NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ISBN 0-315-56362-1

Canada

High-Level Analysis of Test Results For Open Systems

By

Shahram Akhavan-Sarraf

Thesis submitted to
The School of Graduate Studies and Research
in partial fulfillment of the requirements for the
degree of Master of Computer Science

University of Ottawa
Ottawa, Ontario, Canada.
1988

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Abstract

Results obtained from testing protocol implementations usually require enormous amount of analysis. The degree of conformance of results to the protocol standards should be clearly identified in a standardized test report, in order to have achieved the primary objective of conformance testing. A standardized conformance test report has the benefit of condensing all the output results into a manageable, concise and commonly understandable format. In addition, advanced analysis tools are required to evaluate the results and identify problems in execution, incomplete or incorrect specification of test cases, or even errors in formal specifications of protocol standards. Such automated tools should uncover errors much faster than manual results analysis and should greatly improve the cost-effectiveness of protocol testing.

In this thesis, we propose a notation and approach for high-level reporting and analysis of conformance test results in order to facilitate the comparability and auditability of conformance testing activities. We list some requirements on the syntax and content of Executed Test Result reports (ETR's) and on Conformance Test Suites (CTS's) in order to be able to derive Abstract Test Result reports (ATR's). A TTCN-based notation denoted TTCN-TR is presented for ATR's, and a method is given for deriving an ATR from an ETR. This derivation method also involves the verification of verdicts of non-conformance of the Implementation Under Test (IUT) with the CTS. ATR's are shown to be sufficient for representing the traces of normal form specifications, and thus, provide a general means of directly verifying instances of non-conforming IUT behaviour against a formalized protocol specification. Some observations and recommendations are presented related to the issues of comparability and repeatability of test results. Finally, semi-automated methods and techniques are presented for error detection and for characterizing the interdependency of tests.

Acknowledgements

I would like to express my deep sense of gratitude to my thesis supervisor, Professor Robert L. Probert, for his advice, guidance and constant encouragements. I am proud of the opportunity I was given to work with him. I shall remember the valuable experiences I have gained during the preparation of this thesis through out my career.

I wish to thank Dr. Hasan Ural and Charles Truscott, together with all other members of the Protocol Research Group for their useful comments and the stimulating discussions. It has been a pleasure to be a part of the PRG group. I thankfully acknowledge the valuable information and advice of Mr. Marc W. Hornbeek (Manager at Bell-Northern Research) and his group during the course of this work.

I also thank the National Science and Engineering Research Council of Canada (NSERC) and Bell-Northern Research (BNR) for their financial support through the TOCS project.

Finally, my special thanks goes to my parents, for encouraging me through out the years of my studies, and supporting me in achieving my goals in life.

Table of Contents

List of Figures

Glossary of Conformance Testing Terminology

The Conformance Testing terminology given in this section is based on the ISO standard terminology [ISO9646-1,5]. In addition, other terms are presented in order to assist the understanding of this thesis. Terminology which is different from ISO standard terminology is indicated by '*'.

1. Conformance testing terminology:

1.1 Implementation Under Test (IUT) :

The part of the real open system which is to be assessed by testing. This system should be an implementation of one or more OSI protocols in an adjacent user/provider relationship (OSI stack).

1.2 *System Under Test (SUT) :

The real open system which consists of the IUT and operational support software.

1.3 Dynamic Conformance Requirements :

All those requirements (including options) which determine what observable behaviour is required by the relevant OSI standard(s) in instances of communication.

1.4 Static Conformance Requirements :

Constraints in OSI standards which facilitate interworking by defining requirements for the capabilities of an implementation.

1.5 Protocol Implementation Conformance Statement (PICS) :

A statement made by the supplier of an OSI implementation or system, stating the capabilities and options which have been implemented, any features which have been omitted, and selected values of critical parameters.

1.6 **Protocol Implementation eXtra Information for Testing (PIXIT) :**

A statement made by a supplier or implementor of an IUT which contains or references all of the information (in addition to that given in the PICS) related to the IUT and its testing environment, which will enable the test laboratory to run the appropriate test suite against the IUT.

1.7 **Basic Interconnection Testing (BIT) :**

Limited testing of an IUT to determine whether or not there is sufficient conformance to the main features of the relevant protocol(s) for interconnection to be possible, without trying to perform thorough testing.

1.8 **Capability Testing :**

Testing to determine the capabilities of an IUT with regards to static conformance requirements and PICS.

1.9 **Static Conformance Review :**

A review of the extent to which the static conformance requirements are met by the IUT, by comparing the static conformance requirements expressed in the relevant standard(s) with the PICS and the results of any associated capability testing.

1.10 **Behaviour Testing :**

Testing the extent to which the dynamic conformance requirements are met by the IUT.

1.11 ***Conformance Assessment Process :**

The complete process of accomplishing all conformance testing activities necessary to assess the conformance of an implementation or a system to one or more OSI test suite standards to be assessed. It includes the production of the PICS and PIXIT documents, preparation of the real tester and setting up of the IUT, the execution of one or more test suites, conversion of executable test result reports to abstract test result reports, and using the abstract test result reports for further analysis (the latter two are steps in Analysis of Test Results).

1.12 ***Test Purpose :**

A description of the objective which an abstract test case is designed to achieve.

1.13 *Test Event :

An indivisible unit of test activity at the level of abstraction of the specification (e.g. sending or receiving a single PDU).

1.14 *Test Step :

A named subdivision of a test case, constructed from test events and/or other test steps, and used to modularize abstract test cases.

1.15 Preamble (of a test case) :

The test steps needed to define the path from the starting stable state of the test case up to the initial state from which the test body will start.

1.16 Test Body (of a test case) :

The set of test steps that are essential in order to achieve the test purpose and assign verdicts to the possible outcomes.

1.17 Postamble (of a test case) :

The test steps needed to verify arrival at the target state and also, a null response if required.

1.18 *Generic Test Case :

A specification of the actions required to achieve a specific test purpose, defined independent of a particular test method.

1.19 Abstract Test Case :

A complete and independent specification of the actions required to achieve a specific test purpose, defined at the level of abstraction of a particular abstract test method. It includes a preamble and a postamble to ensure starting and ending in a stable state.

1.20 Executable Test Case :

A form of the abstract test case which can be used in execution for testing the implementation under test.

1.21 **Generic Test Suite (GTS) :**

A generic test suite is composed of generic test cases, which has the coverage of the the complete set of test purposes for the particular protocol. The test purposes defined in a generic test suite are identical or a superset of the test purposes defined in any particular abstract test suite for the same protocol.

1.22 **Abstract Test Suite (ATS) :**

A complete set of abstract test cases normally specified as an international standard, possibly combined into nested test groups, in order to carry out the conformance testing.

1.23 **Executable Test Suite (ETS) :**

A complete set of executable test cases, possibly combined into nested test groups, in order to carry out conformance testing.

1.24 **Abstract Test Method (ATM) :**

The description of how an IUT is to be tested, given at an appropriate level of abstraction to make the description independent of any particular implementation of testing tools, but with enough detail to enable tests to be specified for this method. A testing architecture together with an abstract test suite construct an abstract test method. The standard ATM's are described in [ISO9646-1,2]. For testing IUT's one of this set of test methods is used.

1.25 **Selected Abstract Test Suite (SATS) :**

The subset of an abstract test suite selected using a specific PICS. Using SATS a selected set of executable tests can be identified as selected executable test suite (SETS).

1.26 **Parameterized Executable Test Suite (PETS) :**

A selected executable test suite in which all test cases have been made parameterized executable test cases based on the appropriate PICS and PIXIT.

1.27 **Test Campaign :**

The process of executing the parameterized executable test suite for a particular IUT and producing the conformance log.

1.28 **Test Operator :**

The person or persons designated by the test laboratory as being responsible for running conformance tests against the IUT.

2. Test results terminology:

2.1 ***Outcome :**

A sequence of test events together with the associated input/output, either identified in an abstract test case specification (potential outcome), or observed during test execution (actual outcome).

2.2 ***Foreseen Outcome :**

An outcome clearly identified or categorized in the abstract test case specification.

2.3 ***Unforeseen Outcome :**

An outcome not specifically identified or categorized in the abstract test case specification. It is usually captured by ?OTHERWISE statement in TTCN.

2.4 ***Foreseen Invalid Test Event :**

This is equivalent to "syntactically invalid test event" [ISO9646-1]. This is a single test event which violates syntactic requirements of protocol standards. Such an event may be purposely inserted into the test suite to test conformance to error detection and recovery requirements, for example.

2.5 ***Inopportune Test Event :**

This is a test event which, although syntactically correct, occurs or arrives at a point an observed outcome when not allowed to do so by the protocol standard. (It is also usually categorized under the ?OTHERWISE statement in TTCN).

2.6 **Test Results Reporting :**

The process of presenting the log information obtained from executing the parameterized executable test suite. Test results reporting includes generating a summary report. Test results reports are of two kinds:

- o Executed test results reports (ETR's).

o Abstract test results reports (ATR's).

2.7 **System Conformance Test Report (SCTR) :**

A summary report generated at the end of the conformance assessment process, giving the overall summary of the conformance of a multi layer protocol system under test.

2.8 ***Protocol Conformance Test Report (PCTR) :**

The ATR equivalent of the summary report generated at the end of the test session as part of the ETR, providing the summary of the test results.

2.9 ***Executed Test Results reports (ETR's) :**

These reports contain outcomes which are produced when testing an IUT after execution of actual test cases in the test suite. Executed test results reports follow a local format.

2.10 ***Abstract Test Results reports (ATR's) :**

A version of the executed test result reports, represented in a more understandable, higher level, abstract, TTCN-like manner. In addition, ATR's contain summary reports such as PCTR's as well as abstract conformance logs.

2.11 ***Executed Conformance Log (ECL) :**

A record of the actual sequence of observed events as a result of conformance testing of the IUT. The ECL is useful for manually verifying verdict assignments.

2.12 ***Abstract Conformance Log (ACL) :**

The version of the executed conformance log, represented in a more understandable, higher level, abstract, TTCN-like manner. The ACL is intended to promote comparability between ATR's, and to assist in automating verdict verification and explanation.

2.13 ***'Pass' Verdict :**

A verdict given when the observed outcome matches the expected outcome specified in the test body, and when preamble and postamble conditions are satisfied.

2.14 **'Fail' Verdict :**

A verdict given when the observed outcome ends with a foreseen invalid or inopportune test event with respect to the relevant protocol standard(s) or the PICS.

2.15 **'Inconclusive' Verdict :**

A verdict given when the observed outcome is valid with respect to the relevant standard(s), but prevents the test purpose from being accomplished.

2.16 ***Derivation (of results) :**

The process of converting the ETR's to ATR's using the abstract test suite as a reference.

2.17 ***Repeatability (of results) :**

Results are repeatable if results obtained from execution of each test case in the test suite on a particular IUT are the same whenever it is performed.

2.18 ***Comparability (of results) :**

Results are comparable if results obtained from execution of each test case in the test suite on a particular IUT are the same using different test environments.

2.19 ***Auditability (of results) :**

Results are auditable if they can undergo a diagnostic procedure in order to identify that all the execution procedures have been followed correctly.

2.20 ***Cross Reference Table (CRT):**

A table containing all cross references between global modules in the test suite (referenced in test results) and test cases. The CRT's are useful in test results analysis, particularly in identifying common locations of instances of non-conformance in the ATR's. They are also a useful way of maintaining the test history.

2.21 ***Test Dependency Graphs (TDG):**

A set of graphs used to show the dependency relationships between the test results. These relationships are identified on the basis of the commonality of the preamble, test body and/or postamble test modules referenced in the test results. The idea may

also be applied to the test suite, in which case similar graphs may be created to show the dependency relationships between the test cases.

2.22 *Test Dependency Tree (TDT):

A tree used to indicate how a test result is related to other test results. These are used to facilitate the diagnosis of test results. The idea may be applied to the test suite, in which case similar dependency trees may be created to show how a test case is related to other test cases. As well, the later information is useful for interactively managing test suite execution.

2.23 *Analysis of Test Results :

Analysis of test results includes one or more of these steps:

- o Conversion of executable test results reports to abstract test results reports using the standard test suite (assumes validation of abstract test suite).
- o Verifying repeatability of test results.
- o Checking comparability of test results from different test sites.
- o Using abstract results for cross validating the protocol specification or ability to relate the test results to the test purposes.
- o Checking auditability of test results.
- o Diagnosing erroneous behaviour as likely due to one of IUT, test system or test suite.

2.24 *Exception ATR :

An ATR which contains a record of only those test cases which were executed and failed.

