have already been implemented and will be explained in detail in the following sections.

The modules below the line are projected to be completed to enable the automatic generation of an executable notation, either ESTELLE or any other language of choice. These modules and the method of achieving these aims will be discussed at the end of the Chapter.

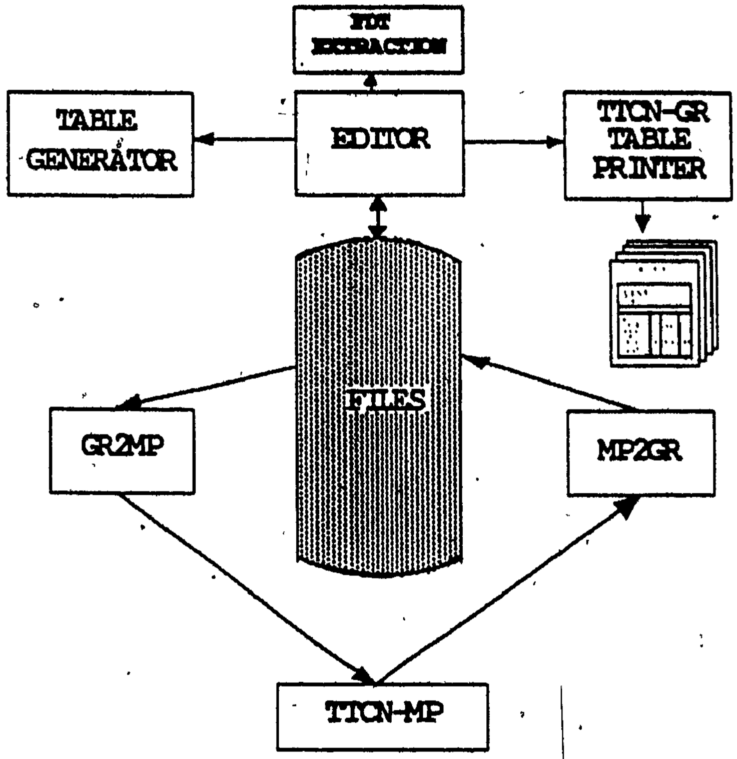


Fig. 5.1. Tool Structure.

5.2 The Editor Module:

The language of choice for implementing the editor was the C language. Once the decision had been made to use the windowing environment on the SUN system, the calls to the window creation and control routines are all in C and as such, C was naturally decided upon. The SUNVIEW windowing environment lends itself naturally to the tabular proformas of TTCN by providing different *subwindows* suitable to different needs. It would have been possible to have used normal ASCII terminals without using special packages to implement the editor, but the development time would have been much longer.

The editor is the heart of the tool. The main routines which call the table generators for each proforma type are contained in this module. The tool makes use of various building blocks which are part of the SUNVIEW environment, the final effect being to produce an interactive editor with sophisticated facilities which is simple and easy to use. All the information pertaining to a test suite is stored in files which use the identifier of the suite as a prefix, making the managing of large suites an easier task.

5.2.1 Tool Building Blocks

The editor module makes use of the following structures which are part of the SUNVIEW environment:

- **FRAME**
- **CANVAS**
- **TEXT SUBWINDOW**
- **PANEL**

Fig. 5.2 shows a generic Frame containing some subwindows. The Frame is a structure upon which various other types of subwindows may be created. In the SUNVIEW notation, there is a root Frame with subframes or subwindows hanging from it. This root Frame "owns" the subwindows. The subwindows may be one the types listed above, each oriented toward a specific function. For example, the Canvas is a surface with built in routines for drawing operations, etc.

The events which are input either from the keyboard or the mouse or any other pointing device are registered with an internal process called the *Notifier* which takes care of passing the event to the Window which currently has the input focus. In the tool, the Window in which the mouse image lies is the current Window. The location of the mouse changes the input context. The menu which comes up when a mouse button is pressed depends on the location of the mouse. For example, if the mouse button is pressed on the Frame anywhere, a menu pops up which contains functions pertain-

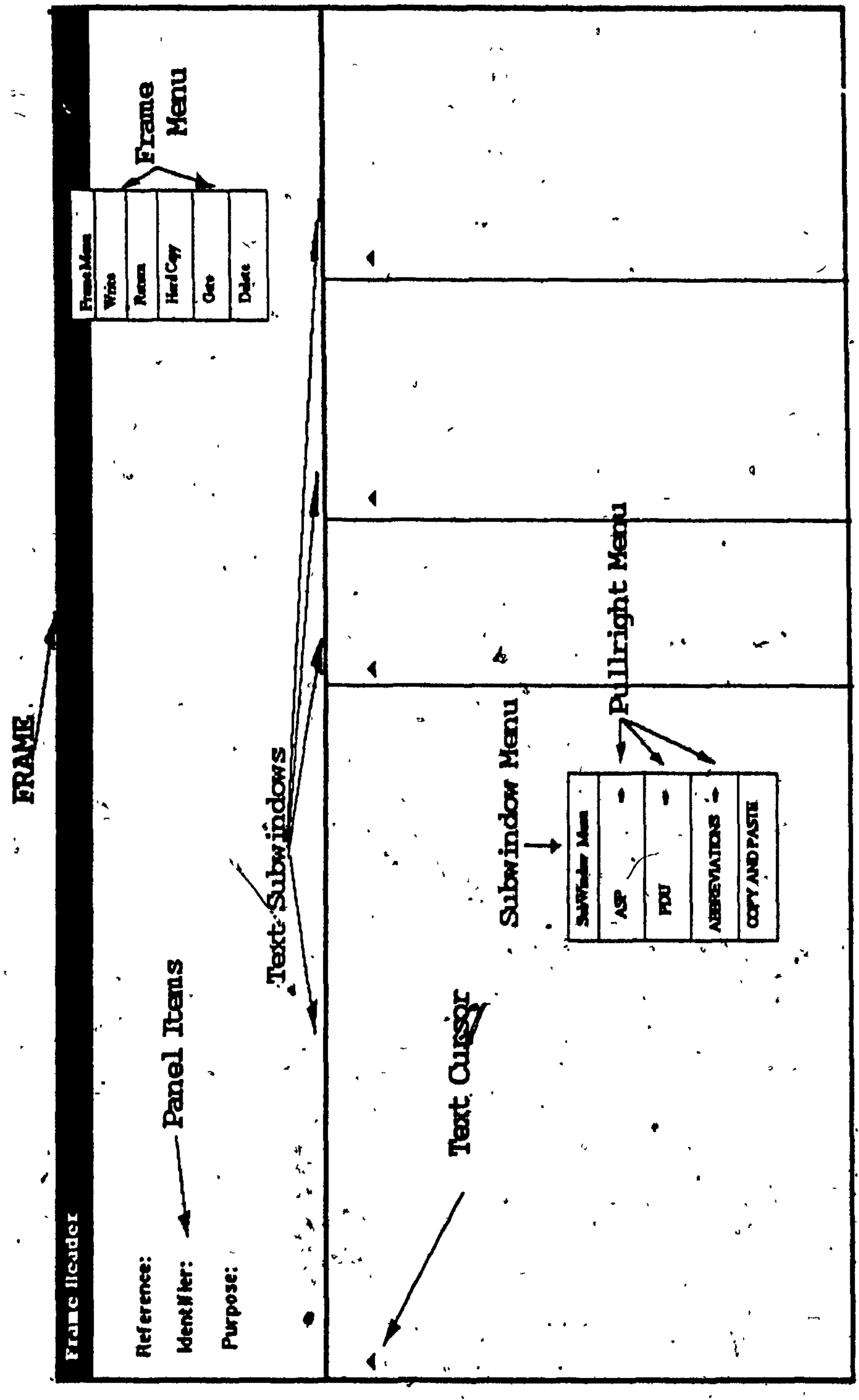


Fig. 5.2. Generic Frame in SUNVIEW.

ing to the entire table such as saving, deleting, obtaining a printout, etc. The menu which pops up in a column would contain items such as finding out the line number of the current entry, or resetting the column etc.

Panels are a type of Window which are generally used for limited interaction with the user for functions such as confirming some actions or providing buttons for item selection in the place of pop up menus. In the tool, all the parts of a table with a fixed number of items have been created using Panels. Pressing <RETURN>moves the cursor from one item to the next in a cyclical order while the mouse is in the Panel.

The Text Subwindow is a mouse and keyboard driven text editor which is scrollable. All the columns of a table are made up of individual Text subwindows. The user is prompted to enter corresponding entries in the columns one after another by tagging the <RETURN> key so that the next column becomes the input focus in a cyclical order. This allows quick entry of fields without having to move the mouse for each field. This action may be circumvented by using the *Line Feed* key. The current point of insertion can be set by pointing with the mouse or using the cursor keys on the right of the keyboard.

The Canvas is used in the tables for Test Cases. The structure of the suite is displayed as a tree with the test case nodes being mouse sensitive. The Canvas is a window oriented towards graphic applications. Drawing operations may be done on a Canvas by obtaining the handle to the *Pixwin* of the Canvas. The Pixwin is an abstraction of the portion of the drawing surface of the screen which is owned by the Canvas. Various functions exist to draw lines and curved shapes along with writing text at specified locations with different fonts.

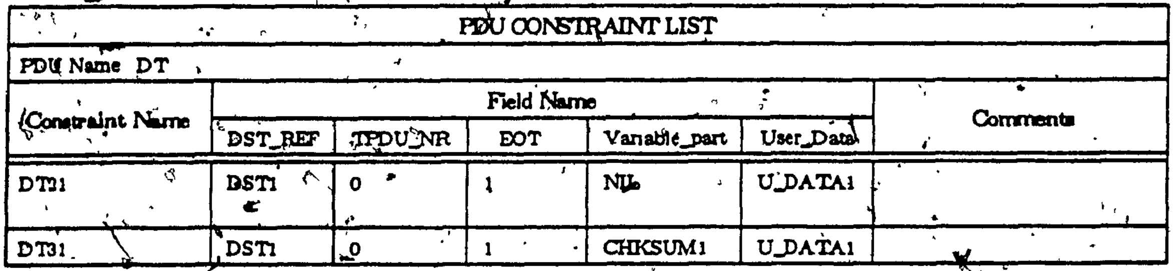
The tool also makes use of the text formatting package called *Troff* which is standard on UNIX machines. Troff involves writing text interspersed with formatting and line drawing commands. Macros are provided for some standard, commonly used routines such as creating tables, equations, etc. The macro *Tbl* is used to create the tabular proformas of TTCN.

On selecting the *Hard-Copy* item from the menu, a file named *__printout* is created automatically. This file contains the information along with the formating commands and is sent to the laser printer using the printer daemon relevant for the system on which the tool is running. The following is a sample Troff file with the tbl macro calls.

```
.ds RF continued on next page....
.ds LH ....continued from previous page
.nr LL 6.5i
.nr PS 9
.TS H
tab (') box;
~~cb~~ s s s s s s.
PDU CONSTRAINT LIST
_
.R
.T&
l s s s s s s.
PDU Name: DT
_
.T&
cb | cb s s s s | cb
cb | c | c | c | c | c | cb.
Constraint Name'Field Name'Comments
' ' ' ' ' '
'DST_REF'TPDU_NR'EOT 'Variable_part'User_Data'
_
.TH
.R
.T&
lw(1i) | lw(0.5i) | lw(0.5i) | lw(0.5i) | lw(0.5i) | lw(0.5i) | lw(1.5i).
_
DT21    'DST1 '0   '1   'NIL  'U_DATA1'
_
DT31    'DST1 '0   '1   'CHKSUM1'U_DATA1'
_
.TE
.sp
.ds RF
.ds LH
```

Fig. 5.3 shows the output of the Troff file above. In the input file, the top two commands define the page header and footer to be used in case the table spans across more than one page. At the end of the file, these two strings are set to null so that if the table fits on a page, the header and footer will be disabled and nothing appears at the foot of the page. This is one of the requirements specified in the TTCN standard for multi page tables.

The commands between .TS and .TE commands are Tbl macros along with the regular Troff commands. These commands signify the starting and ending of a table. The .T& command symbolizes a change in the definition of the column headers and fonts, within a single table definition. In the example, after having defined the table header, which uses only a single column, the .T& macro is invoked. The next line defines the column headers for the body of the table.



PDU CONSTRAINT LIST						
PDU Name DT						
Constraint Name	Field Name					Comments
	DST_REF	TPDU_NR	EOT	Variable_part	User_Data	
DT31	DST1	0	1	NIL	U_DATA1	
DT31	DST1	0	1	CHKSUM1	U_DATA1	

Fig. 5.3. Sample TROFF output.

5.3.2 Program File Structure

The editor is organized as a number of C files which are linked together at compile time. These files each represent the functions required for the different proformas and are completely independent. This means that the user can have a number of tables on the screen at one time, editing in one and printing out another. The main routine is in the file called *ttcn.c* which contains the calls to the different functions in the different files. All the global variables used in the files are declared in this file, such as the icon structures and some character array declarations. This file also contains the routines which takes care of system error handling for all the sections of TTCN.

The general structure of a program in the SUNVIEW environment is given below.

- *Declarations of variables and procedures.*
- *Main routine or procedure*
- *Creation of the Main Frame and its subframes*
- *Registering interesting input events with the Notifier and linking the procedures to be called with the input events.*

The Main frame and subframes are sized and placed so as to replicate the table corresponding to the section. All the buffers and other variables local to the files are declared as *Static* so as not to interfere with another table which may be running in parallel.