2.25 *Detailed ATR :

An ATR which contains a record of all abstract test results.

2.26 *Summary Test Report :

A test report (included in the SCTR) which records only the numbers of passed, failed and inconclusive executed test cases instead of the entire test campaign report.

2.27 **Test Analyst :**

The person or persons designated by the test laboratory as being responsible for providing observations on the test results as seen in the test reports.

Abbreviations

ACL	:	Abstract Conformance Log
ASCII	:	American Standard Code for Information Interchange
ASN-1	:	Abstract Syntax Notation - One
ASP	:	Abstract Service Primitive
ATM	:	Abstract Test Method
ATR	:	Abstract Test Result reports
ATS	:	Abstract Test Suite
CRT	:	Cross Reference Table
CTS	:	Conformance Test Suite
DCE	:	Data Circuit-terminating Equipment
DTE	:	Data Terminal Equipment
ECL	:	Executed Conformance Log
ETR	:	Executable Test Result reports
ETS	:	Executable Test Suite
FSM	:	Finite State Machine
GTS	:	Generic Test Suite
ISO	:	International Standardization Organization
IUT	:	Implementation Under Test
LAPB	:	Link Access Protocol - Balanced
LOTOS	:	Language Of Temporal Ordering Specifications
OSI	:	Open Systems Interconnection
PCO	:	Point of Control and Observation
PCTR	:	Protocol Conformance Test Report
PDU	:	Protocol Data Unit
PETS	:	Parameterized Executable Test Suite
PICS	:	Protocol Implementation Conformance Statement
PIXIT	:	Protocol Implementation eXtra Information for Testing
SAP	:	Service Access Point
SCR	:	Static Conformance Review

SCTR	:	System Conformance Test Report
SETS	:	Selected Executable Test Suite
TCP	:	Test Coordination Procedures
TDG	:	Test Dependency Graphs
TDT	:	Test Dependency Trees
TLAN	:	Test LANguage
TTCN	:	Tree and Tabular Combined Notation
TTCN/GR	:	Tree and Tabular Combined Notation (GRaphical form)
TTCN/MP	:	Tree and Tabular Combined Notation (Machine Processable)
TTCN-TR	:	Tree and Tabular Combined Notation for Test Results
TTCN-TR/GR	:	Tree and Tabular Combined Notation for Test Results (GRaphical form)
TTCN-TR/GR	:	Tree and Tabular Combined Notation for Test Results (Machine Processable)

Chapter 1

Introduction

1.1 Introduction to Conformance Testing and Test Result Analysis

Conformance Testing may be referred to as a facet of software testing, mostly related to functional testing. **Functional software testing** is a methodology that considers the object to be tested as a **black-box**, without the knowledge of the internal structure. The behaviour of this black-box through an interface, can be observed and influenced by its environment. In black-box testing the objective is to find out solely when the input and output behaviour of a program does not agree with its specification. In this approach, test data for an implementation are constructed from its specification alone [Demil87]. The advantage of black-box testing is that the test data can be generated semi-automatically, using the specification, although commercial tools for this purpose have not yet been developed. The major drawbacks of black-box testing are its dependence on the

specification correctness (not usually possible in real life), and the necessity of using most possible inputs in test cases in order to be assured of module correctness.

Testing in general is the controlled execution of an **implementation under test (IUT)** for the purpose of uncovering errors. Testing can not demonstrate the absence of errors in a given implementation, only their presence [Myer79]. It is commonly known that exhaustive testing is impractical, due to the fact that software can have many states and the number of testcases required to test each state increases exponentially with the number of states. Because of this intriguing problem, normally the tester must severely limit the number of testcases. However, then it is necessary to judge whether a sufficient number has been chosen to provide meaningful test coverage. The issue of test completeness is extremely difficult. In a nutshell, the economics of the testing are that a testing service clearly has a cost associated with it; moreover there are an unlimited number of tests which may be applied to the IUT to detect errors. It is the ideal balance between cost and test completeness that we are seeking, bearing in mind of course our policy with respect to obtaining a reasonable level of confidence in the IUT with regards to the specification.

In recent years, **conformance testing** has become recognized as a critical component in the engineering of Open Communication Systems. It is realized that the objective of OSI will not be completely achieved until systems can be tested to determine whether they conform to the relevant OSI protocol standard(s). Conformance testing is a means of demonstrating whether or not a protocol implementation under test conforms to its corresponding specification. The capabilities which are tested in conformance testing are only those stated by the implementor. The question of robustness or reliability of the IUT is not necessarily considered. The purpose of conformance testing is to increase the probability of interworking between implementations.

The conformance of a system under test is decided upon analyzing the produced test results from the test session. **Test results analysis** is the process of assessing degree of conformance of the product to its requirements (e.g. protocol entity to protocol standards), based on conformance testing. Using this process, a verification report of the test results can be obtained and subsequently analyzed to meet the primary objectives of conformance testing, namely an assessment of the degree of conformance of the system under test to the requirements (e.g. standards). In this thesis we focus our attention on issues related to analysis of test results. In some applications, checking test results for a test session can take several weeks and might not be very accurate if manually performed. The cost of analysis may increase unacceptably if test sessions are repeated several times.

In the past, the notion of automating test results analysis has attracted great interest [Panzl78]. The automation can be of a very simple nature, such as matching the observed outputs with the expected outputs to arrive at a verdict, or may address more complex problems such as diagnosis of test results. In the latter case, full automation may not be a possibility, in which case tools may be developed to aid in solving parts of the problem, leaving the other parts to experts (i.e. semi-automatic test results analysis). One of the keys to achieving our goal of automating some aspects of test results analysis is utilization of the available conformance testing standards. This is discussed in the next section.

1.2 Conformance Testing Standards

The International Organization for Standardization (ISO) has directed much of its attention towards standardization of conformance testing, in the hope of establishing some requirements for conformance testing to be followed by each test center and its clients. In addition, standards provide additional procedures and recommendations in order to increase the quality of conformance testing. The ISO project concerned is ISO IEC JTC1 SC21 project 23 which progresses in parallel with Q47 of CCITT subgroup VII. The relevant ISO document which this group is producing is called 'Conformance Testing Methodology and Framework' [ISO9646-1,5]. It consists of five parts and is currently at 'Draft Proposal' stage. The author of this thesis along with other members of the protocols research group, University of Ottawa, has made a number of contributions towards development of this document. Some of the contributions supported the development of a precise, easy to learn test specification language called TTCN (Tree and Tabular Combined Notation). This notation has been adopted by a number of the protocol defining groups as the required language for specifying an international test suite for their specific protocols. As one example, subcommittee 6 (ISO/IEC JTC1/SC6) has selected test suites in TTCN proposed by Canada for data link layer and packet layer conformance testing draft standards.

There are several steps involved in a conformance testing process which are described briefly as follows [ISO9646-1,2, Prob88a]. We will assume the reader is familiar with the ISO document DP9646 in the remainder of this thesis:

1) **Test sequence derivation and adaptation**: the process of establishing a generic conformance test suite (CTS) for a given protocol. Further, adapting the generic CTS for a particular test architecture and obtaining an abstract CTS.

2) **Test sequence selection**: determination of an appropriate subset of the abstract test suite to be applied to the IUT. Selection will be on the basis of the claims of the protocol implementor and perhaps efficiency concerns [Matth86]. This activity can be performed either on-line (during test session) or off-line (prior to test session). Since test planning is seen as essential to effective testing, the latter is preferred.
3) **Test sequence execution**: the process of applying an executable CTS to the IUT. This involves configuration of the test environment for running the test (test set-up) and obtaining test results.
4) **Analysis and evaluation of test results**: the process of checking and reporting test results, verifying repeatability and comparability of test results, and producing possible diagnostic reports. This activity can also be performed either on-line or off-line. Due to the volume of tests, off-line analysis is preferred.

A global picture of conformance testing for two different open systems is shown in Figure 1.1.

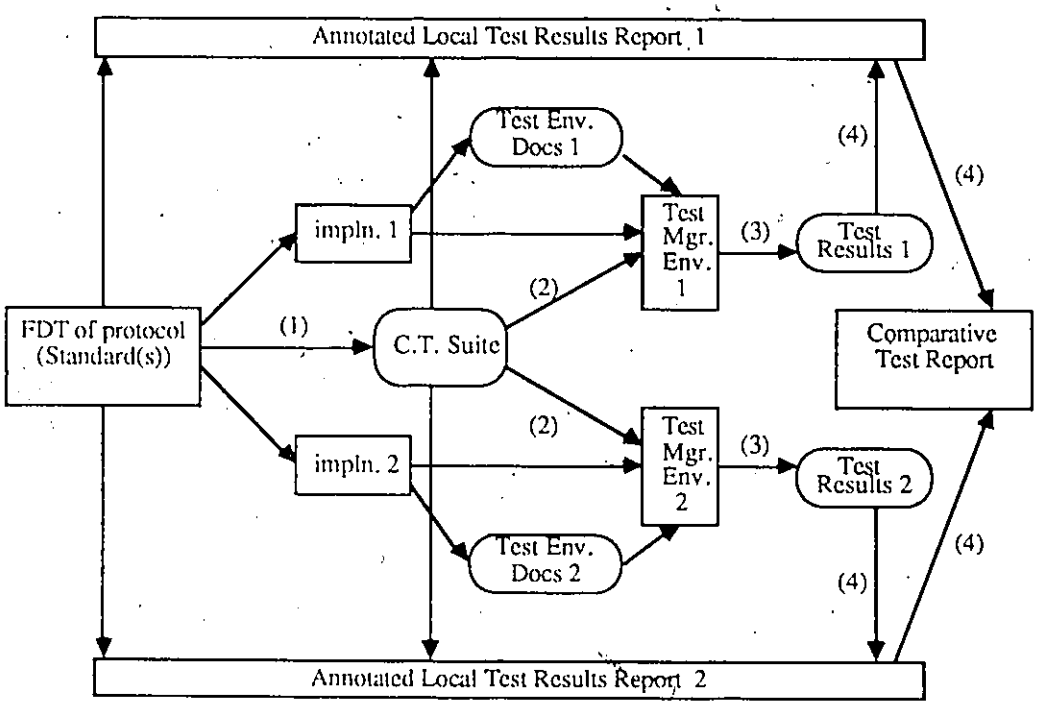


Figure 1.1 - Conformance Testing of Open Systems

The following sections discuss each of these steps in more detail.

1.2.1 Test Sequence Derivation and Adaptation

Test sequence derivation starts with creation of a set of test purposes, with each test purpose focussed on a single conformance requirement of the protocol. A **generic test suite** is then constructed consisting of a **generic test case** for each test purpose. A generic test suite includes all possible abstract test suites for the relevant protocol(s). The generic test suites are updated in two ways to produce an abstract test suite (ATS):

1) If not already present, preamble and postambles are added to the test body. The preamble starts from a chosen state, usually the idle state, and leads to the required initial state of the test body. The postamble starts from the end of the test body and returns to a chosen stable state. The test body corresponds to a specific test purpose.
2) All the extra information for the particular test method is added.

A testing architecture together with an abstract test suite constitute an abstract test method (ATM). Points of control and observation (PCO's) are identified in the testing architecture and are referred to during the abstract conformance test design. Currently the ISO conformance testing group has defined five different test architectures [ISO9646-1,2]. Testing can be performed for Single layers (S), Multiple layers (M), or Embedded single layers (E). These test architectures are described briefly as follows:

In a **Local Test Architecture** (L), the IUT is stimulated directly by ASP's (Abstract Service Primitives) at the points of control and observation (PCO's) above and below the layer (See Figure 1.2-a). Due to the fact that both service interfaces are accessible the local test architecture has a fairly good **error detection capability** (errors are detected when a given behaviour is not in conformance to the specification. Error detection capability of protocol testers depends on test architecture, test suite and the design of the test processes which are used to stimulate/observe the sending/receiving events).

In the **Distributed Test Architecture** (D), the (N)-ASP's are directly controlled and observed with a testing entity called the upper tester (Figure 1.2-b). The upper tester is at the upper service boundary and the lower tester is at the opposite side (N-1) SAP (Service Access Point). In this architecture there is a need for coordinating two testing entities, the upper and lower testers. This is done using a TCP (Test Coordination Procedure). In this method, points of control and observation (PCO's) are distributed over two different locations. The possibility of error detection in this method is less than in the local test architecture.