5.2.3 The Internal File Structure

The editor maintains a set of files for each table that can be created. These files all have the common name of the suite identifier (provided by the user upon calling the program) with suffixes identifying which proforma they belong to. For most tables, there is a file per column and a file for the Panel containing general information about the entries in that table. For declarations which need to have many tables, like ASPs or PDUs, these suffixes have further numerical suffixes. Since there may be a large number of test cases, test steps and defaults, these sections just create one file per test case/step with delimiters marking the end of different fields.

For example, take a test suite with the name Basic_test. This name is entered by the user at the beginning of every session with the editor. It may but need not be the same as the formal Suite Identifier which is declared in the Suite Overview Section. The files which contain the information for the Suite Overview section are named

Basic_test_a.1 for the top field containing general information.

Basic_test_a.c.[1-4] for the four columns of the test case entries.

Basic_test_a.s.[1-4] for test step entries, if any.

Basic_test_a.d.[1-4] for default steps, if any.

An ASP table would create the following files. The * represents the number of the ASP in the order in which it was entered in relation to the other ASPs in the suite.

Basic_test_h..1*

Basic_test_h..2*

Basic_test_h..3*

Basic_test_h..4*

A Test Case would create a file with the name

*Basic_test_o.**

The * represents any number depending when the case was entered. For large test suites, the standard UNIX packages *Tar* and *Compress* may be used to keep compact internal files for the editor. Tar packages the whole directory as one file and Compress reduces the size by upto 1/3.

These internal files are all placed in a directory called *suite_files* in the home directory of the user.

5.3 The GR2MP module

This module has been implemented entirely using standard C. The module goes through the files with the prefix of the suite identifier which is given as the only argument to the program. The conversion is done in the same order as the sections in TTCN-MP. During the conversions, linked lists are maintained of pertinent information for later sections. For example, ASP and PDU identifiers are stored to be checked against those used in the Dynamic Behaviour section.

For the Dynamic Behaviour section, the test case/step/default references are first read in to create a tree structure containing the test groups and subgroups. This tree is searched depth first to create the MP form in the correct order as required.

5.4 The MP2GR Module

This module breaks up the incoming MP file into the internal files in the format described above. This module has been written using the UNIX tools *YACC* and *LEX*. These two tools enables the user to define the rules of the input language using a notation similar to the BNF (Backus Naur Form) and to use the lexical analyzer to return tokens recognized from the input file to be parsed according to the grammar rules. BNF rules for TTCN-MP are given in the standard [4].

This module performs strict type checking and other syntax checks on an input MP specification. If the specification passes all the checks, the program creates the internal files containing input for the editor. The following is an example MP specification of a test suite whose suite identifier is Example with three Test Cases.

```
$Suite
$Suiteid Example

$Begin_SuiteOverview
$SuiteId  Example
$StandardsRef  None
$PICSref  None
$PIXITref  None
$HowUsed  Demonstration of tool
$TestMethods  Quite informal!
$SuiteIndex

$TestCaseIndex

$TestCaseId  t1
$TestCaseRef  Example/group1/tcase1
$Page  1
$Description
First Test Case

$TestCaseId  t2
$TestCaseRef  Example/group2/tcase2
$Page  2
$Description
Second Test Case example.

$End_TestCaseIndex
$TestStepIndex
```

```
$End_TestStepIndex
$DefaultIndex
$End_DefaultIndex
$End_SuiteIndex

$Declarations$End_Declarations
$DynamicPart
.
.
.
$End_DynamicPart
$ConstraintsPart
$End_Constraints
$End_Suite
```

MP2GR would create the following intermediate files for the editor to read containing the information as shown :

- demo_a.1 (* File containing Suite Overview top field information *)

```
Example
None
None
None
Demonstration of tool
Quite informal!
```

- demo_a.c.2 (* File containing first column of Test Case declarations *)

```
t1

t2
```

- demo_a.c.3 (* File containing second column of Test Case declarations *)

```
Example/group1/tcase1

Example/group2/tcase2
```

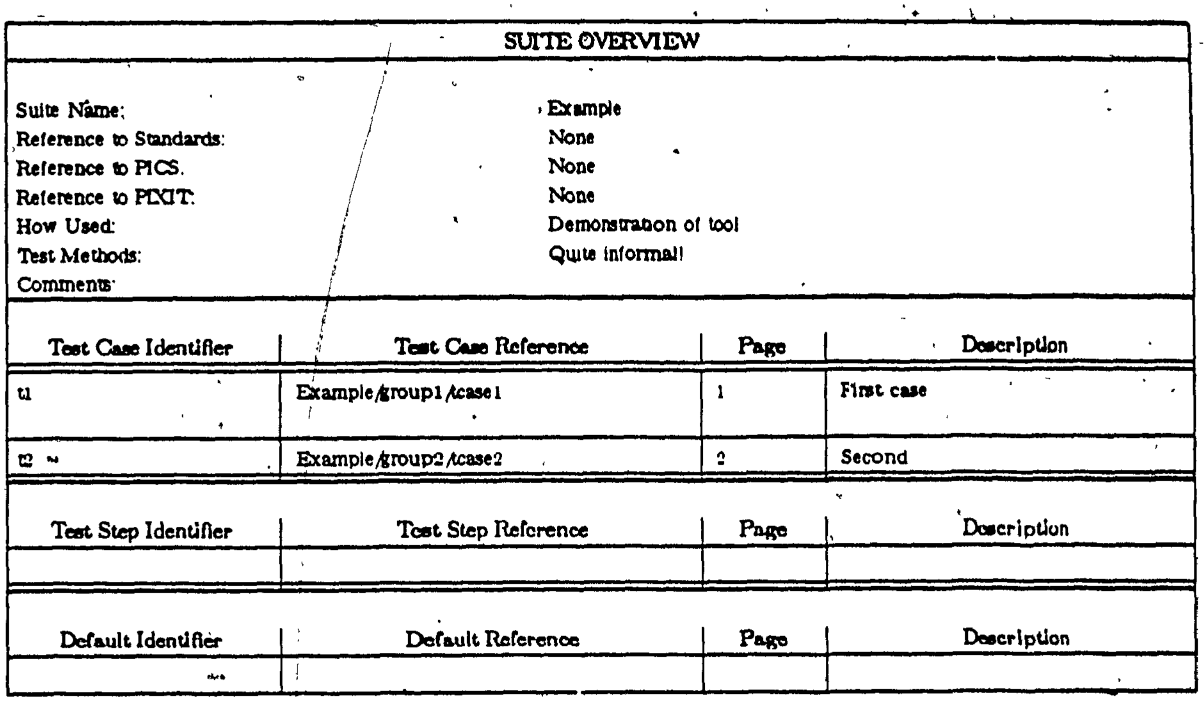
- demo_a.c.4 (* File containing third column of Test Case declarations *)

```
1

2
```

- demo_a.c.5 (* File containing fourth column of Test Case declarations *)

```
First Case

Second Test Case example.
```

Fig. 5.4 shows the table created by the tool once the user invokes the tool with the suite identifier given as *demo*.



SUITE OVERVIEW

Suite Name:	Example
Reference to Standards:	None
Reference to PICS.	None
Reference to PIXIT:	None
How Used:	Demonstration of tool
Test Methods:	Quite informal!
Comments:	

Test Case Identifier	Test Case Reference	Page	Description
t1	Example/group1/case1	1	First case
t2	Example/group2/case2	2	Second

Test Step Identifier	Test Step Reference	Page	Description

Default Identifier	Default Reference	Page	Description

Fig. 5.4. Suite Overview Example Table.

5.5 FDT Extractor For ESTELLE

This module is useful in instances where a formal description of the protocol is available. This module makes use of the NBS (National Bureau of Standards) compiler for the ESTELLE formal description language. A modified version of the file containing the input routines for the lexical analyzer is used to extract the necessary information. This file, called *lexio.c* is something of a preprocessor for the lex source. This file contains routines for reading the input specification line by line to return tokens to the lexical analyser and some other routines to expand *#include* directives, etc.

The routine which reads a line of input is modified so that every occurrence of the keyword **Channel** is recognized. The information following till the end of the channel definition is read and placed in the proper internal files as has been defined in.

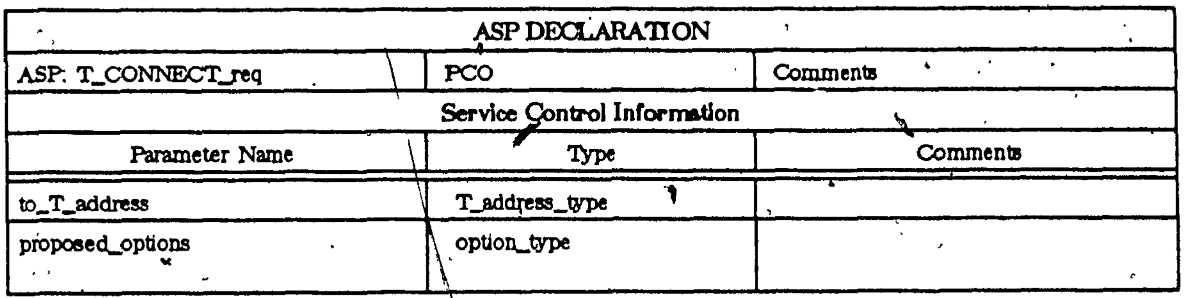
the earlier section.

For example, given an ESTELLE specification of the transport layer :

```
specification Transport;
  reason_type =( TS_user_initiated, wrong_options );

/* Other type definitions*/

/* CHANNEL Definition */
channel TCEP_primitives ( user, provider );
      by user:
         T_CONNECT_req( to_T_address : T_address_type;
                        proposed_options : option_type );
         T_CONNECT_resp( proposed_options : option_type );
         T_DISCONNECT_req;
         T_DATA_req( TS_user_data : data_type );
         user_ready ( length: integer);
      by provider:
         T_CONNECT_ind( to_T_address : T_address_type;
                        proposed_options : option_type );
         T_CONNECT_conf( proposed_options : option_type );
         T_DISCONNECT_ind( disconnect_reason : reason_type );
         T_DATA_ind( TS_user_data : data_type;
                  is_last_fragment_of_TSDU : boolean );

/* rest of the specification */
```

The information in the Channel definition can be extracted to create the tables for the ASP declarations for a transport test suite. In ESTELLE, Channels represent FIFO queues between modules of an entity. These Channels may be used to model SAPs if they communicate with the external environment. So the program looks for the keyword *channel* and gets the ASPs and their parameters and their types. These types are kept in another file for the user to lookup. For example, *T_CONNECT_req* is a primitive which is defined in the service standard of the transport layer. This primitive (according to this simplified specification) has two parameters: *to_T_address* and *proposed_options*.

Of course, the user may have to enter some types used in the specification as User Types in the TTCN suite if they do not belong to the predefined types in TTCN. In this example, both *T_address_type* and *option_type* have to be defined in the User Types. Fig. 5.5 shows the table which contains the information pertaining to the ASP T_CONNECT_req.



ASP DECLARATION		
ASP: T_CONNECT_req	PCO	Comments
Service Control Information		
Parameter Name	Type	Comments
to_T_address	T_address_type	
proposed_options	option_type	

Fig. 5.5: ASP Example Table.

CHAPTER 6 EXAMPLE TEST CASES

INTRODUCTION

This chapter gives some example test cases for two standard protocols, namely, FTAM and TP4. The test cases are explained along with the test purpose using the TTCN-GR notation and the resulting TTCN-MP specification is given for the FTAM test case. The Remote, Coordinated and Distributed Test Architectures, as described in Chapter 2, are used in specifying these examples.

6.1 File Transfer Access and Management (FTAM)

The first protocol of interest is the FTAM [18] protocol. FTAM is an application layer protocol for transferring, accessing and managing files and file systems across open computer systems. FTAM entities operate in either Initiator or Responder mode. FTAM also defines a Virtual File Store (VFS) to enable transfers between computers with different internal file systems. The VFS presents an abstract file system with generic operations which are mapped to the system dependent operations. Fig 6.1 shows the place of FTAM in the Application Layer.

The AP or Application Process is the actual user of the services of FTAM. AP is part of the real operating system and could be either a user program or a human being. AE or application Entity is the program in a system that deals with the lower layers. UE or User Element is the interface between the user and the AE. ASE or Application Service Element is an implementation of one of the typical applications such as X.400 (electronic mail) or in this case, FTAM. SACF or the Single Association Controlling Function routes the AP to the appropriate ASE. At a given time, an AP works with a single ASE. The ACSE or Association Control Service Element is a special

ASE that facilitates the opening and releasing the association between two AEs that communicate by means of a presentation connection.

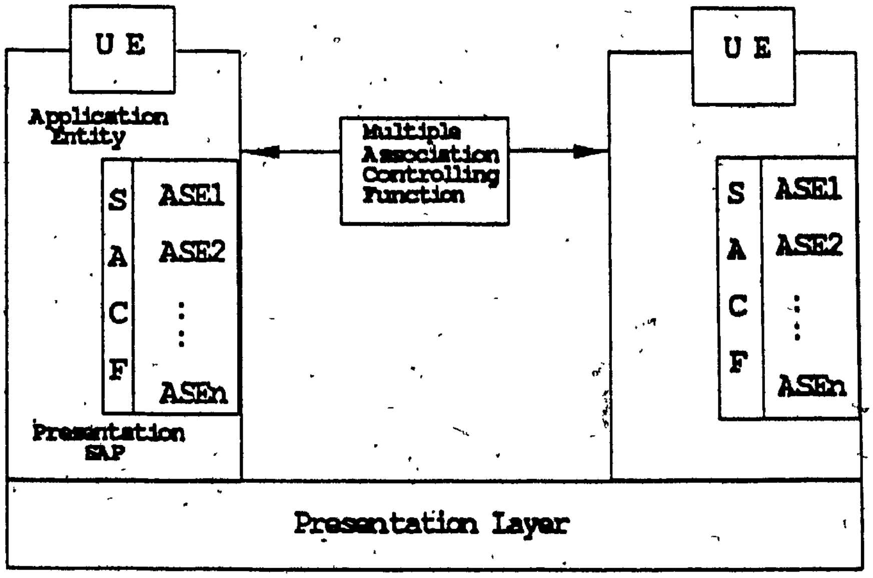


Fig. 6.1. FTAM Model.

The behaviour of the FTAM entity can be divided into four phases called *regimes* in application layer terminology.

- Application Association Regime : This regime exists during FTAM association establishment and release. During this regime, the FTAM entities communicate via the SAP with the ACSE.

- File Selection Regime : During this regime, the selection of a particular file to be associated with the application is carried out.

- File Open Regime : During this period, particular sets of presentation contexts, concurrency and commitment controls are selected to be in force through the operations.

- Data Transfer Regime : This regime exists for the period during which a particular access context and direction of transfer are in force. File transfer and other file operations are performed in this regime.

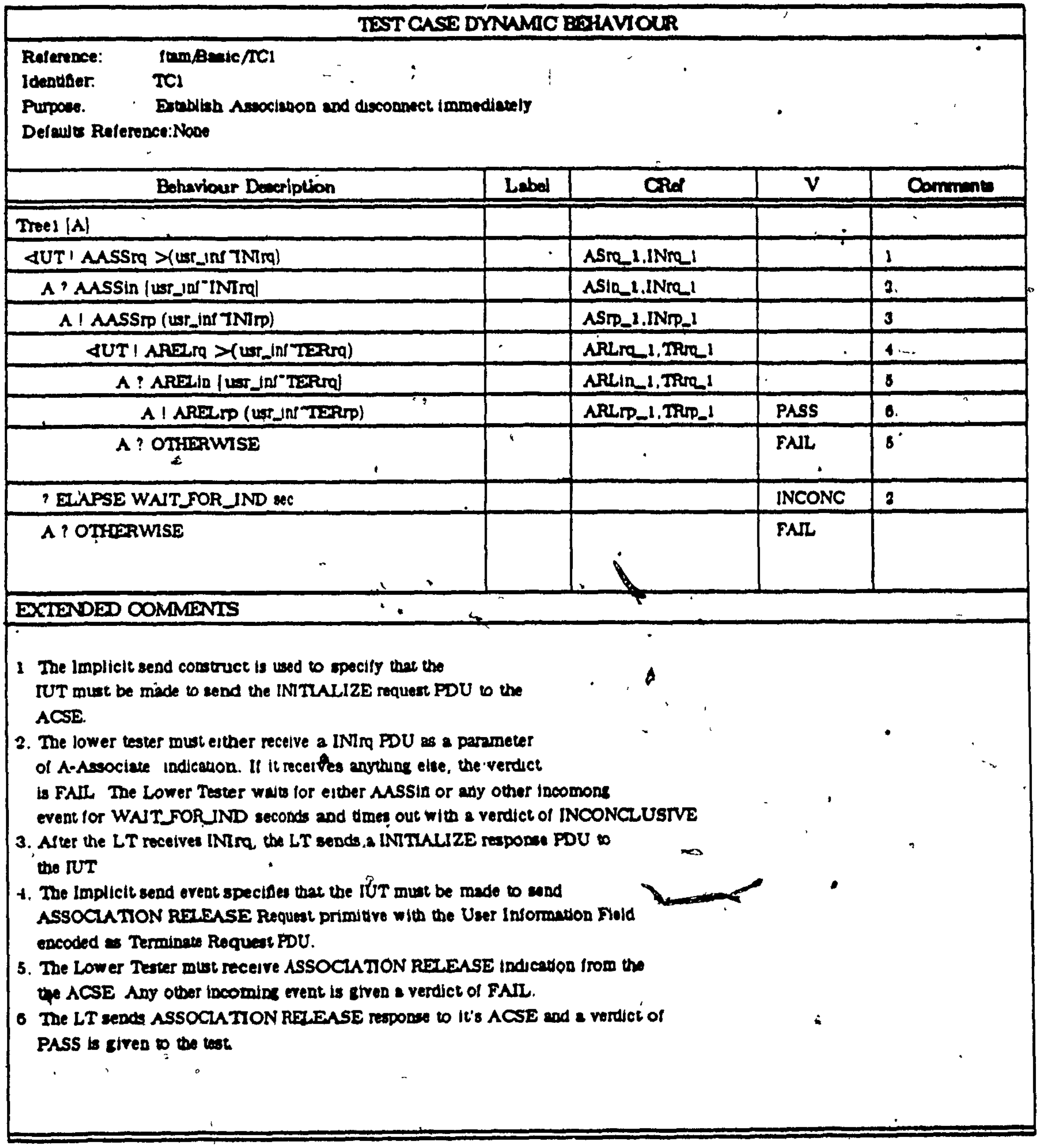
The last three regimes use the SAP with the presentation layer. Every regime has dual operations, i.e, for File Selection, there is a corresponding File Deselection after the end of the operation.

6.1.1 FTAM Test case

The example test case shown here uses the Remote Test Architecture. For an implementation of the Initiator type, DS architecture could also be used. For an implementation of the Responder, only the Remote Test Architecture is indicated because the service user in this case is the Virtual File Store and the service boundary may not be accessible.

The IUT in this test case is an implementation of the Initiator side of FTAM. The main sequence is for the IUT to establish an association and release it immediately. This sequence belongs to the Application Association Regime and thus uses only the SAP between the Lower Tester and the ACSE. This SAP is referred to as *A* in the test case. This sequential progress may be thwarted any time by an Abort from the user or the service provider. The test purpose is to check whether the IUT is capable of establishing and releasing associations by means of the ACSE.

Fig. 6.2 shows the Test Case in TTCN-GR. Fig. 6.3 shows an example ASN.1 constraint which has been placed on the *INIrq* PDU to be sent by the IUT. Application Layer PDUs lend themselves to ASN.1 constraint specification because the ASN.1 definitions of most Application layer PDUs are available in the standards. It is assumed here that the values given to the various fields are of predefined or previously defined User Types.



TEST CASE DYNAMIC BEHAVIOUR

Reference: ftam/Basic/TC1
Identifier: TC1
Purpose. Establish Association and disconnect immediately
Defaults Reference:None

Behaviour Description	Label	CRef	V	Comments
Tree1 [A]				
<UT ! AASSrq >(usr_inf^INIrq)		ASrq_1,INrq_1		1
A ? AASSin [usr_inf^INIrq]		ASin_1,INrq_1		2.
A ! AASSrp (usr_inf^INIrp)		ASrp_1,INrp_1		3
<UT ! ARELrq >(usr_inf^TERrq)		ARLrq_1,TRrq_1		4
A ? ARELin [usr_inf^TERrq]		ARLin_1,TRrq_1		5
A ! ARELrp (usr_inf^TERrp)		ARLrp_1,TRrp_1	PASS	6.
A ? OTHERWISE			FAIL	5
? ELAPSE WAIT_FOR_IND sec			INCONC	2
A ? OTHERWISE			FAIL	

EXTENDED COMMENTS

1 The implicit send construct is used to specify that the IUT must be made to send the INITIALIZE request PDU to the ACSE.
2. The lower tester must either receive a INIrq PDU as a parameter of A-Associate indication. If it receives anything else, the verdict is FAIL The Lower Tester waits for either AASSin or any other incomong event for WAIT_FOR_IND seconds and times out with a verdict of INCONCLUSIVE
3. After the LT receives INIrq, the LT sends a INITIALIZE response PDU to the IUT
4. The implicit send event specifies that the IUT must be made to send ASSOCIATION RELEASE Request primitive with the User Information Field encoded as Terminate Request PDU.
5. The Lower Tester must receive ASSOCIATION RELEASE indication from the the ACSE Any other incoming event is given a verdict of FAIL.
6 The LT sends ASSOCIATION RELEASE response to it's ACSE and a verdict of PASS is given to the test.

Fig. 6.2. FTAM Test Case.

ASN.1 VALUE CONSTRAINT DECLARATION	
PDU Name : INIrq	Constraint Name: INIrq_1
ASN.1 Value	

```
INIrq_1 ::=[0] IMPLICIT SEQUENCE {
     protocol-id  ProtVersion,                    -- These values have been defined
     presentation-context-management [1] IMPLICIT BOOLEAN
           FALSE,      --  previously using ASN.1 User Types
     service-level    reliable,
     service-class   management-class,
     attribute-groups  AttGroups,
     calling-address UT,
     called-address  LT,
     checkpoint-window [8] IMPLICIT INTEGER 1
     }
```

Fig. 6.3. FTAM ASN.1 Constraint.

6.1.2 TTCN-MP specification of Test Case

Below is given the TTCN-MP specification of the FTAM test case. The two specifications are semantically equivalent.

$DynamicPart

$TestCases

$TestGroup
$TestGroupId Basic
$Begin_TestCase

$TestCaseRef ftam/Basic/TC1
$TestCaseId TC1
$TestPurpose
Establish Association and disconnect immediately
$DefaultsRef None
$BehaviourDescription

$TreeHeader Tree1 [A]
$BehaviourLine $Line [1] <UT ! AASSrq >
(usr_inf^INIrq) $CRef ASrq_1 , INrq_1

$End_BehaviourLine
$BehaviourLine $Line [4] A ? AASSin
[usr_inf^INIrq] $CRef ASin_1 , INrq_1

$End_BehaviourLine

```
$BehaviourLine $Line [7] A ! AASSrp
(usr_inf INIrp) $CRef ASrp_1 , INrp_1

$End_BehaviourLine
$BehaviourLine $Line [11] <IUT ! ARELrq >
(usr_inf TERrq) $CRef ARLrq_1 , TRrq_1

$End_BehaviourLine
$BehaviourLine $Line [16] A ? ARELin
[usr_inf TERrq] $CRef ARLin_1 , TRrq_1

$End_BehaviourLine
$BehaviourLine $Line [20] A ! ARELrp
(usr_inf TERrp) $CRef ARLrp_1 , TRrp_1
 $Verdict PASS
 $End_BehaviourLine
$BehaviourLine $Line [16] ? OTHERWISE
 $Verdict FAIL
 $End_BehaviourLine
$BehaviourLine $Line [4] ? OTHERWISE
 $Verdict FAIL
 $End_BehaviourLine
$BehaviourLine $Line [4] ? ELAPSE WAIT_FOR_IND sec $Verdict INCONC
 $End_BehaviourLine

$End_BehaviourDescription

$End_TestCase
$End_TestGroup
$End_TestCases

$End_DynamicPart
```

6.2 Transport Class 4 (TP4)

This is a protocol at layer 4 of the OSI seven layer model. TP4 [19] is Transport Class 4, which means that it is suitable over unreliable network services. If the network service provider uses datagrams, the transport layer has to take care of sequencing and lost packets. This is the most complex of the OSI Transport protocols with a wide variety of options and functions. Multiplexing over network SAPs and concatenation of NSDUs are some of the functions which are supported by Class 4 TP.

The Transport Layer provides services to the Session Layer and requires services from the Network layer. Thus, the IUT will have a SAP between Session and itself and a SAP between the Network and itself.

Either Distributed Architecture or Coordinated Architecture can be used for transport protocols. In the example, a test case is given in both architectures. The purpose of the test case is to determine whether the IUT can support a Transport Protocol Data Unit size of 8192. The test is done by trying all the possible values of TPDU size parameter of the Connection Request PDU.

6.2.1 TP4 DS Test Case

Fig. 6.4 gives the Test Case specified in the Distributed Architecture and Fig. 6.5 shows the same Test Case specified using the Coordinated Architecture. The procedure is quite straight forward in both the tests. The Lower Tester attempts to establish a transport connection with the IUT after establishing the Network connection specified in the Test Step *NCON_L1*. On receiving the Connection Confirm PDU, the TPDU size parameter is checked to see if the IUT has agreed to accept the legal value. The Test Step *SIZE* is attached at this point in the tree. This Test Step accepts two parameters, the TPDU size parameter of the sent CR PDU and the TPDU size parameter of the received CC PDU. A simple Boolean expression could have replaced *SIZE* but a Test Step is used because, if necessary, more parameters could be passed, and more complex expression evaluated without disturbing the structure of the Test Case. If *SIZE* returns a value *pass* stored in the global integer variable *result*, a preliminary verdict of *(PASS)* is given to the Test Case and the value of the TPDU size is increased and the test repeated. At the end of the Test Case, the final verdict is given as RESULT, which evaluates to the value stored in the predefined variable *$R*. This variable contains the last preliminary verdict.