Coordinated Test Architecture (C), is an enhanced version of the distributed architecture (Figure 1.2-c). In this method, test coordination procedures are implemented as a test management protocol. There is a single point of observation and control at the opposite side of the entity under test, in between the lower tester and (N-1) service provider. In this case the interface between the upper tester and IUT is invisible, therefore the cooperation of the manufacturer of the IUT is usually needed in implementing and installing the upper tester.

In the **Remote Test Architecture (R)**, the controlling and observing of (N-1)-ASP's from the IUT are done using a PCO at the opposite side from the entity under test, beneath the lower tester and (N-1) service provider (at the point of interface between the lower tester and the underlying service provider) (Figure 1.2-d). This method also does not provide us with adequate error detection capability, due to the fact that the upper boundary of the IUT is not controlled and observed at the point of interface with the upper tester. The TCP and upper tester are implementation dependent, and may in fact consist of operator messages transmitted to the IUT. This is indicated by the dashed lines in the figure.

Note: In Figure 1.2 where boxes are joined together, it can interpreted that a point of control and observation does not exist. Points of control and observation are clearly identified in the figure.

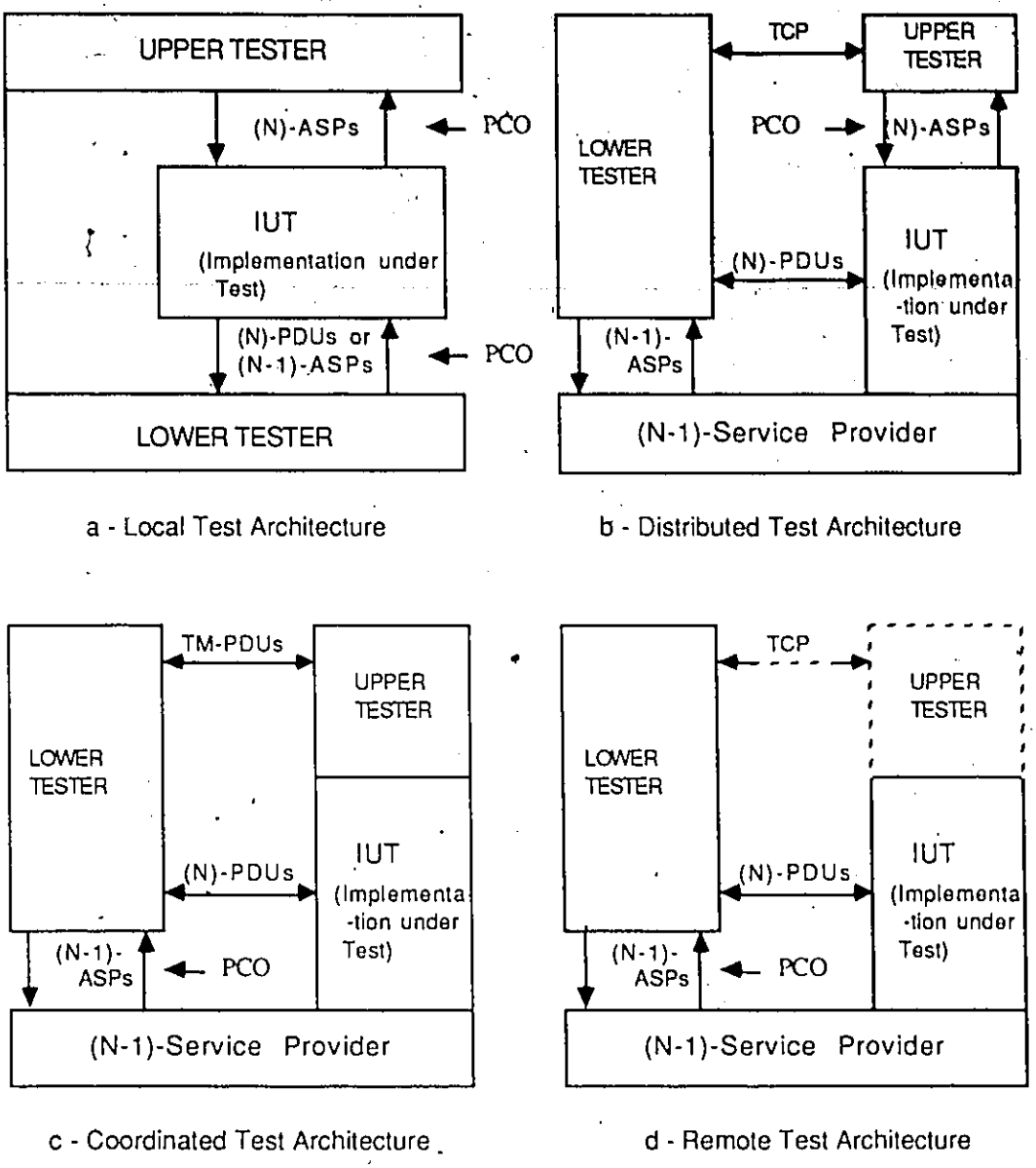
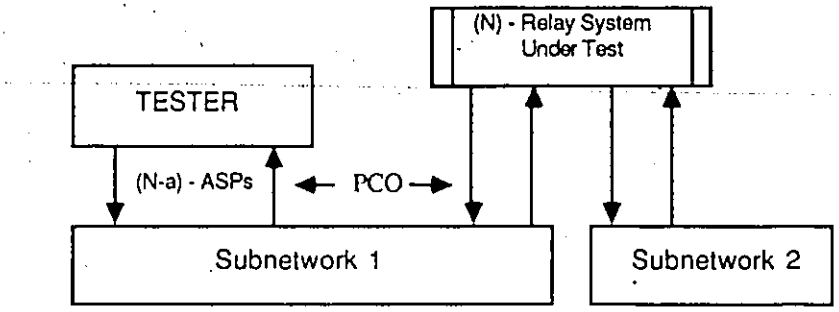


Figure 1.2 (a-d) - Testing Architectures

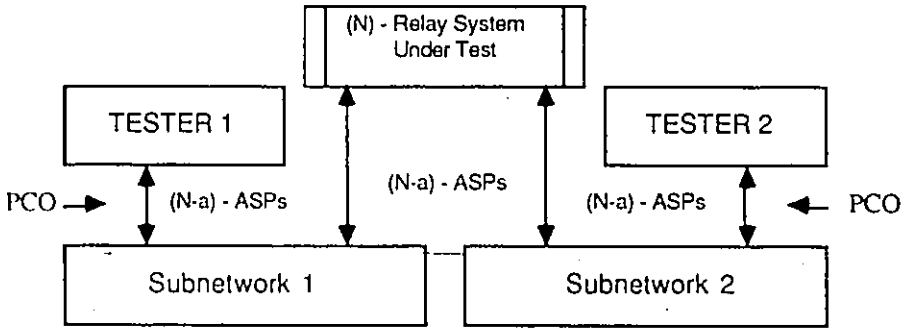
The **Relay Test Architecture (RL)**, is somewhat different from the previous architectures and is designed to test (N) layer relay systems (Figure 1.3-a,b). It involves a relay of messages through the subnetwork systems to the upper tester and vice versa. This can be done by looping back, or observing messages using the second subnetwork:

a) **Loop-back:** (Shown in Figure 1.3(a)) In this test method there are two points of control and observation in the same side of one subnetwork.

b) **Traverse:** (Shown in Figure 1.3(b)) In this test method the points of control and observations are on opposite sides of the service provider from the (N)-Relay, one on each subnetwork.



a - Loop-back Architecture



b - Traverse Architecture

Figure 1.3 - Relay Test Architectures

1.2.2 Test Sequence Selection

When a client requires his system to be tested, he should prepare a Protocol Implementation Conformance Statement (PICS), stating the capabilities and options which have been implemented, together with any features which have been omitted (See Appendix 'C'). This is needed so that the implementation can be tested for conformance against relevant requirements, and against only those requirements [ISO9646-1, Rayn87]. Before the test selection step the PICS is analyzed (Static Conformance Review) for its self-

consistency, and its consistency with the static conformance requirements specified in the relevant standard(s).

In order to test a protocol implementation the test laboratory also requires other information related to the IUT and its environment. This should be submitted by the manufacturer of the protocol. This document is referred to as the Protocol Implementation eXtra Information for Testing (PIXIT), and may contain information such as the following (See Appendix 'D'):

1) Information to help the test operator execute a particular test case (or test suite) on a specific system.
2) Representation of the information in the PICS in a broader fashion (e.g. particular values for parameters such as network addresses, identifiers, counters, timer values, encoding strategies, etc.)
3) Information to identify which capabilities stated in PICS are testable and which are not supported for testing purposes.
4) Other related information which may help to ease the operator's job at the test laboratory. References to various documents for more information is an example.

The PICS aids the test laboratory in selecting the appropriate test cases from the abstract test suite and therefore preparing the equivalent selected executable test suite (SETS). Then the information provided in the PIXIT is used to parameterize the test cases in SETS. The resulting parameterized executable test suite (PETS) is then ready to be executed.

1.2.3 Test Sequence Execution

This step involves execution of the parameterized executable test suite. The standard term is **test campaign**. The test campaign is comprised of executing basic interconnection tests, capability tests and behaviour tests. These conformance test types are described briefly as follows:

a) **Basic Interconnection Tests:** These tests are used to detect any severe cases of non-conformance (e.g. testing the capability of the IUT to see whether it can interconnect with the tester. Successful execution of these tests is a prerequisite to continuing the testing process. Thus, these tests are intended to verify that a connection can be established with the IUT, and PDU's can be transmitted correctly over the underlying transmission medium.

b) **Capability tests:** The purpose of these tests is to check that the observable capabilities of the IUT are in accordance with the static conformance requirements and the capabilities claimed by the client (as stated in PICS).

c) **Behaviour tests:** These tests provide comprehensive coverage with respect to the dynamic conformance requirements specified by the protocol standard. These tests should be confined to the capabilities of the IUT. **State transition testing** (State transition tests provide a thorough testing of an implementation to cover every possible transition of the IUT from one state into a new state as defined in the protocol standard) and **functional range testing** (Functional range tests are used to test the IUT over a full range of protocol parameters and timers) are two types of behaviour testing. More detailed categories of tests can be found in [Rayn87]. It should be noted that representative instances of communication are usually tested. The results obtained are called **dynamic conformance results.**

Note: Earlier we suggested that conformance testing refers to black box testing. However, it can be mentioned that grey-box testing is perhaps a more appropriate way of looking at behaviours [Prob82].

Figure 1.4 represents the complete conformance testing process [ISO9646-5].

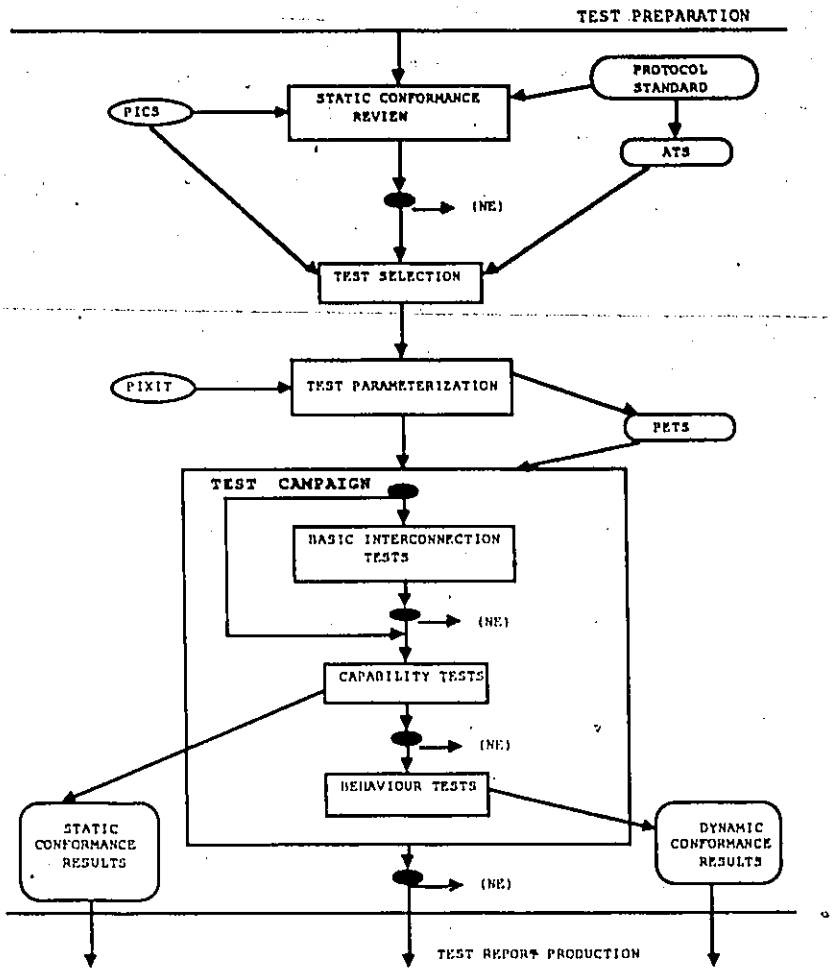


Figure 1.4 - Conformance Testing Process

1.2.4 Test Results Analysis and Evaluation

The final step in the conformance testing is analysis and assessment of test results to investigate instances of non-conformance. At this stage the observed behaviours are validated with respect to the corresponding test purposes and the test cases. Analysis should expose any non-conformance with respect to the conformance requirements. This step of conformance testing is the focus of this thesis.

1.3 Test Specification Notation - TTCN

A standard test notation is used to describe abstract test cases. TTCN (Tree and Tabular Combined Notation) is an informal notation with clearly but not formally defined semantics [ISO9646-2a, 3]. TTCN is available in two forms, a **graphical representation (TTCN/GR)** and **machine processable representation (TTCN/MP)**. The graphical representation is used to provide test cases in a human-readable tabular form and is used in standards documents [ISO8882-2, ISO8882-3]. The machine processable form is transferrable between different computer test laboratories. Some of the useful characteristics of TTCN are as follows:

1) TTCN has a semi-formal definition, which is precise, unambiguous and deterministic.
2) TTCN provides modularity by means of attachment of test trees. A hierarchial structuring of test cases can be achieved.
3) TTCN is applicable to single layer test methods (all layers) and can be also used for multi layer test specification.
4) A number of prototype tools are now available, such as TTCN editors and a TTCN syntax and integrity checker [Prob88a, Wiles86, ITEX88]. Other tools such as a translator for converting the abstract test suite to its executable equivalent are under development.
5) TTCN is currently used for many protocol test suites. In particular, there exist complete test suites in TTCN for various layers of OSI.
6) TTCN has been developed by testers for testers.

TTCN consists of four main components [ISO9646-2a, 3]. These components are briefly discussed as follows:

Test Suite Overview table (Figure 1.5) contains the information for a general understanding of the test suite. It contains references to other documents and describes the purposes of the test suite.

Suite Overview			
Reference to Standards :	ISO Draft International Standard		
Reference to PICS :	Example PICS		
Reference to PIXIT :	Example PIXIT		
How used :	Not applicable		
Test Method :	Remote Test Method		
Comments :	Example Test Suite		
Test Case Id.	**Test Case Reference**	**Page**	**Purpose**
Test_case_1	Test_suite/Test_group_1/Test_case_1	1	Example one
Test_case_2	Test_suite/Test_group_1/Test_case_2	2	Example two

Figure 1.5 - Example of a Test Suite Overview

Declaration tables (Figure 1.6) are used to describe the types of events used in the test suite. These include ASP's which occur at PCO's, timer identifiers, user defined types and operators, test suite parameters, global variables and constants, PCO's, PDU data types and any abbreviations which is used in the test suite specification. Proformas are available for each of these items.

Data Type Declaration		
PDU: X	**Comments: None**	
Protocol Control Information		
Field Name	**Type**	**Comments**
U1	Octetstring	Address
U2	Bitstring	Final bit

(a)

Timer Declarations		
Timer Type Name	**Duration**	**Comments**
Receive-Timer	1..3 min	Mandatory Timer

(b)

Figure 1.6 (a-b) - Example of Declaration Tables

Dynamic Behaviour tables (Figure 1.7) contain columns for behaviour descriptions, labels, constraints references to particular ASP's, or PDU's, verdicts and

comments. As shown in the figure 1.7, there is also some space allocated for specifying synchronization requirements and general comments, if needed. The dynamic behaviour tables start with a heading for test references and a statement of test purpose, followed by the behaviour trees in the behaviour description section. In the event tree, events are listed across the behaviour description section, with alternative events appearing at the same level of indentation.

The behaviour tree test stimuli (sending events) by the tester are identified by "!", and the IUT responses (receiving events) are identified by "?". Other test events used in the behaviour tree are pseudo events, such as OTHERWISE, TIMEOUT and ELAPSE. These are briefly explained as follows:

1) OTHERWISE is used to denote the event of receiving any incoming event not explicitly stated in the preceding alternatives to OTHERWISE.
2) TIMEOUT event is used within a behaviour tree to check expiration of the specified timer. It is not associated with any particular PCO.
3) ELAPSE event puts an upper bound on the time in which a tree will remain at a given level of indentation (i.e. in a given state). It is not associated with any particular PCO.

Test events may be associated with expressions. There are two kinds of expressions; assignment clauses and boolean expressions.

In TTCN, time management is performed using timer operations such as START, CANCEL etc. Semantics of timer operations is given in [ISO9646-2a]. TTCN provides a number of useful facilities, commonly used in protocol testing, via TTCN statements. A subtree can be used in place of a single event, in which case it is referenced by a "+" (i.e. ATTACHMENT), followed by the name of the subtree. Parallel trees are specified with separate PCO's, and synchronization between those is also possible in TTCN. Events may be labelled in the label column. If an event is labelled, it is permitted to GOTO that choice among events at the level of the labelled event from a point lower in the tree to the labelled choice. This is done using the notation:

-> A or GOTO A (A is the label).

The REPEAT statement is another mechanism in TTCN behaviour description which is used to iterate a test step a variable number of times. To emphasize the main path through a test , it is possible to specify separately default behaviours for any subsequent event which a tester may receive (cf. Figure 1.8). The main behaviour descriptions of the dynamic part may be completed by appending default behaviour descriptions to every set of alternatives.

For example, RESET is a command which may be received at any time. Instead of including it in every set of alternatives in the main behaviour, it can be specified using a default behaviour table, as shown in Figure 3.12 of Chapter 3 (the example represents a real test using default behaviours).

Dynamic Behaviour				
Reference :	Test_suite/Test_group_1/Test_case_1			
Identifier :	Test_case_1			
Purpose :	Example Test Case			
Default Reference :	None			
Behaviour Description	**Label**	**Cons. Ref.**	**Verdict**	**Comments**
EX_TEST[L]				Test case header
L! X	LL	X(S0)		A send event
L? Y		Y(S0)		A receive event
+ EX_SUBTREE[L]				Tree attachment
L! A		A(S2)		
L? B		B(S2)	PASS	Pass outcome
L? Z		Z(S1)		
-> LL				
L? OTHERWISE			FAIL	Fail outcome
EX_SUBTREE[L]				Test subtree header
L! C		C(S0)		Start of subtree
L? D		D(S1)		
Synchronization Requirements:	None			
Extended Comments:	None			

Figure 1.7 - Format of a Dynamic Behaviour Table

Default Dynamic Behaviour				
Reference :	Test_suite/Default_group_1/Default_1			
Identifier :	Default_1			
Purpose :	Default Behaviour Example			
Behaviour Description	**Label**	**Cons. Ref.**	**Verdict**	**Comments**
EX_DEFAULT[L]				
L? E		E(S0)		See Comment **
L! F		F(S1)		See Comment **
Extended Comments: **	This behaviour may be anticipated at any level of the levels of alternatives in the test case or test step which references it.			

Figure 1.8 - Format of a Default Dynamic Behaviour Table

All of the TTCN events or statements have semantics rules associated with them which are explained in [ISO9646-2a, 3]. An example of a subset of a real test suite specified in TTCN is provided in Appendix 'B'.

For each entry in the behaviour description there may be an associated verdict which is recorded in the verdict column (i.e. PASS, FAIL or INCONC). Similarly, comments for each entry may be recorded in the comments column. The constraints reference column is used to refer to an instance of a specific constraint placed on parameters of an ASP or PDU. Such constraints are defined in the constraints tables. Usually a constraint reference is given for each and every ASP or PDU that appears in an event line. Alternatively parameter values for the ASP's and PDU's can be recorded and referenced using assignment or boolean statements. However, this is not recommended.

Constraints Tables (see figure 1.9) describe in detail the value settings of parameters for PDU's and ASP's. TTCN provides proformas for such specification. In the constraints reference column of the dynamic behaviour tables references are made to the constraints specified in the constraints tables, and specific binary or hexadecimal values are assigned to parameters.

PDU Constraints			
PDU: X			
List Name	Field Name		Comments
	U1	U2	
S0	'03'H	'001'B	None

Figure 1.9 - Example of a Constraint Table

In TTCN, constraints and declarations may also be specified using the abstract syntax notation (ASN-1) [ISO8824, ISO8825]. In this thesis we only consider the tabular form of TTCN.

1.4 Executable Test Languages

Due to extensive backward and forward referencing in abstract test suites, execution by interpreting abstract tests may turn out to be a very slow process. This makes abstract test notations such as TTCN unsuitable for direct test execution. Therefore a more powerful, executable test case representation is required.

Executable test languages enable users to perform tests more efficiently. In executable test languages, the declaration of PCO's, test suite global variables and constants, and test suite parameters are usually initialized in the test set-up files. Actual values of parameters for the ASP's and PDU's are directly encoded in the executable test code (i.e. there is no concept of constraints references). The executable test languages operate in a basic manner; i.e., they provide facilities to send any ASCII string via a PCO and to receive and verify any string at a selected PCO. In addition, they provide some control mechanisms such as loops and conditional statements for test execution. Comments may also be incorporated to explain events in the executable test cases.

The conversion of ATS to ETS is usually performed by translator tools [Prob88a, Wiles86, ITEX88]. At the University of Ottawa, a translator is being developed to convert test cases written in TTCN to TLAN* equivalents. The following points should be observed when an ATS is converted to an ETS [ISO9646-4]:

1) Each executable test case should be the realization of a single abstract test case.
2) The test purpose and verdict assignments of each abstract test case should be preserved in the corresponding executable test case.
3) The group relationships defined in the abstract test suite should be maintained in the equivalent executable test suite.

1.5 Contribution of Thesis

The objective of this thesis is to report on a detailed study of the test results analysis problem. The main contributions of this thesis are as follows:

1) We present a framework for test results analysis. We break down the problem into manageable portions and propose mechanisms for automating some aspects of the problem.
2) An original contribution in this thesis is the introduction of a notation for describing test results in a higher level, abstract manner. This is the first attempt towards high level analysis of test results. Moreover, the abstract reports include conformance log information.
3) The test language TTCN was found to be very useful, and it is used as the basis for this abstract test results notation. The author along with members of the Protocol Research Group, University of Ottawa, has made original contributions to ISO/IEC JTC1/SC21

* TLAN (Test LANguage) is a proprietary test language of Northern Telecom.

for the development of TTCN as a test notation. He also has made other contributions towards test result analysis related topics in the ISO document, including some of the work in this thesis.

4) We demonstrate that abstract test results reports are a useful vehicle for representing instances of scenarios permitted by the protocol specification.
5) We describe an approach for using the abstract test results notation to assess repeatability, comparability and auditability of test results.

1.6 Organization of Thesis

Chapter 2 describes the current state of the art in test results analysis. It contains a study of the standards proposal and current methods for test results analysis and reporting.

Chapter 3 introduces the abstract test results notation. Examples are provided to show the mapping between abstract test results and abstract test cases. It is also shown how abstract test results can be derived from executable test results. A comprehensive example of actual test results in abstract form is also provided.

Chapter 4 discusses a high level test results analysis methodology and shows how abstract test results may be used in high level test results analysis. We define a mapping from the abstract test results to the protocol specifications given in finite state machine and Estelle normal form specifications. We also propose models for summarizing test results data to expedite diagnosis of test results.

Chapter 5 contains the summary of the thesis and introduces some ideas for further research in the area of test results analysis.

Chapter 2

Test Result Analysis & Reporting

2.1 Objectives and Constraints

Currently the conformance of a system under test is decided upon analyzing the actual test results [Prob88b]. A clear methodology and framework for reporting and analyzing test results is unavailable. Our primary objective is to define an effective methodology and framework for test results reporting and analysis. This requires formalizing the test results reporting process.

As part of our methodology in this thesis, we wish to demonstrate how test results can be related both to test specifications and to protocol specifications. In other words we are proposing a method to overcome the gap which currently exists between test results and protocol standards. Most analysis of test results is currently done manually at the test execution level. We think the key to achieving such a goal is to report test results and carry out analysis at a level higher than the operational level.

As more and more conformance test laboratories are established around the world, it will be necessary to compare and relate conformance test results obtained at different sites on different conformance test systems. We hope to show that our techniques and framework will facilitate such comparison activities.