TEST CASE DYNAMIC BEHAVIOUR

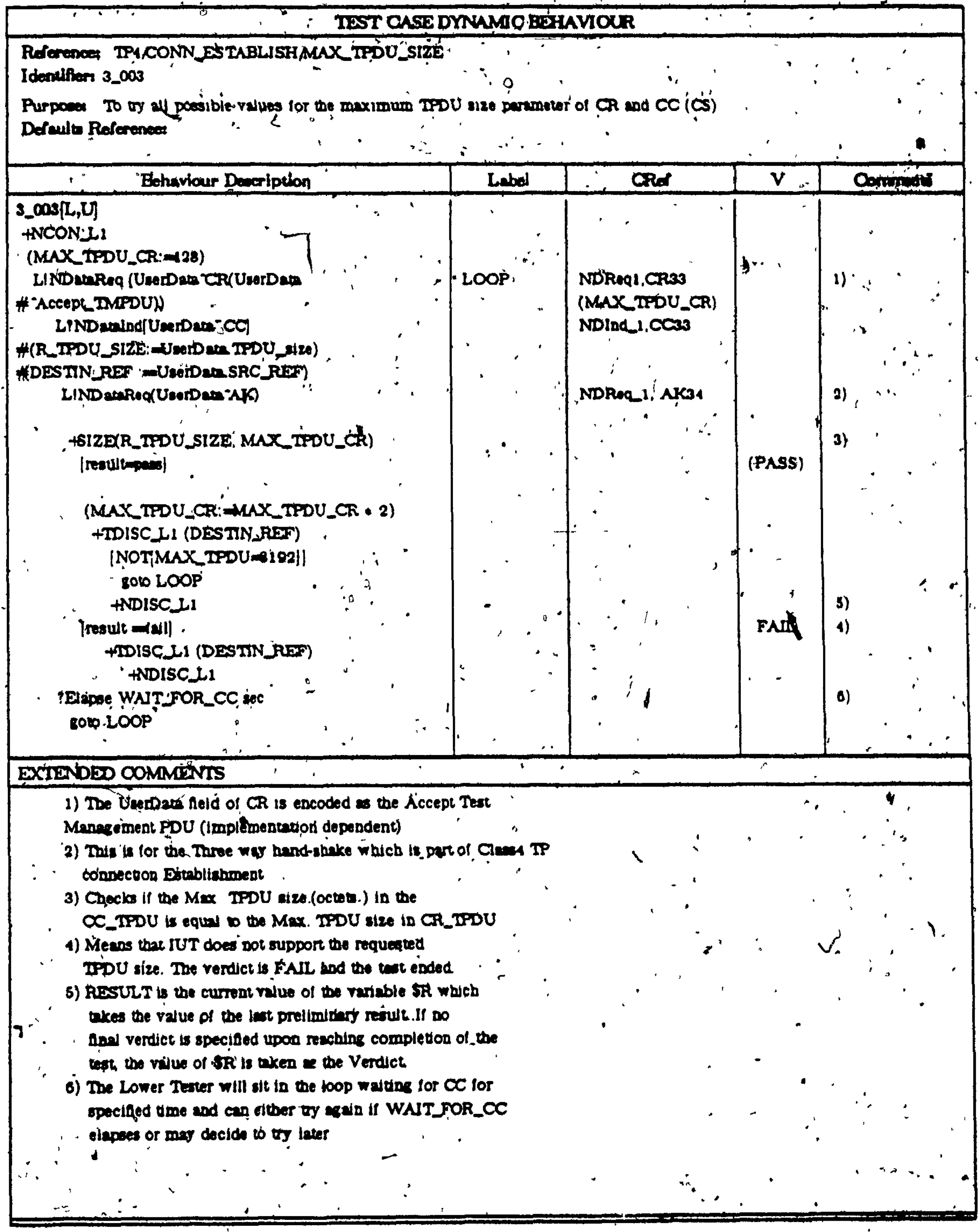
Reference: TP4/CONN_ESTABLISH/MAX_TPDU_SIZE
Identifier: 3_003
Purpose: To try all possible values for the maximum TPDU size parameter of CR and CC (DS)
Defaults Reference:

Behaviour Description	Label	CRef	V	Comments
3_003[L,U]				
+NCON_L1				
(MAX_TPDU_CR:=128)				
L!NDataReq(UserData^CR)(UserData	LOOP	NDReq1, CR1		
#.TPDU_size:=MAX_TPDU_CR)		(MAX_TPDU_CR)		
U?TConInd		TCind1		
U!TConResp		TCresp1		
L?NDataInd(UserData^CC)		NDind1, CC1		1)
#(R_TPDU_SIZE:=UserData.TPDU_size)				
#(DESTIN_REF :=UserData.SRC_REF)		NDReq1, AK34		
L!NDataReq(UserData^AK)				2)
+SIZE(R_TPDU_SIZE, MAX_TPDU_CR)				3)
[result=pass]			(PASS)	
(MAX_TPDU_CR:=MAX_TPDU_CR * 2)				
+TDISC_L1(DESTIN_REF)				
[NOT(MAX_TPDU=8192)]				
goto LOOP				
+NDISC_L1				6)
[result=fail]			FAIL	4)
+TDISC_L1(DESTIN_REF)				
+NDISC_L1				
?OTHERWISE			FAIL	
+NDISC_L1				
? ELAPSE WAIT_FOR_CR sec				5)
goto LOOP				

EXTENDED COMMENTS

1) Save the Source Reference of the received CC PDU for Transport Disconnection
2) Three way Handshake necessary for Class 4 TP
3) Checks if the Max. data recd.(octets) in the CC_TPDU is equal to the Max. data in CR_TPDU
4) Means that IUT does hot support the requested TPDU size
5) WAIT_FOR_CR is a previously negotiated value which is set depending on the quality of the network service and other considerations If ELAPSE occurs, it may be either due to IUT or in the network. So either a verdict of INCONC may be given or the loop tried again.
6) If the Test Case comes to this test step, the Verdict is the same as the preliminary verdict because none is specified. A verdict cannot be assigned to an Attach statement so by default, a Verdict equal to $R is given to the Test Case.

Fig. 6.4. TP4 DS Test Case.



TEST CASE DYNAMIC BEHAVIOUR

Reference: TP4/CONN_ESTABLISH/MAX_TPDU_SIZE

Identifier: 3_003

Purpose: To try all possible values for the maximum TPDU size parameter of CR and CC (CS)

Defaults Reference:

Behaviour Description	Label	CRef	V	Comments
3_003[L,U]				
+NCON_L1				
(MAX_TPDU_CR:=128)				
L!NDataReq (UserData CR(UserData	LOOP	NDReq1,CR33		1)
# Accept_TMPDU))		(MAX_TPDU_CR)		
L?NDataInd[UserData CC]		NDInd_1,CC33		
#(R_TPDU_SIZE:=UserData.TPDU_size)				
#(DESTIN_REF :=UserData.SRC_REF)				
L!NDataReq(UserData AK)		NDReq_1, AK34		2)
+SIZE(R_TPDU_SIZE, MAX_TPDU_CR)				3)
[result=pass]			(PASS)	
(MAX_TPDU_CR:=MAX_TPDU_CR * 2)				
+TDISC_L1 (DESTIN_REF)				
[NOT[MAX_TPDU=8192]]				
goto LOOP				
+NDISC_L1				5)
[result =fail]			FAIL	4)
+TDISC_L1 (DESTIN_REF)				
+NDISC_L1				
?Elapse WAIT_FOR_CC sec				6)
goto LOOP				

EXTENDED COMMENTS

1) The UserData field of CR is encoded as the Accept Test Management PDU (implementation dependent)

2) This is for the Three way hand-shake which is part of Class4 TP connection Establishment

3) Checks if the Max TPDU size (octets) in the CC_TPDU is equal to the Max. TPDU size in CR_TPDU

4) Means that IUT does not support the requested TPDU size. The verdict is FAIL and the test ended

5) RESULT is the current value of the variable $R which takes the value of the last preliminary result. If no final verdict is specified upon reaching completion of the test, the value of $R is taken as the Verdict

6) The Lower Tester will sit in the loop waiting for CC for specified time and can either try again if WAIT_FOR_CC elapses or may decide to try later

Fig. 6.5. TP4 CS Test Case.

6.2.2 TP4 CS Test Case

The Coordinated Test Case uses only the SAP L between the Lower Tester and the Network. The Lower Tester attempts to set up a connection with the IUT using a CR TPDU whose user data parameter has been encoded as the *Accept_TMPDU*. This Test Management PDU is recognized by both systems and is assumed to have been previously negotiated and standardized. It tells the non-standard Upper Tester to accept this connection. The test of the TPDU size parameter is performed in a manner similar to the DS Test Case. It is assumed that the Test Step *SIZE* is a member of the global Test Step Library, and thus accessible to both Test Cases. Had the Test Step been defined in one Test Case following the main Tree, it would be invisible to other Test Cases.

CHAPTER 7 TEST GENERATION AND EXECUTION

INTRODUCTION

This Chapter describes a test sequence generation tool called CONTEST-ESTL [20] and the translation of these sequences to the abstract test notation, TTCN. The execution of TTCN tests is also discussed and some possible methods of achieving this aim are introduced.

7.1 CONTEST-ESTL

The ESTELLE based test design methodology is based on functional program testing of software. The protocol specification in ESTELLE is analyzed and transformed so that it allows the decomposition of the specification into its functions. This functional analysis is the basis of the test design methodology in [21].

A specification in ESTELLE can be modeled using graphs, one representing the flow of control and the other representing the flow of data. Functions of the protocol can be identified from the data flow graphs. It is easy to derive the data flow graphs if the specification is first transformed into an equivalent form containing only normal form transitions.

7.1.1 Transformations.

The following constructs are removed in the normal form representation of the ESTELLE specification.

- Variant Records
- Major state lists in FROM and TO clauses.
- Conditional IF and CASE statements.
- Procedure/function calls.

Variant records are linearized by tagging each type by the tag used used in the variant record definition. Major state lists are eliminated in the FROM TO clauses by generating a normal form transition (NFT) corresponding to each possible state value. Symbolic execution of sequential programs is used to remove conditional statements and local procedure and functional calls. A new path is created for every distinct path in the BEGIN block and the PROVIDED clauses are modified to reflect the conditions for taking those paths. Local procedure and function calls are eliminated by symbolically executing them. Similar action is taken with IF and CASE statements.

As an example, consider a transition from the specification of the FTAM protocol.

```
trans
 when F.FINIrq
  { F-INITIALIZE primitive acceptable }
  provided (level=usr_corr) and (class=transf_class)
  from CLOSED to INITIALIZE_PD
  begin
   INIrqPDU( { local procedure (subroutine) }
    { in }
    prot_id, pres_cont_man, level, class, units, att_groups,
    rollback, contents, in_id, acc, fs_passw, checkp_wind,
    { out }
    PDU);
   output A.AASSrq(
   0 { Version1 }, CalledAET, CallingAET, ISO_8571_FTAM, PDU);
  end;
```

This would be normalized to the NFT named number 1 :

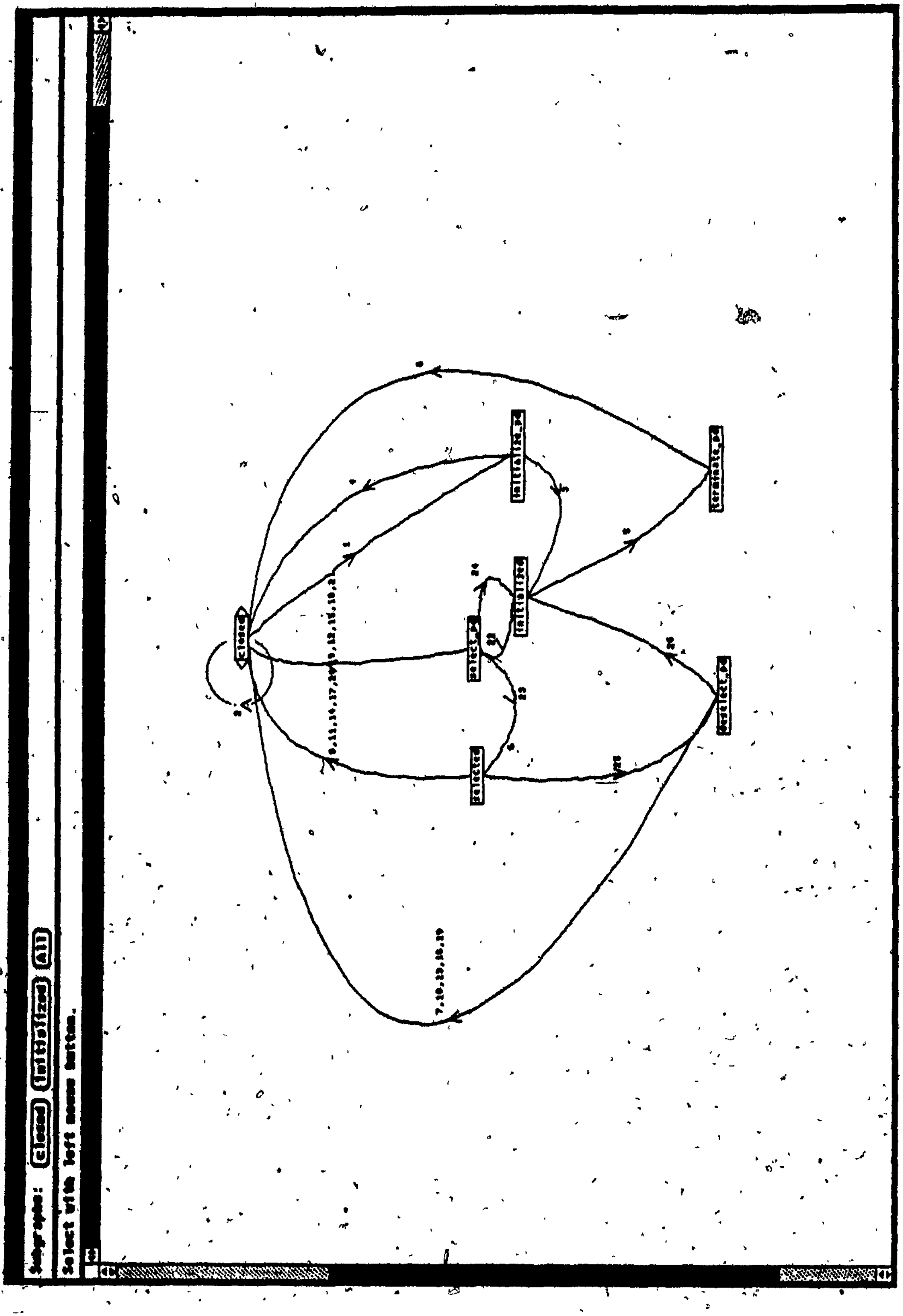
```
    trans
```

```
{ 1 }
when f.finirq
provided (level =usr_corr) and (class =transf_class),
from closed
to initialize_pd
begin
   pdu_inirq.v0.prot_id :=prot_id;
   pdu_inirq.v0.pres_cont_man :=pres_cont_man;
   pdu_inirq.v0.level :=level;
   pdu_inirq.v0.class :=class;
   pdu_inirq.v0.units :=units;
   pdu_inirq.v0.att_groups :=att_groups;
   pdu_inirq.v0.rollback :=rollback;
   pdu_inirq.v0.contents :=contents;
   pdu_inirq.v0.in_id :=in_id;
   pdu_inirq.v0.acc :=acc;
   pdu_inirq.v0.fs_passw :=fs_passw;
   pdu_inirq.v0.checkp_wind :=checkp_wind;
   output a.aassrq_inirq(0, calledaet, callingaet,
             iso_8571_ftam, pdu_inirq)
end;
```

As can be seen, procedure call INIrqPDU has been eliminated and its body replaced within the transition. PDU was originally defined as a variant record with a tag for all the types of PDUs. These tags are removed and a record defined for each type. Thus a PDU whose type was defined as INIrq would be converted to pdu_inirq. ESTELLE is not case sensitive and so case is not significant. Had there been some control statements in the original transition, it would have produced more than one NFT. In this case, the transition is mapped to a single normalized transition.

7.1.2 Flow graphs.

The control flow graphs show the changes of the value of the major state variable. This is the Finite State Machine model of the protocol. It is easily constructed from the FROM TO clauses in the NFTs. The sequences which start and end in the initial state are called **Subtours.** Fig. 7.1 shows the control flow graph derived from a specification of the FTAM kernel functional unit. The arcs are labeled with the NFT numbers associated with transitions from the starting state to the resulting state. A sample subtour would be the sequence 1 3 5 6.



Select with left mouse button.

Fig. 7.1. FTAM Kernel Control Flow Graph.

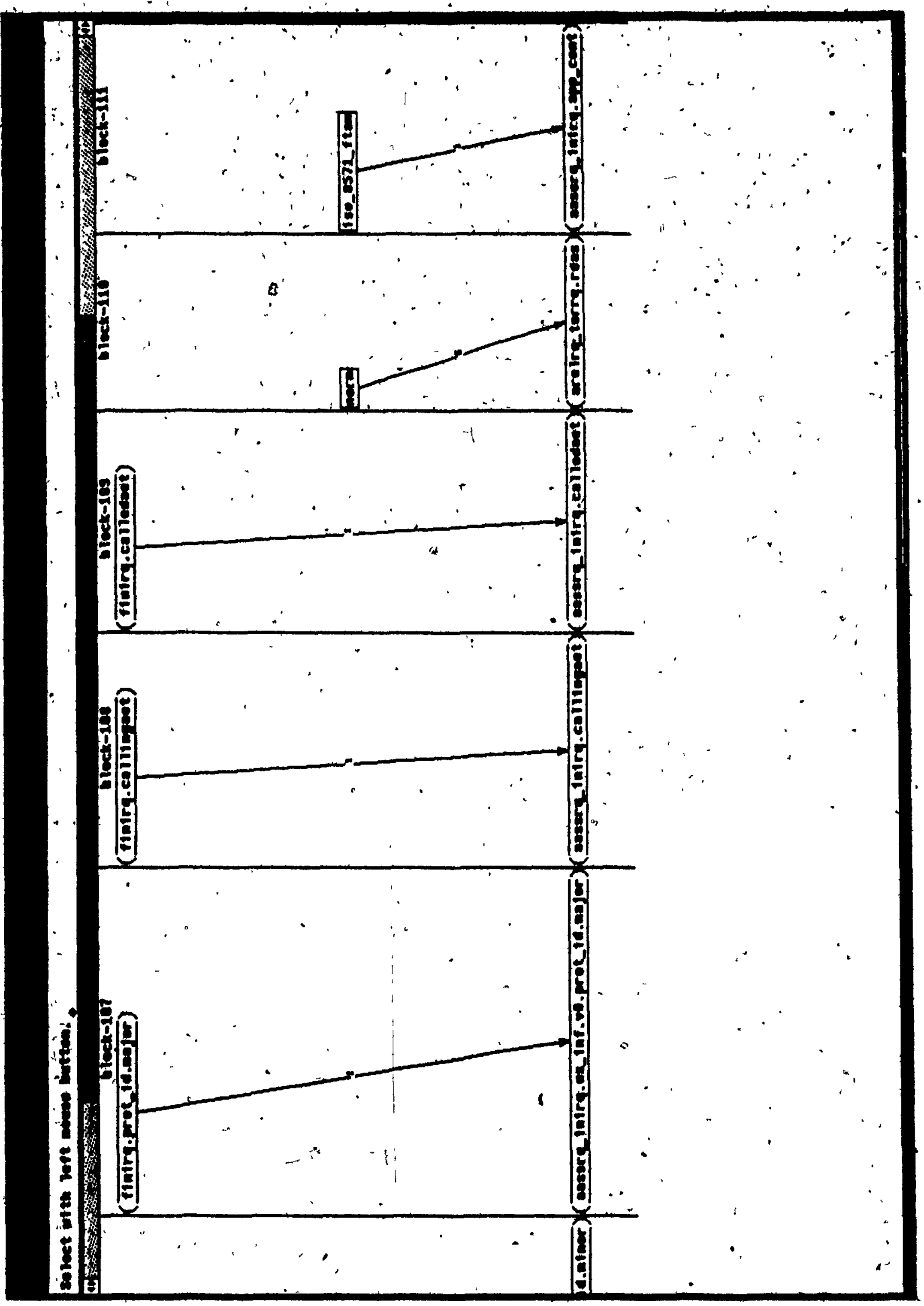


Fig. 7.2. FTAM Kernel Data Flow Graph.

The data flow graph models the flow of information in a normal form specification. There are four types of nodes defined - I,O,D and F (input, output, context variables or constants and data operations or functions). The data flow graphs are partitioned into blocks, each representing the flow over one context variable. Some of these blocks can be merged to represent all the protocol functions like addressing, disconnection, data transfer etc. Fig. 7.2 shows an example data flow graph of the same specification as above. Nodes at the top of the graph represent the input nodes of transitions and nodes at the bottom of the graph represent Output nodes. The rectangular nodes in the middle of the graph represent data nodes or D-nodes and the nodes with diamond sides represent the function nodes or F-nodes.

7.1.3 Test methodology.

From the data flow graph, a block representing a function is selected. This is tested using all the subtours of the block. A subtour belongs to a block if it contains the NFT number belonging to a block. The input primitive parameters that belong to a function are determined. These are enumerated while all others are kept fixed to certain values. Next, the output primitives and parameters corresponding to the subtour selected and the function to be tested are determined. The expected values of the remaining parameters are determined from the fixed values of the other parameters. This methodology covers all the control paths in the specification and verifies the data flow in each function.

7.2 An example

In this section, a specification of the FTAM kernel functional unit is taken and the above test methodology applied in order to generate unparametrized test sequences for the testing of implementations. The methodology of translating such sequences to parametrized TTCN test cases is explained, along with some examples.

The example specification under consideration is a subset of the File Transfer, Access and Management protocol. This protocol has, among others, three major functional units :

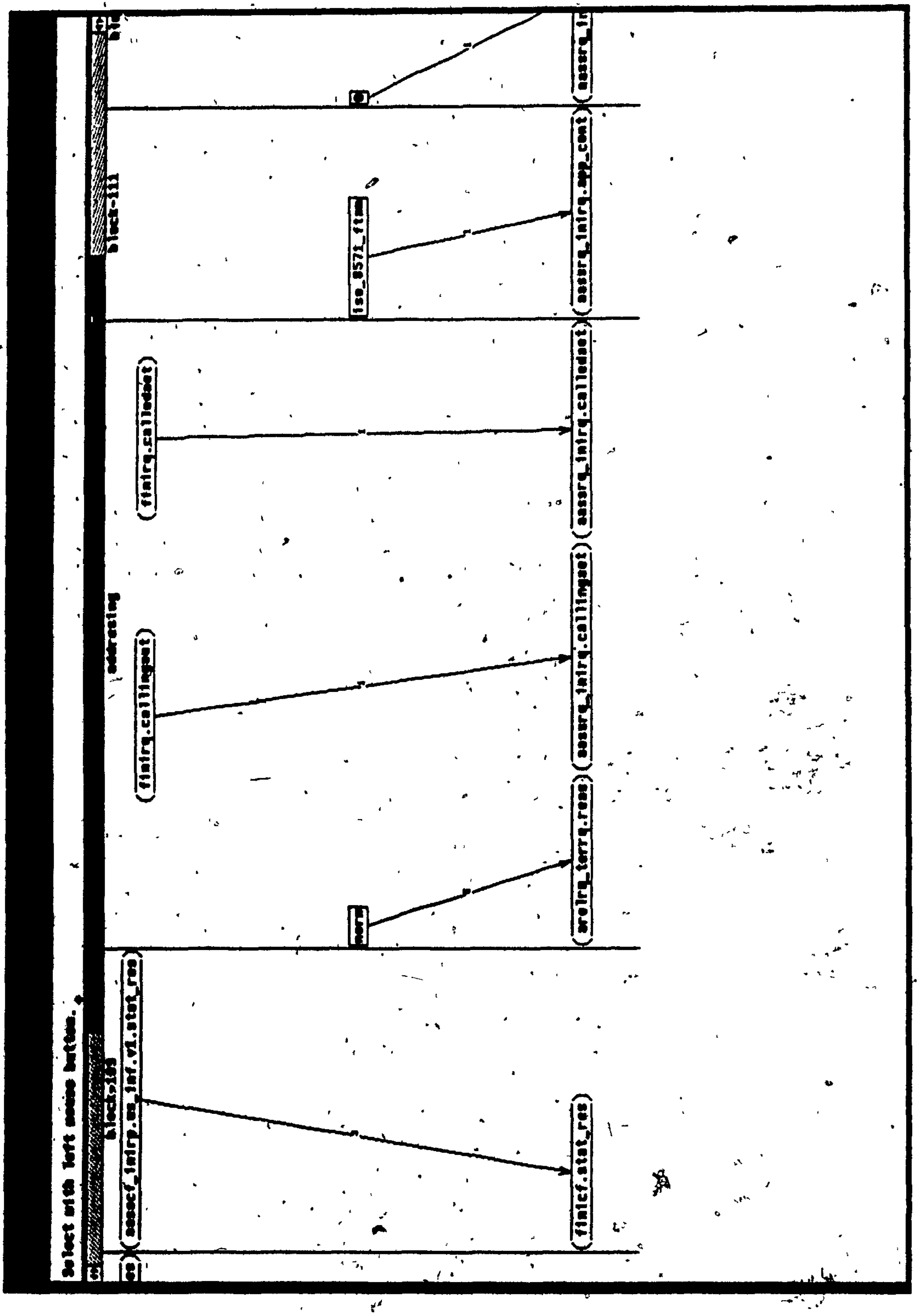
- Kernel Unit
- Read Unit
- Write Unit

The kernel unit takes care of Association Establishment with the peer implementation and negotiates such parameters as passwords, cost, etc., before proceeding to the other file transfer functions. The test methodology has been applied to the kernel unit specification and the resulting data flow was decomposed into fourteen functional blocks. Each block includes a number of possibly overlapping normal form transitions. Subtours which are obtained from the control flow graphs of the same specification are selected so that the coverage of each block is complete.

7.2.1 Subtours and Test Sequences.

As an example consider the functional block named *Addressing*. The naming and creation of functions is based on a knowledge of the protocol in question and the nodes contained within. Fig. 7.3 shows the data flow graph showing the functional block *Addressing*. This block contains transitions numbered 1 and 5. Subtours which traverse these transitions are chosen as appropriate testing sequences for this block. Thus, the subtours selected from this function are those that cover these transitions and are given below.

```
Name of Function =>addressing

      2 subtours
subtour : 0

closed          f.finirq          [a.aassrq_inirq]      1
initialize_pd    a.aasscf_inirp     [f.finicf]          3
initialized     f.fterrq          [a.arelrq_terrq]      5
terminate_pd      a arelcf_terrp      [f.ftercf]         6
```



Select with left mouse button.