Diagnosis of test results is another difficult procedure in test results analysis. Often the test analysts spend a great deal of their time locating and relating errors detected in the test results. Our associated objective is to introduce methods for generating graphical summarized result reports to aid the test analyst to arrive at better conclusions in less time.

Our final objective addresses the issue of automation of some of the functionalities discussed above. There is no question that the process of analyzing test results is extremely labour-intensive, "tedious and error prone", when manually performed. In order to accomplish a significant degree of automation, we need to define requirements and formats for the inputs to the test results analysis tools, using the current standardization work by ISO IEC JTC1 SC21 project 23 as a basis. Our intention is to design and illustrate a framework and methods for semi-automated test results analysis.

2.2 Current Draft Standards for Test Results Analysis

In sections 1.2 and 1.3 we briefly discussed conformance testing standards. In this section we discuss in detail the current standards draft proposal for test result analysis and reporting aspect of conformance testing [ISO9646-1,4-5]. The four major topics addressed in the standards with regards to test results analysis are discussed each individually as follows:

1) General Analysis [ISO9646-1,4-5]

The general analysis is concerned with the comparison of observed outcomes with expected or foreseen outcomes. The foreseen outcomes are identified and defined in the abstract test case specifications according to the protocol standard. The observed outcome is the series of events which occurred during execution of test case, usually ending with a verdict. There is no specific guidance as to how this comparison should be done (i.e. manual or automated, during or after execution time). Each test result verdict should be **passed**, **failed** or **inconclusive**. The meanings of "pass" and "fail" are self-evident.

"Inconclusive" verdicts indicate that the test could not be completed satisfactorily so that a verdict of "pass" or "fail" could be assigned by the conformance test system.

The summary results of conformance testing will be documented in a set of conformance test reports. These reports are of two types:

a) a System Conformance Test Report (SCTR) for the complete SUT.
b) a Protocol Conformance Test Report (PCTR) for each of the IUT's within the SUT.

2) Repetition of Results [ISO9646-1]

The results of a credible conformance test should be the same if testing of an IUT is performed more than once (e.g. It is possible that IUT may react differently depending on the load of the machine at the times of testing). Every effort should be made to minimize the possibility that a test case produces different outcomes on different occasions.

3) Comparison of Results [ISO9646-1]

This step involves comparing the overall summary of conformance testing of IUT's tested in different test environments. Some of the items which should be performed very carefully in order to achieve a full comparability are as follows:

a) Design of abstract test case specification.
b) Specification of the real tester which is used to run the test suite.
c) Specification of PICS and PIXIT.
d) Specification of the procedures which should be followed by the test laboratories.
e) A proforma for a conformance test report [ISO9646-1,5], or better a formal syntax for representation of test results [PrAk88].

4) Audit of Results [ISO9646-1]

An audit is the process of reviewing the observed outcomes from the execution of a conformance test suite in order to make sure that all procedures have been correctly followed.

Figure 2.1 provides us with an overview of the inputs required for analyzing test results.

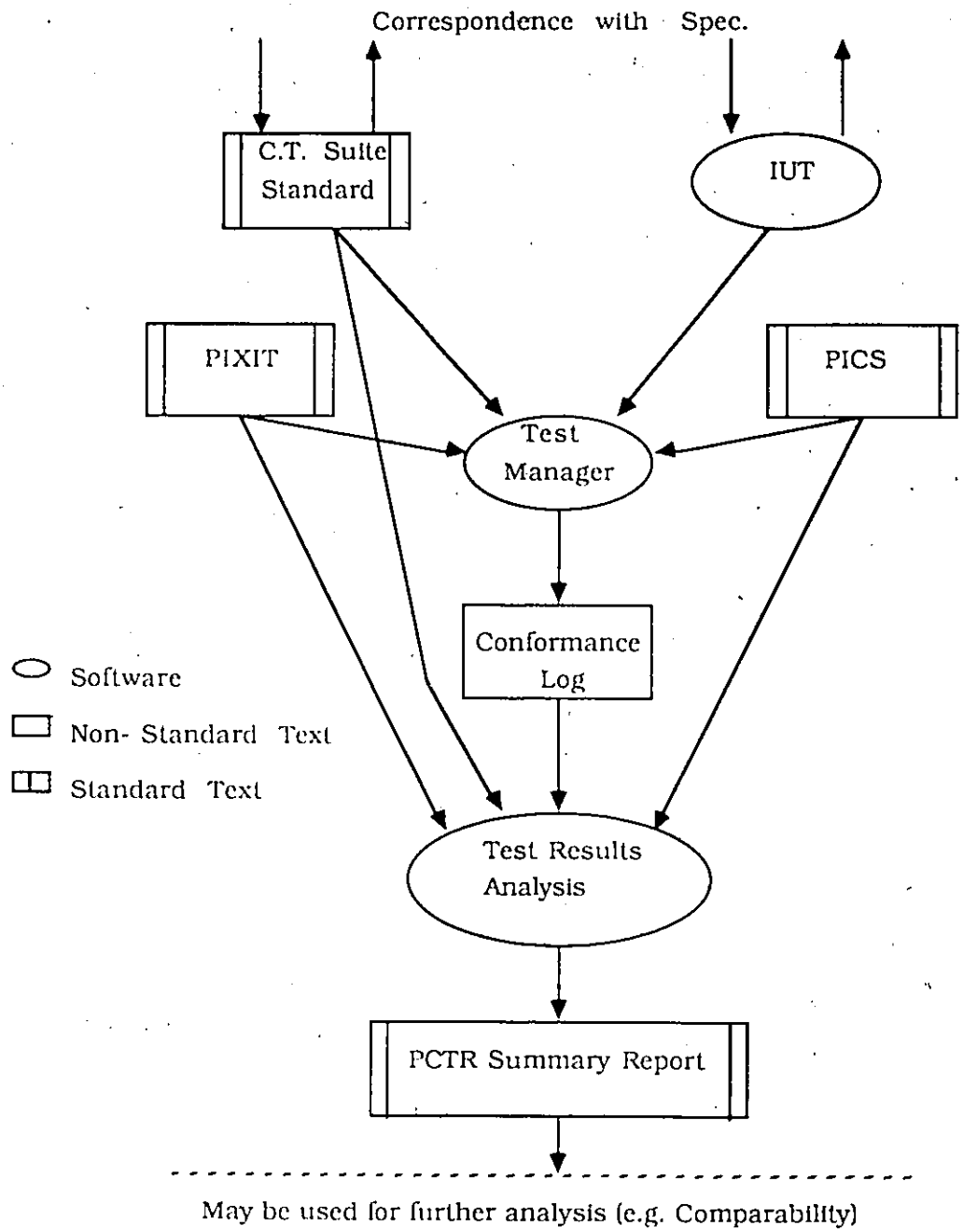


Figure 2.1 - Test Results Analysis as Viewed in the Standards

Most of the emphasis of the current standards documents is on general analysis of test results. Although there seems to be a great deal of interests in repeatability,

comparability or auditability of results, there is not enough guidance given for such activities. The need for automation of such activities is noted, however lack of guidance and the absence of a well-defined framework is an obstacle to automation.

2.3 Executed Test Results (ETR's)

Currently most of the test results reports generated during conformance testing of protocols are in the form of **Executed Test Result reports (ETR's)**, also known as test summaries, test exception reports, and conformance logs. These consist mainly of individual conformance log reports generated by the execution of test cases in the Parameterized Executable Test Suite (PETS). The information in the ETR's varies among different test laboratories. Some of the ETR's may be extremely detailed, whereas others may lack some important information, such as time stamps. In ETR's most of the test results information refers to the conformance log level; therefore, in order to carry out serious analysis, the test analyst must consult the conformance logs. Test summary reports may also be available, which document the number of test cases **failed, passed,** or **cancelled,** out of the number executed. The meaning of "fail" and "pass" is the same as discussed before. "Cancelled" is assigned to the tests which are not relevant to the conformance claims made by the vendor or manufacturer of the IUT.

Although the format of ETR's is test system dependent, the draft standards require that ETR's include the following (* denotes the extra requirements which are introduced by us):

1) Complete data for the summary information sections of the PCTR's. This information may be made available in any form [ISO9646-5].
2) An ETR conformance log for each executed test case containing the following information:
   a) Test case reference, test case identifier, and a statement of the purpose of the test case [ISO9646-4].
   b) An indication of the start and end time of the test case [ISO9646-4].
   c)* A record of the values of the global variables at the beginning and end of the test case execution. This is recommended, but optional.
   d) All protocol data units sent by the lower tester to the IUT and received by the lower tester from the IUT [ISO9646-4].

e) All test events including all abstract service primitives (ASP's) relevant to the test method, together with an identification of the relevant points of control and observation (PCO's) [ISO9646-4].

f)* Time stamps for each timer with respect to each timer operation.

g)* Total number of retries for any sequence of events. This is recommended, but optional.

h) An indication of verdict assignment for each test case [ISO9646-4].

i)* In the case of failure or cancellation a brief diagnostic comment may be included. Supplying such information is recommended, but optional at present.

Where possible, the above information should be presented in a form which can be easily parsed and machine processed. This enables one to develop automated tools to convert ETR's to a more uniform human-readable representation for further processing and/or analysis. An example of an appropriate ETR according to present standards is given in Figure 2.2. The ETR provides the usual report statistics, the start and end times, a record of global variables at the beginning and end of the execution, result identification, number of retries and an end comment to aid the test analyst in his/her manual analysis.

```
*************************************************************************
*************************************************************************
Test Case Reference :    X.25-DTE/LOG/DL4-205
Test Case ID :           LOG_DL4_205
Test Case Purpose :      Verify DTE sends FRMR/W1, CR=0 in response
                         frame
*****************************************
Test Case Start Time :   15:48:34.79
*****************************************
Initial Values for Global Variables:
Resp              =      FALSE
Hyper-DTE         =      FALSE
*****************************************

Sent by 0                :      DISC, 03, *
Timer T1 (Start)         :      15:48:35.79
Timer T1 (Timeout)       :      15:48:35.99
Sent by 0                :      DISC, 03, *
Timer T1 (Start)         :      15:48:36.23
Recv by 0                :      UA, 03, *
Timer T1 (Cancel)        :      15:48:38.52
Sent by 0                :      DM, 01, 0
Timer T1 (Start)         :      15:48:39.42
Recv by 0                :      SABM, 01, *
Timer T1 (Cancel)        :      15:48:41.01
Sent by 0                :      UA, 01, *
Timer T1 (Start)         :      15:48:41.33
Recv by 0                :      I, 01, 00, 1000FB0000
Timer T1 (Cancel)        :      15:48:41.80
Sent by 0                :      I, 01, 1000FF
Timer T1 (Start)         :      15:48:42.90
Recv by 0                :      RR, 03, *
Timer T1 (Cancel)        :      15.48:43.39
Sent by 0                :      HEX, 01FF, 01
Timer T1 (Start)         :      15:48.44.51
Recv by 0                :      HEX, 015E, 01

*****************************************
Initial Values for Global Variables:
Resp              =      TRUE
Hyper-DTE         =      TRUE
*****************************************
Test Case End Time :     15:48:34.79
*****************************************
Number of retries :      1
End Status        :      FAILED
End comment       :      Data Mismatch
*************************************************************************
*************************************************************************
```

Figure 2.2 - Example of Executed Test Results reports (ETR's)

2.4 Analysis of Executed Test Results in Practice

Currently, analysis of test results is performed on executed test results reports (ETR's). These results are analyzed against the ETS (actually PETS, but for brevity we will use ETS from now on) and the corresponding test environment files. From the analysis it may be concluded that errors exist either in the IUT, or the test environment.

Test result analysis of ETR's is a step in the error detection and location process that consists of generating test results reports, analyzing such reports, uncovering and reporting the test environment (e.g. ETS) errors or SUT faults. However, at the ETR level, test verdicts may be quite terse. For example, tests may be reported as simply passed, failed, or cancelled, and the analysis of executed test results is often performed at a very low level. In such an analysis, the results are often not checked against the test suite standard (in TTCN) or the protocol specification standard (in English), which are the original source of requirements. Rather, the analysis begins with diagnosis based on the ETS [Trusc88, Prob88b]. It is only after the test analysts are unable to spot the error, that they decide to refer to the standard test specification and the protocol specification. In other words the analysis is conducted using a bottom up approach. The potential danger of this approach is drawing an inappropriate conclusion because standards were not consulted.

Sometimes, test analysts look at the summary to see how many test cases have passed or failed. This gives them an idea of what kind of an examination they have to undertake. In case of a large number of test case failures, they look for major problems, whereas if a few test cases have failed they look for more detailed problems. This provides some guidance to analysts, and can save them time. In section 4.4, we discuss other analysis approaches.