Fig. 7.3. Addressing Block.

subtour : 1

closed	*f.finirq*	*[a.aassrq_inirq]*	*1*
initialize_pd	*a.aasscf_inirp*	*[f.finicf]*	*3*
initialized	*f.fselrq*	*[p.pdatrq_selrq]*	*22*
select_pd	*p.pdatin_selrp*	*[f.fselcf]*	*23*
selected	*f.fdesrq*	*[p.pdatrq_desrq]*	*25*
deselect_pd	*p.pdatin_desrp*	*[f.fdescf]*	*26*
initialized	*f.fselrq*	*[p.pdatrq_selrq]*	*22*
select_pd	*p.pdatin_selrp*	*[f.fselcf]*	*24*
initialized	*f.fterrq*	*[a.arelrq_terrq]*	*5*
terminate_pd	*a.arelcf_terrp*	*[f.ftercf]*	*6*

The format of the subtours is one line per sequence. The sequence is given in this format :

Initial State Input Output(s) NFT-number

Inputs and outputs are preceded by the interaction point of the module which could be represented by Service Access Points for TTCN specifications. The initial state of the following sequence gives the final state for the current sequence. Thus, for subtour 0 for this function, line 1 stands for the transition whose NFT number is 1, whose starting state is *closed*, whose final state is *initialized_pd*, whose input interaction occurs at the SAP named *f* and whose output occurs at the SAP named *a*. The output could be a list of events, possibly at different SAPs. It will be noticed that all identifiers consist of only lower case letters and that some primitives and PDUs are given suffixes. These are the product of normalization of variant records, which gives rise to unique record names. It is quite straight forward to map to the real ASPs and PDUs from the normalized specification to the original specification.

7.2.2 Translation to TTCN

The translation to TTCN of the test sequences can be done by following the steps given below :

- The ESTELLE specification is written from the point of view of the IUT with the SAPs being its external interaction points. So, in order to convert the subtours obtained from the specification to test sequences, Inputs and Outputs are reversed and the resulting model maps naturally to the Local Test Architecture. Inputs to the IUT from the SAP between the Service User and the IUT map to the test events to be sent by the Upper Tester and the outputs from the IUT at the interaction point between the Service Provider and the IUT become test events observed by the Lower Tester. Similarly, inputs to the IUT at the lower SAP become the test events controlled by the Lower tester and outputs by the IUT at the upper SAP are the test events observed by the Upper Tester.

- The above step is straight forward when dealing with transitions which contain both input and output. For transitions which do not contain input but use the *Delay* clause, a different procedure is adopted. A *Delay* clause in ESTELLE specifies the length of time the machine will remain in a state before performing some action, unless the necessary input comes in before the time specified runs out. This clause may be used, for example, in specifying the action to be taken if an Acknowledgement for a PDU is not received in time.

For the translation to TTCN, a START timer clause is invoked with a value equal to the minimum value in the *Delay* clause. The next level of indentation is the TIMEOUT statement followed by the output, if any from the IUT.

- This role reversal produces the bare skeleton of the intended path of the Test Case. Next, all those transitions which may occur in the IUT spontaneously when the IUT is in the same state as the current test sequence need to be specified as alternative events. This is necessary to ensure that normal action of the IUT is not mistaken as unspecified and therefore, as erroneous behaviour.

After specifying possible alternative behaviour from the specification, the next possible alternative is given as *PCO ? OTHERWISE*, where *PCO* is the point of control and observation at which the original event is to occur. This clause is used only if the original event is a Receive event.

- Next, the constraints have to be specified on the transmitted and received PDUs. The *Provided* clause in the ESTELLE transition may define the constraints on the ASP or PDU to be sent to the IUT because the *Provided* clause may specify constraints on the parameter of the *When* clause. If the parameters of the *Provided* clause do not have anything to do with the parameter of the *When* clause, these constraints maybe on the general firing constraints on the transitions such as the state of context variables, etc. In such a case, it must be expected that the transition may not fire and the sequence continued because the context variables are not within the direct control of the Upper or Lower Tester. The assignment statements within the body of the transition specify the constraints on the ASP or PDU to be transmitted by the IUT, and correspondingly, on the ASP or PDU to be received by either the Upper or Lower Tester.

The above steps may be used to translate a sequence to TTCN but if loops or iterations are needed, these have to be done by the test designer. The test sequences obtained from the test generation tool CONTEST-ESTL do not contain loops but rather specify a particular path through the control tree. This method can be automated to a large extent with the test designer supplying some steps which are subtle such as retransmission times for ELAPSE statements, etc, which are implementation dependent and require a deeper knowledge of the protocol.

7.2.3 An Example

The example specification here is the FTAM specification of the kernel functional unit. The function under test is the Addressing function which is part of the Association Establishment phase. One of the subtours which is used to test this function is given below.

SUBTOUR

closed	*f.finirq*	*[a.aassrq_inirq]*	*1*
initialize_pd	*a.aasscf_inirp*	*[f.finicf]*	*3*
initialized	*f.fterrq*	*[a.arelrq_terrq]*	*5*
terminate_pd	*a.arelcf_terrp*	*[f.ftercf]*	*6*

The Normal Form Transitions corresponding to the transitions in this subtour are given below to serve as a reference for the example. The final output of the test generation tool is in the form shown above. Translation to the TTCN equivalent is the next step, with as much automation as possible.

Normal Form Transitions

```
trans
     { 1 }
     when f.finirq
     provided (level =usr_corr) and (class =transf_class)
     from closed
     to initialize_pd
     begin
        pdu_inirq.v0.prot_id :=prot_id;
        pdu_inirq.v0.pres_cont_man :=pres_cont_man;
        pdu_inirq.v0.level :=level;
        pdu_inirq.v0.class :=class;
        pdu_inirq.v0.units :=units;
        pdu_inirq.v0.att_groups :=att_groups;
        pdu_inirq.v0.rollback :=rollback;
        pdu_inirq.v0.contents :=contents;
        pdu_inirq.v0.in_id :=in_id;
        pdu_inirq.v0.acc :=acc;
        pdu_inirq.v0.fs_passw :=fs_passw;
        pdu_inirq.v0.checkp_wind :=checkp_wind;
        output a.aassrq_inirq(0, calledaet, callingaet, iso_8571_ftam,
           pdu_inirq)
     end;

trans
     { 3 }
     when a.aasscf_inirp
     provided ((protvers =0) and (res =ass_acc) and
        (us_inf.v1.stat_res =succ_res) and (us_inf.v1.act_res =succ_act))
     from initialize_pd
     to initialized
     begin
        output f.finicf(us_inf.v1.stat_res, us_inf.v1.act_res,
          us_inf.v1.prot_id, us_inf.v1.pres_cont_man, us_inf.v1.units,
          us_inf.v1.attr_groups, us_inf.v1.rollback, us_inf.v1.contents,
          us_inf.v1.diag, us_inf.v1.checkp_wind)
     end;
```

```
trans
    { 5 }
    when f.fterrq
    from initialized
    to terminate_pd
    begin
        output a.arelrq_terrq(norm, pdu_terrq)
    end;

trans
    { 6 }
    when a.arelcf_terrp
    from terminate_pd
    to closed
    begin
        output f.ftercf(us_inf.v8.charg)
    end;
```

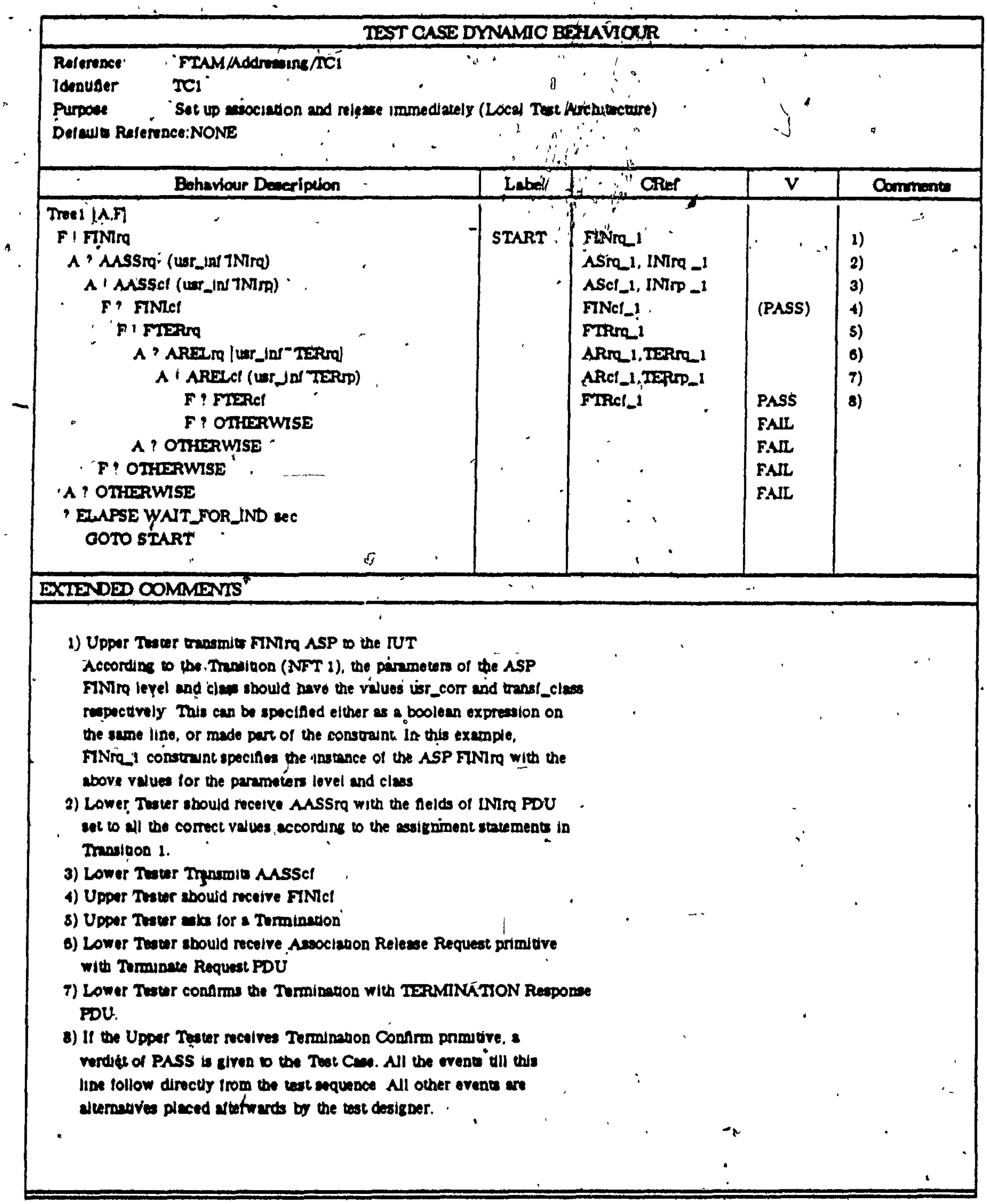
In transition number 1, the IUT receives from the SAP at *f* the primitive FINIrq. The provided clause checks the two parameters of this primitive so these would serve as the constraints on this primitive before it is transmitted to the IUT by the Upper Tester at SAP *f*. The assignment statements within the body of the transition serve as the constraints of the PDU which is a parameter of the AASSrq primitive which is transmitted to the Lower Tester at SAP *a*.

The translation of the remaining sequences is similar and gives rise to the TTCN specification as given in Fig. 7.4.

7.3 Execution of TTCN Test cases

The next important step in the testing process is the actual execution, or the translation of the TTCN cases into an executable form, for performing the tests. The standard document which defines TTCN gives an algorithmic explanation of the semantics of the various TTCN dynamic behaviour constructs in an Annex (discussed in section 4.6). As explained in the Annex, this does not purport to be a method of direct execution, but rather only serves as an aid to understanding the language.

Though it is possible to find a method to directly execute TTCN Test Cases, the conversion of the Test Cases to another Formal Description Technique has some



TEST CASE DYNAMIC BEHAVIOUR

Reference: FTAM/Addressing/TC1
Identifier TC1
Purpose Set up association and release immediately (Local Test Architecture)
Defaults Reference:NONE

Behaviour Description	Label/	CRef	V	Comments
Tree1 [A,F]				
F ! FINIrq	START	FINrq_1		1)
A ? AASSrq (usr_inf INIrq)		ASrq_1, INIrq _1		2)
A ! AASScf (usr_inf INIrp)		AScf_1, INIrp _1		3)
F ? FINIcf		FINcf_1	(PASS)	4)
F ! FTERrq		FTRrq_1		5)
A ? ARELrq [usr_inf TERrq]		ARrq_1, TERrq_1		6)
A ! ARELcf (usr_inf TERrp)		ARcf_1, TERrp_1		7)
F ? FTERcf		FTRcf_1	PASS	8)
F ? OTHERWISE			FAIL	
A ? OTHERWISE			FAIL	
F ? OTHERWISE			FAIL	
A ? OTHERWISE			FAIL	
? ELAPSE WAIT_FOR_IND sec				
GOTO START				

EXTENDED COMMENTS

1) Upper Tester transmits FINIrq ASP to the IUT
According to the Transition (NFT 1), the parameters of the ASP FINIrq level and class should have the values usr_corr and transf_class respectively This can be specified either as a boolean expression on the same line, or made part of the constraint. In this example, FINrq_1 constraint specifies the instance of the ASP FINIrq with the above values for the parameters level and class
2) Lower Tester should receive AASSrq with the fields of INIrq PDU set to all the correct values according to the assignment statements in Transition 1.
3) Lower Tester Transmits AASScf
4) Upper Tester should receive FINIcf
5) Upper Tester asks for a Termination
6) Lower Tester should receive Association Release Request primitive with Terminate Request PDU
7) Lower Tester confirms the Termination with TERMINATION Response PDU.
8) If the Upper Tester receives Termination Confirm primitive, a verdict of PASS is given to the Test Case. All the events till this line follow directly from the test sequence All other events are alternatives placed afterwards by the test designer.

Fig. 7.4. Example FTAM Test Case (Unparametrized).

advantages. For one thing, verification of the tests can be done using techniques already available for the target FDTs and another, execution of the target FDT may be possible. Consider the FDT ESTELLE. A Test Case specified in ESTELLE can be verified against the specification of the standard and an executable form (for example, in the language C) may be obtained using compilers available for this purpose. [22] have investigated the translation of TTCN test suites into LOTOS specifications for the purpose of validating TTCN test suites.