The test results with "pass" verdicts are not analyzed any further. The assumption is made that it is highly unlikely that the ETS would erroneously assign a "passed" verdict to a non-conforming IUT behaviour.

Cancelled test cases are usually analyzed in order to determine whether the cancellation occurred because the test was not applicable to the IUT, or because of some deficiencies in the test suite or test system environment.

2.5 Summary

In this chapter, we discussed the current status of work in test results reporting and analysis; in particular we examined how test result analysis is currently conducted. We showed the need for specifying a framework for test results analysis and also investigated the need for a notation to represent test results at a higher level of abstraction than the executed test results inorder to achieve a more effective analysis.

In the next chapter we define such an abstract reporting notation and show how executed test results may be converted to their equivalent abstract form.

Chapter 3

High Level Results Reporting

As mentioned in chapter 2, high level test results reporting is the key to carry out analysis at a level higher than the operational level. As well, reporting test results data in an abstract form would allow some degree of automation. In this chapter we introduce **Abstract Test Results reports (ATR's)** and their relation to the current standards [ISO9646-1,5]. We introduce some extra requirements for TTCN test notation in order to achieve our goal of constructing an effective notation to represent abstract test result reports. We then present our notation for abstract test result reporting (TTCN-TR). Finally, we show how ATR's are derived from ETR's and give some examples for ATR's.

3.1 Need for Abstract Test Results (ATR's)

Test result reports from a conformance test consists of two types, a summary type of report which the standard denotes as a Protocol Conformance Test Report (PCTR) [ISO9646-5], and a detailed report often informally referred to as a conformance log which records the complete sequence of all interactions between the IUT and the testing

environment. The standards state some requirements concerning conformance logs. One of these is "the means of testing should provide a means for converting the machine-readable version of the conformance log into a human-readable version, whose format shall be based on the notation used in the reference ATS standard" [ISO9646-4].

In this thesis we satisfy all current standards requirements for conformance logs via a notation referred to as TTCN-TR (Tree and Tabular Combined Notation for Test Results). TTCN-TR is explained in detail in section 3.3. The high level, abstract test reports presented in TTCN-TR are referred to as ATR's (Abstract Test Result reports). ATR's condense the test results output in such a way that:

1) They are comprehensible to the reader. The ATR's behaviour descriptions are higher level than typical conformance logs, yet precise.
2) Multiple standards can be avoided for conformance tests. Clients of different test laboratories in various countries would tend to have the same interpretation of the test reports.
3) A means is provided for cross-validating test results with each other and with the corresponding sections of the protocol standard itself.
4) Reusable documents are created as the reports would no longer be system dependent.
5) A better basis will exist for comparability and auditability of results produced by distinct test laboratories.

3.2 TTCN Enhancements Required to Assist High-Level Results Reporting

Our goal in high-level results reporting is to provide more information-intensive data for analysis and diagnosis of test results. The current available version of TTCN [ISO9646-2a] lacks formal semantics. Moreover, the standard for test specification does not require means of automatically producing such data in our test reports. Two areas which should be enhanced are test case verdicts and test case structures.

a) Enhancements to Test Case Verdicts

Verdicts in ETR's may be "passed", "failed" or "cancelled". These verdicts should usually be enhanced by adding some diagnostic information in the ATR's. Accordingly, the ATS should satisfy properties such as the following:

(Syntactically)

- o The ATS should be restricted to using "fail", "pass" and "inconclusive" as verdicts.

(Semantically)

- o A test case should be terminated only when a "pass", "fail" or "inconclusive" verdict has been assigned.
- o A "Pass" verdict may only be assigned to the leaves of a test case tree. In the case of other test steps an empty verdict column at the leaves of a test subtree means the test should be continued.

b) Enhancements of Test Case Structure

Wherever applicable, test cases should be structured using **preamble**, **testbody**, and **postamble**, where "preamble" and/or "postamble" may be non-existent.

Test body represents the test purpose, preamble is the sequence of event which brings the test to a state where the test body can start, and the postamble is the sequence of events needed to verify arrival at the target state and also, a null response if required.

Note: Other structures are not prohibited, but use of this structure is necessary to facilitate automation of high-level test result reporting and analysis.

In order to correctly parse the test case and identify the preamble and postamble(s), we need to add the following additional syntactic and semantic restrictions on the ATS (See Figure 3.1):

(Syntactically)

- o The test suite specification should explicitly identify the preamble and postamble(s) by references in the test case identification section for each test case.

(Semantically)

- o Preamble should be specified through a single test step attachment in the test case. Such test step may attach other test steps and so on. If no preamble is used the list may remain empty (i.e. None).
- o Postamble(s) may be specified using a single test step attachment for each occurrence. This is due to the fact that different behaviours may require different

postambles to end in different stable states. Again other test steps may be attached to each of the test steps referenced in the test case.

- o Once all the preamble and postamble test steps are identified within a test case, the remaining test steps must be the steps related to the test body (i.e. test steps directly related to the test purpose).

Dynamic Behaviour				
Reference :	TEST_SUITE/GROUP/EXAMPLE			
Identifier :	EXAMPLE			
Purpose :	Example of a test case structure			
Default Reference :	None			
Preamble:	Example:Preamble			
Postamble(s):	Example:Postamble_1, Example:Postamble_2			
Behaviour Description	**Label**	**Cons. Ref.**	**Verdict**	**Comments**
TEST CASE [L]				
+ Preamble [L]				preamble component
L! Event_A				related to purpose
L? Event_B				related to purpose
+ Test_Body_1 [L]				related to purpose
+ Postamble_1 [L]			PASS	postamble comp.
+ Test_body_2 [L]				related to purpose
+ Postamble_2 [L]			PASS	postamble comp.
L? Event_C			INCONC	related to purpose
L? OTHERWISE			FAIL	related to purpose

Figure 3.1 - A typical Test Case Skeleton

3.3 Detailed Definition of Abstract Test Result Reports in TTCN-TR

First, we precisely define ATR's (Abstract Test Result reports). ATR's must contain the following four components:

(For a test session)

1) **Protocol Conformance Test Report (PCTR)**

This is a summary report, i.e., a general concise presentation of the results together with a summary of the verdicts and test system observations [ISO9646-5].

2) **Abstract Test Results Parameter Tables**

This is a record of the values of test suite parameters from executable test environment data files, and also executable test suite global constants used during testing.

(For each test case)

3) **Abstract Test Results Behaviour Tables**

This is the detailed report of each test case execution trace in abstract form (also referred to as an abstract conformance log) which records all interactions between the IUT and the complete testing environment. It is accompanied by the abstract test results parameter tables and abstract test results behaviour reference tables which contain the actual values sent or received and the detail of the times for timer operations. For each abstract test result behaviour there also exists a table which records the initial and final values of global variables.

4) **Abstract Test Results Behaviour Reference Tables**

This section contains the actual parameter settings and control codes for ASP's, PDU's, and their parameters, captured during the test session, including references to invalid captured ASP's and PDU's. In addition, for timer operations, exact times are recorded in timer reference tables, for each timer referenced in that test case. Elapsed time for receive and send events can be calculated from the timer references tables (i.e. by subtracting the end time from the start time in the corresponding timer reference table entry).

ATR's are represented in a tree and tabular form based on TTCN, denoted TTCN-TR. TTCN-TR is a notation which satisfies DP9646 Parts 4,5, but is not necessarily the only suitable notation. We expect further discussions and refinements will be needed to progress TTCN-TR as a standard test result notation and ATR's as a standard report format. We now give a detailed definition and illustration of our proposed ATR format and TTCN-TR.

The last three components are our proposed extensions to the standards [ISO9646-4,5], which basically represent the conformance log at a higher level. Conventions used in this section are adapted from the draft standard document [ISO9646-2a,3]. Our notation (TTCN-TR) supports either of the two following forms:

a) A graphical (TTCN-TR/GR) form, more suitable for human readability.

b) A machine processable form (TTCN-TR/MP) suitable for transmission of ATR descriptions between machines, test laboratories and for other possible automated processing. The syntax of TTCN-TR/MP is defined in Appendix 'A' by means of a BNF grammar.

In the rest of this section we discuss each component of the ATR's in more detail and give examples.

3.3.1 Protocol Conformance Test Report (PCTR)

The proforma for the PCTR is already given as part of the draft standard document [ISO9646-5] and for this reason, we do not go into detail here. The PCTR includes the following information:

a) Identification summary reference to PCTR and SCTR.
b) Introduction to the Implementation Under Test (IUT).
c) Information regarding the test environment.
d) Limits and reservations noticed during test session.
e) Static conformance summary.
f) Dynamic conformance summary.
g) Static conformance review report.
h) Test campaign report. This includes a complete index to the test results conformance log for all test cases in the standard test suite, together with other relevant information such as recording of all unforeseen events.
i) Overall observations.

In the PCTR all the test cases in the abstract test suite are listed. For each of these test cases a summary is given. The example in section 3.4.1 shows how PCTR's are used.

3.3.2 Abstract Test Results Parameter Tables

The abstract test results parameter tables record actual values for test suite parameters as provided by the test environment (Figure 3.2) and for global constants referenced during the test execution (Figure 3.3). There is always one of each table type in an entire ATR. The reason that they are recorded in a separate tables is because same parameter values are used in testing all the tests in a test session. The names of parameters and constants are obtained from the standard abstract conformance test suite (Test Suite Parameter Table & Test Suite Global Constant Table). The values are gathered from the executable test environment files and the executable test suite used in the test session, and are checked against the actual PICS and PIXIT during the test results analysis. Often the test failures are caused by inappropriate or incorrect settings of these parameters, therefore it is useful to keep these values separate from the rest of the test results log for a more

efficient analysis. The proformas for these tables are given in figure 3.2 and 3.3. A complete example based on realistic (LAPB) results is given in section 3.4.2.

Test Environment Parameters	
Name	Value
<Name>	*<Value>*

Figure 3.2 - Test Environment Parameters proforma (for one test session)

Test results Global Constants	
Name	Value
<Name>	*<Value>*

Figure 3.3 - Test Results Global Constants proforma (for one test session)

3.3.3 Abstract Test Results Behaviour Tables

Abstract test results behaviour tables condense the private details of the conformance log of each test case in separate tables (Figure 3.4(a-b). Test analysts usually refer to the results behaviour tables when they would like to thoroughly analyze one particular test case result. The abstract test results behaviour tables consist of the following items (see Figure 3.4(a)). (Most of these items are required by the standards. Some of them are added by us which are useful during analysis):

a) **A reference:** The reference gives a name representing context information to the test case results. The reference specifies the location of the corresponding abstract conformance log.

b) **An identifier:** The log identifier is a short name which may be used interchangeably with a reference. The log identifier must be unique within the set of test results behaviour tables. It will be used to relate the tables within the ATR's for each test results behaviours.

c) **Global table identifier:** This is an identifier which is unique within an ATR for a test session. It is used to reference the global variable tables for each test results

behaviour. The proforma which is used to record the values for global variables is given in Figure 3.4(b). This information will be used by limited number of people who would like to investigate more about the behaviour of the test results. It is separated into a distinct table inorder to keep the test results behaviour table as simple as possible. The initial and final values may be used to show that the assignments were executed correctly by the test system according to the ATS. One should be able to arrive to the final value of a global variable by applying all the assignment statements in the test results behaviour to its initial value. Each table only contains the list of global variables specified in the ATC and referenced in its corresponding test results behaviour table.

d) **A statement of purpose:** This is the statement of the purpose of the test case. It is the same as the one in the abstract test case or equivalent executable test case.

e) **Conformance log description:** This section is a detailed description in TTCN-TR of the actual test events which occurred during testing. It includes all other related information about each event in a number of columns.

f) The **start time** and **end time** of each test case.

g) **Number of retries** which occurred in each test case.

h) The **Final verdict** of the test case. In cases of failure or inconclusive verdicts (i.e. cancelled or failed in ETR's), a qualification of 1, 2, 3 is noted depending on location of the verdict assignment, i.e., preamble, test body or postamble respectively.