7.3.1 Conversion to ESTELLE

A TTCN test suite may be represented as an ESTELLE specification with each Test Case represented by a module definition with a common declarations part derived from the Declarations and Constraints part of TTCN. The conversion of the declarations and the value instantiation using constraints from TTCN to Pascal (ESTELLE data) structures is, for the most part trivial, and will not be further considered here. The ESTELLE module declarations is of greater interest and the behaviour conversion is considered below. The following paragraphs represent a preliminary investigation of the feasibility of translation and is explained informally giving some examples. The Test Architecture of interest is the Remote Architecture because it allows the specification of one module for the Lower Tester and does not need formal specification of the synchronization issues between the Upper and Lower Testers.

It is assumed that the Test Case has been completely expanded before the conversion procedure : i.e., all abbreviations have been replaced, all *Attach* statements have been replaced by the corresponding Test Steps and all defaults have been appended. *Repeat* statements are also assumed to have been replaced by the use of boolean expressions, *Attach* and *Goto* statements. This leaves only Sending and Receiving Events, Boolean and Arithmetic Pseudo Events, Timer events and Goto statements in the Test Case.

Each Test Case produces an equivalent ESTELLE module specification with a number of transitions and a state list corresponding to the depth of indentation in the Test Case. If the Test Case is seen as a tree with alternative events as siblings, and successive events as child nodes, all events belonging to the same level in the tree will start in the same state, ie., the *From* clause will specify the same state name. The *To* state will be different for the alternative events.

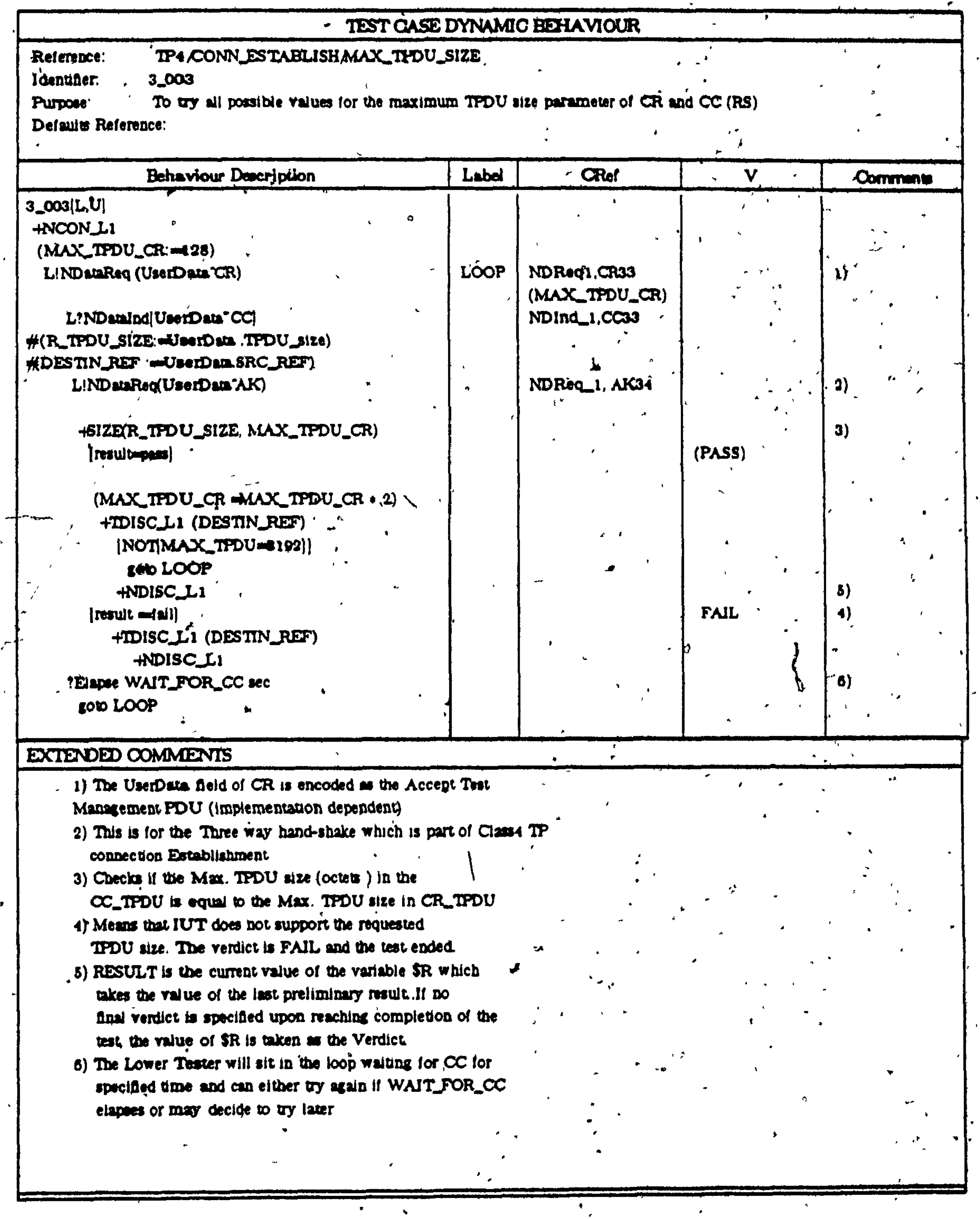
Pseudo Events form spontaneous transitions with boolean expressions, if any, making up the *Provided* Clause. A timing module is assumed to be present for global use with a known interaction point *T*. This timing module accepts the primitives **Start** with the parameters *duration* and *Timer_id*, **Cancel** with parameter *Timer_id*, **Timeout** with the parameter *Timer_id* and **Read** with the parameter *Timer_id*. If a set of alternatives contains an *ELAPSE* event, the parent transition will send a **Start** interaction to the timer module with a duration equal to the amount specified in the *ELAPSE* clause. The *ELAPSE* clause will be replaced by a transition whose *When* clause specifies a *T.Timeout* interaction. A *GOTO* clause is an empty transition block whose *To* clause specifies the state name of the target set of events.

7.3.2 An Example

Consider the TP4 Test Case specified in the Remote Test Architecture in Fig. 7.5. This is the same Test Case for TP4 which was given in Chapter 6 specified in the RS architecture. Assume, for the sake of simplicity that the preamble *NCON_L1* and the postambles *TDISC_L1* and *NDISC_L1* are not present, since they are not part of the main body of the Test Case. Below is given the ESTELLE module equivalent of the above Test Case.

- <u>ESTELLE Test Module :</u>

```
module max_tpdu_size_type process
ip {interaction point list }
L : N_access_point(user) individual queue;
```



TEST CASE DYNAMIC BEHAVIOUR

Reference: TP4/CONN_ESTABLISH/MAX_TPDU_SIZE
Identifier: 3_003
Purpose: To try all possible values for the maximum TPDU size parameter of CR and CC (RS)
Defaults Reference:

Behaviour Description	Label	CRef	V	Comments
3_003[L,U]				
+NCON_L1				
(MAX_TPDU_CR:=128)				
L!NDataReq (UserData^CR)	LOOP	NDReq'1,CR33 (MAX_TPDU_CR)		1)
L?NDataInd[UserData^CC]		NDInd_1,CC33		
#(R_TPDU_SIZE:=UserData.TPDU_size)				
#(DESTIN_REF :=UserData.SRC_REF)				
L!NDataReq(UserData^AK)		NDReq_1, AK34		2)
+SIZE(R_TPDU_SIZE, MAX_TPDU_CR)				3)
[result=pass]			(PASS)	
(MAX_TPDU_CR :=MAX_TPDU_CR * 2)				
+TDISC_L1 (DESTIN_REF)				
[NOT(MAX_TPDU=8192)]				
goto LOOP				
+NDISC_L1				5)
[result :=fail]			FAIL	4)
+TDISC_L1 (DESTIN_REF)				
+NDISC_L1				
?Elapse WAIT_FOR_CC sec				6)
goto LOOP				

EXTENDED COMMENTS

1) The UserData field of CR is encoded as the Accept Test Management PDU (implementation dependent)
2) This is for the Three way hand-shake which is part of Class4 TP connection Establishment
3) Checks if the Max. TPDU size (octets) in the CC_TPDU is equal to the Max. TPDU size in CR_TPDU
4) Means that IUT does not support the requested TPDU size. The verdict is FAIL and the test ended.
5) RESULT is the current value of the variable $R which takes the value of the last preliminary result. If no final verdict is specified upon reaching completion of the test, the value of $R is taken as the Verdict.
6) The Lower Tester will sit in the loop waiting for CC for specified time and can either try again if WAIT_FOR_CC elapses or may decide to try later

Fig. 7.5. Example TP4 Test Case.

```
T : Timer_access_point(user) common queue;
end;

body max_tpdu_size_body for max_tpdu_size_type;

{Type definitions }
{Any event which assigns a final verdict brings the machine to E_state} state
state_1,state_2,state_3,state_4.1,state_4.2,state_5,state_6.1,
state_6.2,state_7,state_8,state_9,E_state;

initialize
to state_1
begin
end;

trans
from state_1
to state_2
begin
  MAX_TPDU_CR :=128
end;

trans
from state_2
to state_3
begin
 {form NDATAreq and CR_PDU according to constraints NDreq1 and CR33 }
 output L.NDATAreq;
 output T.Start(T1, WAIT_FOR_CC)
 { To take care of the ELAPSE statement in the successive events }
end;

trans
from state_3
to state_4.1
 when L.NDATAInd
  provided {decoding requirements --could be a primitive procedure call}
 begin
  R_TPDU_SIZE :=UserData.TPDU_size;
  DESTIN_REF :=UserData.SRC_REF;
  output T.Cancel(T1)
 end;

trans
from state_3
to state_4.2
 when T.Timeout(T1)
 begin
 end;

trans
from state_4.1
to state_5
```

```
begin
 { form NDATAreq and AK PDU according to NDReq_1, AK34 }
 output L.NDATAreq
end;

{ Test Step SIZE expands to two Pseudo events
  [R_TPDU_SIZE =MAX_TPDU_CR] (result :=pass)
        and
  [NOT[[R_TPDU_SIZE =MAX_TPDU_CR]] (result :=fail)
}

trans
from state_5
to state_6.1
 provided R_TPDU_SIZE =MAX_TPDU_CR
 begin
  result :=pass
 end

trans
from state_5
to state_6.2
 provided not (R_TPDU_SIZE =MAX_TPDU_CR)
 begin
  result:=fail
 end;

trans
from state_6.1
to state_7
 provided result =pass
 begin
  PR :=Pass
 end;

trans
from state_6.2
to E_state
 provided result =fail
 begin
  R :=FAIL
  { Disconnect Transport and Network connections }
 end;

trans
from state_7
to state_8
begin
 MAX_TPDU_CR :=(MAX_TPDU_CR * 2)
 { disconnect Transport connection }
end

trans
```

```
from state_8
to state_9
 provided not (MAX_TPDU_CR =8192)
 begin
 end;

trans
from state_9
to state_2
begin
 {This is for the GOTO statement if the MAX_TPDU_SIZE is less than 8192}
end;

trans
from state_4.2
to state_2
begin
 {This is for the GOTO statement after ELAPSE statement }
end;
```

CHAPTER 8 CONCLUSIONS

8.1 Conclusions

In this thesis, a test notation for protocol test suites called the Tree and Tabular Combined Notation has been described and an editor for defining and managing test suites specified in the TTCN is introduced. It has been seen that the editor can be used as a stand alone system or, with some extensions, may be used in conjunction with a test sequence generation tool called CONTEST-ESTL.

The importance of an editor in the test design process cannot be overstated. Typical test suites are generally very large and management is a common problem faced by suite designers. Syntax errors are inevitable when the tests are specified so the syntax checking feature of the tool is invaluable. Since the tables of TTCN form the basic structure of the language, automatic creation of these proformas is one of the strong points of the tool.

Experience with the tool include designing tests for Transport Class 4 using the Distributed Test Architecture [23] and for specifying some FTAM Test Cases [24]. It is expected that as the use of the tool increases, more features may be added to make the editor more functional and useful to the test designer.

During the system design process, a few points worthy of mention have been noticed by the author. As has been seen, the MP form of TTCN is very wordy and a typical test suite would require large amounts of storage space. This presents a few problems when it is required to load and compile a test suite written in TTCN-MP to convert to TTCN-GR or for further processing. For large suites, it may be required to break up the test suite into smaller suites, each containing the declarations part and constraints part and maybe a subgroup of the test case tree. Test Cases are independent of each other except that some may assume that a previous test case has determined that some required behaviour is present in the IUT. The results and parameters may be stored for to be used as parameters for the following tests. Therefore, if the ordering of the test cases is preserved, partitioning of the suite should present no problem.

8.2 Future Work and Improvements

The editor has been designed to be very modular to allow future extensions and modifications to the standard be incorporated with a minimum of effort. The file structures present a uniform interface for a variety of modules to interact with and modify the information independently. This does not make for very efficient storage and file naming requirements. Streamlining of the internal file structures may be necessary for the tool. The issue of portability to different systems makes the use of the X-Window environment a desirable extension. Fortunately, the file structure need not be modified to accommodate portability requirements.

Extensions to the tool in the direction of execution of test suites is necessary to round off all the requirements of a complete editor/executor of the TTCN language. To this end, implementation of the algorithm given in chapter 7 for the conversion to ESTELLE modules may be a feasible method. Some more research is required in this and related areas.