Test Results Behaviour					
Reference :	*<Test Results Log Reference>*				
Identifier :	*<Identifier>*				
Global Table id. :	*<Test Results Global Table Identifier>*				
Purpose :	*<Test Purpose>*				
Log Behaviour Description	**E**	**ATS event Location**	**Segments Start**	**End**	**Behaviour References**
<Log Behaviour Description>	*<E>*	*<Location>*	*<S>*	*<E>*	*<Reference>*
Test Case Start Time :	*<Start time>*				
Test Case End Time :	*<End Time>*				
Num of Retries :	*<Num of retries>*				
Verdict	*<Test verdict>*				
Comment	*<Comment>*				

Figure 3.4 (a) - Test Results Behaviour proforma (for one test case)

Test results Global Variables		
Reference : *<Test Result Log Reference>* **Global Table Id :** *<Test Result Global Table Identifier>* **Behaviour Table Id :** *<Test Result Behaviour Table Identifier>*		
Name	**Initial Value**	**Final Value**
<Name>	*<Value>*	*<Value>*

Figure 3.4 (b) - Test Results Global Variables proforma (for one test case)

The most complex part of a test results behaviour table is the **Conformance Log behaviour description** part (See Figure 3.4(a)). It is composed of lines consisting of several entries. Each line in the conformance log description in the test results behaviour table consists of (i) and (iii) and possibly (ii), (iv) or (v) depending on the particular event type, where items (i) to (v) are described as follows:

i) **Log behaviour description**: This may be a test case or test step name, send or receive event, timer operations or expressions such as assignments or booleans. The behaviour description syntax used in TTCN [ISO9646-2a] for these types of behaviours is fully adopted in this thesis (except the indentation style). For log behaviour description the test event sequences for the preamble and postamble are indented for clarity.

   A continuation character is used to handle the cases where a line in the log behaviour description is too long. The character '#' would be prefixed at the beginning of the log behaviour description to demonstrate continuation (this may also be used to indicate continuation in other columns of the event line).

ii) **Enumeration column** ('E'): This column is used to count occurrences of various TTCN constructs, such as repeat constructs, which appear in the conformance log description.

iii) **Location identifier**: Used to give the details of the actual event location within the standard abstract test suite. For each event occurrence a test case name or test step name is supplied in this column. By looking at this column, one can easily see which parts of the abstract test suite is referenced by an abstract conformance log of a test case. This will be used to identify the relationships between the test results.

iv) **Segments column**: Used to identify the start and end of particular blocks of entries in the conformance log description. The start and end of trace segments such as traces of preambles, traces of postambles, test bodies, or default behaviours can be recorded, as well as start and end of other TTCN constructs (two columns are used for start and end of segments in order to increase the readability).

v) **Behaviour reference column**: Used to point to the additional information which is captured for send and receive events (the details about ASP's and PDU's) and timer events (start and end of each timer during the testing session) in the behaviour table. In particular, send events are usually associated with a "start timer"; in this way, timestamps are provided for send events. Similarly, receive events are usually associated with a "cancel timer", in which case the timestamps for receive events may also be provided. The syntax that we use to point to the behaviour reference table is given as follows:

ASP or PDU or TimerName followed by "(" Si ")", where
Si ($i \geq 0$) is a reference index.

This corresponds to the TTCN syntax.

Now we give examples to show how execution traces of TTCN constructs in an ATC can be represented (in TTCN-TR) in an ATR.

a) Repeat Construct

The start of an execution trace of a repeat construct in the ATR is denoted by the symbol Ri, where ($i \geq 1$), in the start segment column. Each repetition of the loop is denoted by a '*' symbol in the 'E' (Enumeration) column. The '*' is only recorded for the first line in each repetition. The end of a repeat construct is represented by the same Ri symbol. The '*' symbol is always accompanied by (z), where z denotes the number of times the block is repeated.

Example 3.1: Comparison of a REPEAT construct in an ATC behaviour description in TTCN (Figure 3.5) and its execution trace represented as an ATR log behaviour description in TTCN-TR (Figure 3.6).

Behaviour Description	Label	Cons. Ref	Verdict	Comments
MAIN-TREE [L]				
(Resp := FALSE)				
REPEAT SUB-TREE [L] Count				
[Resp = TRUE]			PASS	Repeat Guards
[Count > N2]			FAIL	After N2 retries
SUB-TREE [L]				
L! DISC		DISC(S1)		
(START T1)				
? TIMEOUT T1				
L? UA		UA(S2)		
(Resp = TRUE)				
(CANCEL T1)				
.				
.				Rest of Test

Figure 3.5 - Example of an ATC Repeat Construct in TTCN

Figure 3.5 contains only a part of the test behaviour tree, but enough to provide the execution trace shown in Figure 3.6. In the test case, N2 is a protocol-defined retransmit count limit. The REPEAT construct is exited either when count is exceeded its maximum number (i.e. N2) or when an unnumbered acknowledgement response has been received. We can see directly from the trace in the ATR (Figure 3.6) that the IUT timed out after the first try before responding after a second try. Since the IUT successfully responded, a "pass" verdict will be assigned (the verdict is not included in Figure 3.6 - figure represents only a part of the test results behaviour table).

Log Behaviour Description	E	ATS event Location	Segments Start	End	Behaviour References
MAIN-TREE [L]			TS		
(Resp := FALSE)		Test case id			
L! DISC	*(1)	Test case id	R1		DISC(S0)
(START T1)		Test case id			T1(S0)
? TIMEOUT T1		Test case id			T1(S0)
L! DISC	*(2)	Test case id			DISC(S0)
(START T1)		Test case id			T1(S1)
L? UA		Test case id			UA(S0)
(Resp := TRUE)		Test case id			
(CANCEL T1)		Test case id			T1(S1)
[Resp = TRUE]		Test case id		R1	
MAIN-TREE [L]				TE	

Figure 3.6 - Example ATR Trace of Repeat Construct using TTCN-TR

b) GOTO Construct

GOTO constructs may be represented in a similar way. It is assumed that results are logged sequentially, therefore presence of a "goto" is not known until we have reached the point where results indicate that such a "goto" has occurred. Hence, the natural semantics of "goto" does not allow us to uniquely identify the beginning of a "goto" segment. However, we are capable of counting the number of times that jumps occur. We use a '+' sign and an integer (z), where *z* denotes the number of "goto" occurrences. The number shown by z is always the number of times the "goto" is repeated.

Example 3.2: Comparison of a "goto" construct in an ATC behaviour description in TTCN (Figure 3.7) and its execution trace represented as an ATR in log behaviour description in TTCN-TR (Figure 3.8).

Behaviour Description	Label	Cons. Ref	Verdict	Comments
MAIN-TREE [L]				
(Resp := FALSE)				
(Count := 1)				
+SUB-TREE	A			
(Count := Count + 1)				
[Resp = TRUE]			PASS	Check1
[(Resp = FALSE) and (Count≤N2)]				Check2
--> A				
[(Resp = FALSE) and (Count>N2)]			FAIL	Check3
SUB-TREE [L]				
L! DISC		DISC(S1)		
(START T1)				
? TIMEOUT				
L? UA		UA(S2)		
(Resp := TRUE)				
(CANCEL T1)				
				Rest of Test

Figure 3.7 - Example of an ATC GOTO Construct using TTCN

This behaviour is equivalent to Example 3.1. Again we have only included part of the test behaviour, enough to provide the execution trace shown in Figure 3.8. There are three boolean expressions in the main-tree behaviour description (commented as check1, check2 and check3 in Figure 3.7). First one happens if a response is received, in which case we

may assign a "pass" verdict. The second happens if no response is received and we have not yet exceeded the maximum number of retransmission. Finally, if we exceed the maximum number of retransmission and we have not yet received a response, we may then assign a "fail" verdict. We can see from the ATR's that again we had to repeat the transmission once inorder to get a response.

Log Behaviour Description	E	ATS event Location	Segments Start	End	Behaviour References
MAIN-TREE [L]			TS		
(Resp := FALSE)		Test case id*			
(Count := 1)		Test case id			
L! DISC		Test case id			DISC(S0)
(START T1)		Test case id			T1(S0)
? TIMEOUT T1		Test case id			T1(S0)
(Count := Count + 1)		Test case id			
[(Resp = FALSE) and (Count ≤ N2)]	+(1)	Test case id			
L! DISC		Test case id			DISC(S0)
(START T1)		Test case id			T1(S1)
L? UA		Test case id			UA(S0)
(Resp := TRUE)		Test case id			
(CANCEL T1)		Test case id			T1(S1)
[Resp = TRUE]		Test case id			
MAIN-TREE [L]				TE	

Figure 3.8 - Example ATR Trace of GOTO Construct using TTCN-TR

□

c) Parallel Trees & Synchronization

Parallel Trees consist of separate behaviour descriptions which can execute in parallel provided that certain **synchronization constraints** are respected. Once the tests are run the behaviours (execution traces) of these parallel trees may be treated as a serialized segment. It is useful to know where in the log behaviour description parallel trees were used. The start and end of parallel trees can be identified using Pi symbol, where ($i \geq 1$), in the start segment column and end segment column respectively.

As synchronization may only be used between parallel trees, we may show synchronization using enumeration column (i.e. 'E'). For example, if we have a synchronization requirement (a > b), we will mark the trace to designate the occurrence of a

* Note: "ATS event Location" is specified as "Test case id" for all events because here we are not referring to a complete test suite and therefore we are unable to specify an accurate "ATS event Location". "ATS event Location" has several uses and one of them is discussed in section 4.4.

synchronization checkpoint by using a symbol '>' followed by (z) for event 'a' and '<' followed by (z) for output event 'b', where z is an integer identifier for this synchronization.

Example 3.3: Comparison of parallel trees and synchronization in an ATC behaviour description in TTCN (Figure 3.9) and its execution trace represented as an ATR log behaviour description in TTCN-TR (Figure 3,10).

Behaviour Description	Label	Cons. Ref	Verdict	Comments
MAIN-TREE [UT, LT]				
|| LT-TREE [LT],UT-TREE[UT]				
UT-TREE [UT]				
UT? DATAind	L	DATAind(S0)		
LT-TREE [LT]				
LT! DATAreq [D-bit-set = TRUE]	L1	DATAreq(S1)		with D-bit set
(START T1)				
LT? DATAind [P(R) < this-packet]	L2	DATAind(S1)		
(CANCEL T1)				
LT? DATAind [P(R) < this-packet]	L3	DATAind(S2)		
(CANCEL T1)			PASS	
Synchronization Requirements: LT-TREE/L3 > UT-TREE/L				

Figure 3.9 - Example of an ATC Parallel Trees and Synchronization Construct in TTCN

Tests specified in TTCN should be always serializable; thus, this use of parallel trees is different than the use of parallelism in formal description of protocols (i.e. parallel trees in TTCN do not mean parallel processes). In Figure 3.9 when event "L1" is executed, at that stage "L2" together with "L" can be considered as the next set of alternatives event. However "L3" can not be considered since there is a synchronization requirement stating that "L" should take place before "L3" can take place. So if "L" happens then "L3" would be a valid alternative in the next consideration. (In case of similar events with different constraints in the ATC, behaviour references are consulted in the ATR to find out which events occurred, i.e., "L", "L2" or "L3").

Log Behaviour Description	E	ATS event Location	Segments Start	Segments End	Behaviour References
LT! DATAreq [DT packet with D-bit set]		Test case id	P1		DATAreq(S0)
(START T1)		Test case id			T1(S0)
UT? DATAind	<(1)	Test case id			DATAind(S0)
LT? DATAind [P(R)'< this packet]	>(1)	Test case id			DATAind(S1)
(CANCEL T1)					T1(S0)
				P1	

Figure 3.10 - Example ATR Trace of the Parallel Trees and Synchronization Construct using TTCN-TR

□

d) Other Useful Trace Segments

Groups of events within an abstract test case such as default behaviours, preambles, test bodies, and postambles which constitute a segment and are of importance for test result analysis may also be highlighted in the log behaviour description using start and end symbols in the segments column. Figure 3.11 provides a list of these symbols.

		Start	End
Test case Start and End	:	TS	TE
Preambles	:	PA	PA
Test bodies	:	TB	TB
Postambles	:	PO	PO
Default Behaviours	:	DB	DB

Figure 3.11 - Start and End symbols for various segments in TTCN-TR

Default behaviours are valid protocol behaviours which may be appended to every set of alternatives in the main behaviour descriptions. IUT may execute default behaviours at any stage of execution. This will put the IUT in a stable state, which also means that the IUT has not satisfied the test purpose (i.e. test purpose is incomplete). Hence, it is useful to record such sequences of events in the ATR's, and consider it during analysis.

Example 3.4: Comparison of specification of allowed default behaviours in ATC behaviour description in TTCN (Figure 3.12) and associated execution trace in the ATR log behaviour description in TTCN-TR (Figure 3.13).

Note: In this example timer operations (in the ATC) and time stamps.(in the ATR) are not included, inorder to keep the example simple.