REFERENCES

[1] L. G. Roberts and B. D. Wessler, "Computer network development to achieve resource sharing," in *Proc. SJCC,* pp. 543-549, 1970.

[2] L. Pouzin, "Presentation and major design aspects of the CYCLADES computer network," in *Proc. 3rd ACM-IEEE Comm. Symp.*, (Tampa, FL, Nov. 1973), pp. 80-87.

[3] ISO, "Basic reference model for Open Systems Interconnection," *ISO 7498,* 1983.

[4] ISO, "OSI conformance Testing Methodology and Framework," Parts 1 to 3, *ISO 9646,* Sep 88.

[5] R.J. Linn, "An Overview of Conformance Testing Methodology," to appear in *IEEE JSAC.*

[6] "Recommendation X.25," *CCITT Study Group VII,* Working Paper 8, pp 100-190, fasc. VIII.2, Sept. 1981.

[7] ISO, "X.25 DTE Packet Level Conformance Test Suite," Canadian Contribution for *DP 8802-3,* Sept. 87.

[8] "User Guide for the NBS Prototype Compiler for Estelle," *Report No. ICST/SNA - 87/3,* October 87.

[9] ISO, "ESTELLE- A formal description technique bases on an extended finite state transition model," *ISO DIS 9074,* July 87.

[10] ISO, "LOTOS, A Formal Description technique Based on the Temporal Ordering of Observational Behaviour," *DIS 8807,* August 87.

[11] R. Milner, "A Calculus of Communicating Systems," Springer Verlag, LNCS 92, 1980.

[12] H. Ehrig, W, Fey, H.Hansen, "ACT ONE : An algebraic specification language with two levels of semantics," *Bericht-Nr.* 83-03, Inst. für Softw. Und. Theor. Inf., Tech. Univ. Berlin 83.

[13] E. Brinksma, "LOTOS", Tutorial Notes, 7th IFIP Symposium on Protocols, Zurich, May 87.

[14] ISO, "Profile of Abstract Syntax Notation One", and "Abstract Syntax Notation One Encoding Rules," *IS 8824, 8825,* 1987.

[15] Thomas Reps, Tim Teitelbaum, "Language Processing in Program Editors," *Computer,* IEEE, November 87.

[16] Takao Tenma, Hideaki Tsubotani, Minoru Tanaka and Tadao Ichikawa, "A System for Generating Language Oriented Editors," *IEEE Transactions on Software Engineering,* Vol. 14, No. 8, Aug 88.

[17] Anthony Wiles, "ITEX - An Interactive TTCN Editor and Executer," *SICS Technical Report, 1988* SICS , Box 1263, S-163 13 SPANGA, SWEDEN.

[18] ISO, "File Transfer, Access and Management," *ISO DIS 8571,* 1987.

[19] ISO, "Transport Protocol Specification," and "Transport Service Specification," *IS 8073,* and *IS 8072,* June 82.

[20] M. Barbeau, B. Sarikaya, "A Computer Aided Design Tool for Protocol Testing," *Proc. of INFOCOM '88,* pp. 86-95, IEEE, March 88.

[21] B. Sarikaya, "Test Design for Computer Network Protocols," Ph.D Thesis, McGill University, May 84.

[22] B. Sarikaya, Qiang Gao, "Translation of Test Specifications in TTCN to LOTOS," *Proc. of the Symposium on Protocol Testing, Verification and Validation,* Atlantic City, June 88.

[23] Sanjeev Farwaha, "Conformance Testing of Class 4 Transport Protocol in TTCN,"

Technical Report, Concordia University, Oct 88.

[24] M. Barbeau, B. Sarikaya, S. Eswara, V. Koukoulidis, "FTAM Test Design Using an Automated test Tool," *Proc. of INFOCOM '89,* April 1989.

APPENDIX

CONTEST-TTCN - USER'S MANUAL

SECTION 1 INTRODUCTION

ISO has recommended the Tree and Tabular Combined Notation (TTCN) as a preferred test notation. TTCN is an informal notation without precisely defined semantics. The defined purpose of TTCN is

a) to provide a common reference for assessing other test notations and to assist in examining the problems arising in test case and test suite design;

b) to provide a basis for the translation of test cases into other test notations.

A test suite written in TTCN has the four following sections in order:

a) Suite overview

b) Declarations

c) Dynamic part

d) Constraints part

TTCN is provided in two formats: TTCN-GR and TTCN-MP. TTCN-GR is the graphical form suitable for human readability and TTCN-MP is the machine processable form more suitable for transmission of TTCN descriptions between machines and possibly for automated processing.

This editor was designed to help the test designer enter tests in standard TTCN-GR notation and to automatically get the MP form of the specification. It is assumed that the tool user has some knowledge of the suntools environment and of the packages contained in it. Of course, a thorough knowledge of the "syntax" of TTCN-GR is assumed. The user would be well advised to have the BNF description of TTCN-MP handy when entering suites.

SECTION 2 INSTALLATION INSTRUCTIONS

CONTEST-TTCN software runs under the SUN-UNIX (4.2 release 3.3 and release 4.0) operating system. It uses the SUNVIEW package and the programs are written in C. A SUN graphics workstation is essential and the user must be in the SUNVIEW environment to invoke the editor.

- **FILES**

The CONTEST-TTCN distribution tape contains all the source files in the directory CONTEST-TTCN.

- **Compiling**

After having copied the distribution tape to your local file system, the first step is to change the working directory to CONTEST-TTCN and type make to compile the editor program and make gr2mp to create the MP2GR program.

SECTION 3 USING THE EDITOR.

The name of the TTCN-GR editor is ttcn. The editor is invoked by typing ttcn and <return> The user is required to enter the name of the test suite at this point. This is just an identifier whose BNF definition is given in the TTCN-MP document. There are seven items on the menu. These are selected using the left mouse button. The items are suite overview, declarations, main behaviour, default behaviour, constraints and quit. Each of the items are explained below. The editor, when used for the first time, creates a directory suite_files, in which are kept the data files of the GR version.

Suite Overview.

Selecting this item brings up the tables for the suite overview part of TTCN. The menu items are self explanatory for the most part. To enter fields in top panel of the table, move the arrow to that panel and just type in text. <return>takes the cursor to the next field. To enter the fields in bottom part of the table, move the mouse to that field. Remember that one line is taken as one field. A newline demarcates entries in the same field. All the four fields must be entered for each entry. A '_' can be used to represent a blank. **Remember to leave a blank line between the entries in each field,** especially for the 'purpose' field because this field is free text. 'Write and Return' saves the changes and returns to the main menu. 'Return' destroys any changes made and returns to the main menu. 'Proforma' brings up a pull right menu to enter either Test Cases, Steps or Defaults.

Declarations.

The declarations part of TTCN-GR is used to describe the set of test events and all other attributes to be used in the test suite. All objects used in the dynamic part should be declared in the declarations part except constraint declarations. The test events are ASP's (abstract service primitives) which occur at the PCO's (points of control and observation) and timer events. Other attributes to be specified appear on the menu when the 'Declarations' item is selected.

Abbreviations.

This table contains three fields. A newline demarcates each entry in the first two fields. A blank line is not necessary between entries. As before, there must be an expansion for every abbreviation entered.

Operators.

Selecting this item brings up a table consisting of a upper part with four entries and a lower part consisting of one field for a free text description of the operation. In the upper part, the first and fourth entries must be entered. The second and third entries are optional, but if entered, must be given as a list separated by commas. The menu items are self explanatory.

User Types.

This item brings up a menu for entering either TTCN User Types or ASN.1 User Types. The TTCN User Types table contains four fields in which the second field, Base Type, is optional for all entries. All other fields must be entered. The definition syntax must follow exactly that given in the TTCN-MP syntax.

Parameters.

This table contains three fields as shown and if there is an entry in the first field, there must be a corresponding entry in the second field.

Constants.

This table has four fields. The name, type and value of each global constant must be entered here. The syntax must be same as in the standard.

Variables.

This table has three fields and is used to declare global variables global to both the dynamic and constraints part. Initially, all variables are unbound.

PCO's.

This table lists the set of points of control and observation to be used in the test suite. Both fields must be entered for all entries and **a blank line must be left between entries,** especially in the ROLE field.

ASPs.

The entries for this table and for those of PDU's are the same. Remember that names are identifiers and as sucb, are composed of upper and lower case letters, numbers, hyphen ('-') and the underscore ('_') only. Names given in parentheses are full identifiers.

ASN-1 entries.

Tables for these definitions contain two fields. The first field is the identifier and the second is the name of a file containing the ASN-1 definition of the ASP or PDU.

Timers.

This table consists of three fields. The first is the timer name and second is timer duration. Examples are 1 ... 3 min, a to b sec, etc.

Dynamic Part

Main behaviour.

Selecting the *Dynamic Behaviour* button on the main menu brings up another menu. The second menu has three items, testcase, teststep and defaults. The structure of the suite using this editor will be organized as a tree with the suite name as root and all test cases as its children. When the menu is first called, the current test case number is 1.

In the bottom fields, the first one is the behaviour description. Here, the important thing to remember is that only tree names can start in column number 1. Comments can be entered using '/*' starting in column number 1. For GOTO statements, only the word GOTO (in upper or lower case) is acceptable. In order to find out the present line number, the left mouse button is first pressed at the line in question. Then, depressing the right button pops up a menu with the item "what line #. Selecting this item displays the present line number. This facilitates entering corresponding entries in each field. Some other menu items of interest in this subwindow are :

Suite-Tree : This item brings up a window which shows the structure of the suite as a tree with the suite identifier as the root with children representing groups and test cases. The user can select the test case of interest with the mouse and browse through the suite.

Goto : This item can be used to jump to a specified test case by giving the number or to create a new test case.

ASPs, PDUs, Abbreviations: These items bring up pull right menus containing all the ASPs, PDUs and Abbreviations which have been entered for that suite. The selected item is pasted at the current point of insertion. This eliminates spelling errors and proves handy when defining large test suites containing many ASPs, etc.

V-Hair: This item draws a vertical bar representing the mouse in the first column. This is useful to align events which are at a distance from each other.

Constraints

This section brings up a menu with all the possible constraint entries, one for ASPs and one foe PDUS. Since ASP constraints are similar to PDU constraints, only PDU constraints will be explained.

PDU Constraints :

There are three forms of PDU constraints and two generic constraint proformas. The first is the TTCN-PDU constraints. This table is displayed when PDU constraints is selected. The top panel has two entries, PDU name and constraint name. The bottom part has three fields. The first entry has the field name and the second, the value. The editor does not support the compact constraints table. So a different table has to be called up for each TTCN-PDU constraint.

The second form of constraints is the compact constraints where it is possible to use one proforma per PDU to define all the constraints on that PDU. Each constraint is entered on one line with values given for all the fields which are entered as column names. One field column is displayed at a time with menu items to go the left and right fields.

The third type of constraint is the ASN-1 description of constraints for PDU's and ASP's which have been defined using ASN.1.

Hard Copies

Every table created by the editor has an item in the frame menu called hard copy. Selecting this item automatically creates the necessary file as input to the formatter Troff for output to a laser printer. Device independent Troff (Ditroff) may be used to make the output files independent of the printing device.

Creating MP form

The program gr2mp tries to create the MP version of the suite entered. The program is called by giving the suite identifier as the argument. The error messages, if any come up, are self explanatory mostly. If there is an entry missing for a field for which there is an entry in the first field, the error message would be to say that there is no entry for the last entry in the first field. This can be avoided by entering '_' for any field for which there is no entry. This is because, unless stated otherwise explicitly in the BNF description, every field must have an entry if there is an entry in the first field (except comments, of course). This does not hold for the fields in dynamic behaviour and ASN-1 constraints and for the base type field in User Type definitions. The MP version, if successful is stored in the file suitename.MP where suitename is the identifier entered in the beginning.

Intermediary Files

The following is the list of intermediate files created by the editor as a data base. The '*' stand for digits which represent the ordering of the entry in its type of pro-forma.

suite overview	=tname_a.1 tname_a.c.[2-5] tname_a.s.[2-5] tname_a.d.[2-5]
Declarations	
abbreviations	=tname_b.[1-3]
operators	=tname_c.*.[1-2]
user_types	=tname_d.[1-4]
ASN.1 user types	=tname_i.*.[1-2]
parameters	=tname_e.[1-4]
constants	=tname_f.[1-4]
PCOs	=tname_g.[1-2]
ASPs	=tname_h.*.[1-4]
ASN-1 ASPs	=tname_j.*.[1-2]
PDUs	=tname_k.*.[1-4]
ASN-1 PDUs	=tname_l.*.[1-2]
global vars	=tname_m.[1-4]

```
timers        =tname_n.[1-3]
```

Dynamic Behaviour

```
test cases     =tname_o.*
test steps     =tname_O.*
default beh     =tname_p.*
```

Constraints

```
PDU constraints =tname_q.*.[1-4]
Compact PDU con =tname_z.*.[<34]
ASN-1 PDU const =tname_r.*.[1-2]
generic PDU con =tname_t.*.[<34]
gen. pdu fields =tname_s.*.[<34]

ASP constraints =tname_u.*.[1-4]
Compact ASP con =tname_A.*.[<34]
ASN-1 ASP const =tname_v.*.[1-2]
generic ASP con =tname_w.*.[<34]
gen. asp fields =tname_x.*.[<34]
```

Conversion from MP to GR

The program mp2gr attempts to create the necessary intermediate files from a given MP file for viewing and editing by CONTEST-TTCN. This program is invoked by giving the MP file as the argument to the command.