Behaviour Description	Label	Cons. Ref	Verdict	Comments
MAIN-TREE [L]				
L! CONreq		CONreq(S0)		
L? CONconf		CONconf(S0)		
L! DATAreq		DATAreq(S1)		
L? DATAind		DATAind(S1)		
L! DISreq		DISreq(S2)	PASS	

(a)

Behaviour Description	Label	Cons. Ref	Verdict	Comments
DEFAULT-TREE [L]				Inconclusive
L? RSTind		RSTind(S0)		since purpose
L! DISreq		DISreq(S2)	INCONC	is incomplete

(b)

Figure 3.12 (a-b) - Example of an ATC Default Behaviour in TTCN

The purpose of the test case is make a connection establishment. However, after sending a connection request a reset indication is received which is responded by a disconnect request from the lower tester. This is not an invalid behaviour according to the protocol specification, however, it prevents us from achieving our test purpose. Therefore an "inconclusive" verdict may be assigned. Figure 3.13 only represents the log behaviour description part of the abstract test result behaviour table, that is why the verdict can not be seen.

Log Behaviour Description	E	ATS event Location	Segments Start	Segments End	Behaviour References
MAIN-TREE [L]		Test case id	TS		
L! CONreq		Test case id			CONreq(S0)
DEFAULT-TREE [L]		Default id	DB		
L? RSTind		Default id			RSTreq(S0
L! DISreq		Default id		DB	DISreq(S0)
MAIN-TREE [L]		Test case id		TE	

Figure 3.13 - Example ATR Trace of Default Behaviours using TTCN-TR

□

3.3.4 Abstract Test Results Behaviour Reference Tables

Abstract Test results behaviour reference tables are used as to record the actual coding of parameters in ASP's and PDU's (i.e. using parameter reference tables) and the time stamps related to each timer events (i.e. using timer reference tables). This is used to keep the conformance log detail from overwhelming the ATR reader by maintaining the actual values in a separate location.

These tables are similar to the TTCN constraints reference tables. The outline for parameter reference tables can be automatically generated using the standard abstract test suite. The format of the tables is exactly the same, except for a comments column (see Figure 3.14). On receipt of a PDU or ASP, a reference name is recorded in the behaviour reference column of the test results behaviour table. The list of the captured values for such a PDU or ASP would then be recorded in the related parameter reference table, together with the same reference name. Reference names for each table start with "S0" (for example) and continues upto "Sn" (for example). Using these tables all parameters of the PDU's or ASP's, invalid as well as valid, can be recorded and subsequently analyzed.

PDU References			
PDU:.. *<Name>*			
Behaviour Table Id : *<Test Result Behaviour Table Identifier>*			
List Name	Field Name		
	<Field Name>1		*<Field Name>m*
<Reference Name>1	*<Value>1,1*	....	*<Value>1,m*
.....	.....	....	.....
<Reference Name>2	*<Value>2,1*	....	*<Value>2,m*
<Reference Name>n	*<Value>n,1*	....	*<Value>n,m*

Figure 3.14 - Proforma for Parameter Reference Tables (for one test case)
(For ASP parameter tables replace the word PDU)

Other useful data are records of times associated with timers. The proforma for timer references is given in Figure 3.15.

Timer References		
Timer: *<Timer Name>*		
Behaviour Table Id : *<Test Result Behaviour Table Identifier>*		
List Name	**Field Name**	
	Start Time	**End Time**
<Timer Reference Name>1	*<Time>x1*	*<Time>y1*
<Timer Reference Name>2	*<Time>x2*	*<Time>y2*
.....	.....	.....
<Timer Reference Name>n	*<Time>xn*	*<Time>yn*

Figure 3.15 - Proforma for Timer Reference Tables (for one test case)

For each timer and for each test case separate table is required. Each reference consists of the timer name, together with the reference names obtained from the corresponding event behaviour references column of the test results behaviour table. The time stamps are recorded sequentially in the timer reference tables using reference names (e.g. S0, ... , Sn). The time stamps recording for the timer operations and timer pseudo events in TTCN-TR is assigned according to the guidelines presented in Figure 3.16 (e.g. Cancel is always associated with type end-time, where as start is always associated with type start-time).

		Start Time	End Time
Start Timer Operator	:	✓	✗
Cancel Timer Operator	:	✗	✓
Resume Timer Operator	:	✓	✗
Suspend Timer Operator	:	✗	✓
Elaspe Pseudo Time Event	:	✓	✓
Timeout Pseudo Time Event	:	✗	✓

Figure 3.16 - Guidelines for Recording Time Stamps for Timers

Examples of abstract behaviour reference tables are available in Figure 3.20, 3.21 of section 3.4.2, where a complete ATR of a test case for LAPB is given. Note especially the use of this time stamps where explicitly designated in the ATS. (The serious reader is encouraged to study the LAPB example in section 3.4.2 carefully).

Thus, the syntax and semantics of ATR's are designed to provide sufficient detail from the execution trace of an ETS to allow verification that the ETR is the trace of a valid implementation of tests in the ATS.

For completeness, we need to show how to derive ATR's from the corresponding ETR's in a uniform way. This is done in the next section.

3.4 A Method for Deriving ATR's

We now give a uniform approach to deriving ATR's from ETR's (containing conformance logs), ATS, PICS, PIXIT, and the actual executable test environment files with a view to automated derivation. The detailed design of such a derivation algorithm however, is beyond the scope of this thesis. Our assumptions with regards to the derivation are as follows:

1) We assume that only traces of test events such as send/receive and timer operations exist in the ETR.
2) It is assumed that the ETS faithfully represents the booleans and assignment statements of the ATS and the test system executes such statements correctly. (It is also assumed that these statements are specified correctly in the ATS).
3) If at any stage of the ATR derivation the current test event record (i.e. send/receive or timer event) in the ETR does not match any of the ATS-specified alternative events (i.e. TTCN test events - send/receive, TIMEOUT, ELAPSE, OTHERWISE or timer operations) the derivation process can be terminated with comments that ATS is incomplete. Thus, we assume the ATS is complete for successful ATR derivation.
4) It is also assumed that there is a well-defined mapping between the names of the parameters, global constants or variables in each of the ETS and ATS.

3.4.1 Description of the ATR derivation Method

In this section, we explain how each of the four components of the ATR may be derived from the ETR using ATS or other related documents. In case of the informational subsections of the components we explain where the information comes from.

a) Protocol Conformance Test Report (PCTR)

The PCTR is required by standards [ISO9646-5]. It can be automatically derived from the machine processable PCTR information in ETR. Otherwise, if the information is not available in a machine processable form, or some parts of the information is missing in that format, it can be input to the derivation tool by other means (e.g. somebody typing the information in the available proforma).

b) Test Results Parameter Tables

The information in the test results parameter tables and global constants tables can be derived from the test environment files used during test execution (e.g. the maximum allowed response time of the IUT). Test environment files are usually machine processable, therefore this part of the ATR may be derived automatically.

c) Test Results Behaviour Tables

The test results behaviour table header can be generated automatically using the conformance log of the ETR, and should be similar to that of PCTR conf. log. [ISO9646-5]. Data such as identifier and test purpose may be obtained from the ATS.

The abstract conformance log behaviour description is the most complex part of the derivation. It is derived using ETR test case conformance log in parallel with the ATS specification of the test case as follows (this procedure can be automated):

i) Select the first event in the ETR test case conformance log.

ii) Make the first level of alternatives in the ATC the current level of alternatives and initialize one queue. As the first level of alternatives is always a main behaviour tree name, push it into the queue and make the next level of alternatives as current level of alternatives.

iii) Make copies of the queues to cover all the different combinations of traces (upto this point). Now that we have enough queues to record all different paths, for each of the events in the current level of alternatives of the ATC (including the first level of alternatives in the default behaviour) perform the appropriate action below:

- o If the statement* is the test step behaviour tree name, TTCN expression (i.e. Boolean or assignment statement), or TTCN test event (send/receive, timer event or OTHERWISE) add it to appropriate queue to mark a new trace in the ATC.
- o If the statement is a tree attachment, add the first level of alternatives in the attached tree to the appropriate queue to mark a new trace in the ATC, and if entering a preamble or postamble, record for later addition to the segment column.

---

* Note: A statement in TTCN may be a test case name (TCname[...][...]), test step name (TSname[...][...]), TTCN test event such as send or receive event (! ... or ? ...) etc., a boolean expression ([...]), assignment statement ((...)), tree attachment (+...), repeat (REPEAT ...), 'goto' ('-->'...or GOTO ...) or parallel trees (||...).

- If the statement is a repeat or 'goto' statement, enter the repeating block and add the first level of alternatives in the repeat or 'goto' block (for 'goto' the block starts from the first event after the jump has occurred) to the appropriate queue to mark a new trace in the ATC, record the initial event of the block for the segment column, and keep count of the number of times the block is repeated for the 'E' column.
- If the statement is a parallel tree event in the ATC, enter the parallel trees and add the alternatives in the subtrees to the appropriate queue to mark a new trace in the ATC, and set a flag for entering the block (to be later added to the segment column).

iv) Now try to match the ETR test case event with the items at the most rear end of queues and perform one of the followings:
- If there is no match and none of the rear end items in the queues is a TTCN test event, proceed to the next level (i.e. instruction 'vi').
- If there is no match while there are some rear end items in the queues which are TTCN test events, terminate the derivation, and report that an instance of possible non-conformance to ATS has been detected (proceed with instruction 'vii').
- If there is a match with one of the rear end items in the queues, write the content of the matched queue in the conformance log description (i.e. FIFO), including the matched event. (for this item 'd' actions should also be carried out). At this point restart the tracing process from this point in the ATC, by initializing an adequate number of queues for different alternative events which follow and proceed with the next instruction (i.e. instruction 'v').
- If there is more than one match (i.e. The TTCN event is the same, but different boolean conditions), add the TTCN test events to the appropriate queues to mark new traces in the ATC and continue to the next level (instruction 'vi').

v) Select the next event in the ETR conformance log. If no more ETR conformance log events are left, proceed with instruction 'vii'.

vi) Assign the next level of alternatives to current level of alternatives and repeat process from instruction 'iii'.

vii) If end of a trace is encountered (i.e. End of conformance log is reached and has been matched to the end of a trace in the ATC) with success, add the start and end time of the test, the verdict (according to the ATC), and the comments (from ETR) to the test results behaviour table.

The algorithm above is quite complex and therefore the description is given in a high level nature. It was necessary to abstract the essense of the process and avoid getting mired into detail.

d) Test Results Behaviour Reference Tables

For each PDU, ASP or timer event, the reference tables should be updated. The updating of the reference tables takes place following one of the two procedures below:

i) If a table for the PDU, ASP or timer event is non-existent, create the table (using the ATS), enter the reference item using S0 as the reference code for the first entry, upgrade the code to S1 for the next entry.

ii) If a table for the PDU, ASP or timer event exists, add the new reference item to the table and upgrade the reference code from Si to Si+1 for the next entry.

Creation of test result behaviour reference tables can be automated using the available ATS constraint tables proformas to start with, and filling it with the actual captured values.

This completes the derivation of the ATR. Note that inappropriate test result behaviours are identified and flagged as potential instances of non-conformance to the ATS. Moreover, provided that the ETR's are machine processable with an identified format, this process can be automated.

3.4.2 An Example : LAPB Abstract Test Result reports (ATR's)

Here we present the ATR of a test case (see Appendix 'A') extracted from the working document X.25-DTE data link layer conformance test suite [ISO8882-2]. The corresponding PICS and PIXIT are given in Appendix 'C' and 'D' respectively. The simulated results for this test case are represented in TTCN-TR. The ATR is generated from a given ETR (not provided in this thesis - a similar example is available in chapter 2, Figure 2.2) and the corresponding executable test environment files (which are equivalent to PICS and PIXIT).

This example clearly indicates that the requirements in standards documents [ISO9646-4,5] are utilized, using TTCN-TR notation. All the qualities of ATR's stated in section 3.1 are illustrated in this example.

a) Protocol Conformance Test Report (PCTR)

The first section is the summary report of the results known as Protocol Conformance Test Report. (Figure 3.17). Note that in PCTR's all the test cases in the standard test suite are listed. However, other sections (e.g. test results behaviour tables etc.) can either be generated for all the executed test cases (detailed ATR's) or only for the test cases which indicated instances of non-conformance (exception ATR's).

Protocol Conformance Test Report (PCTR) for DATA LINK LAYER							
Section1: Identification Summary							
(i) PCTR Introduction							
PCTR Number: 2							
PCTR Date : 24/04/1988							
Corresponding SCTR Number: 100							
Corresponding SCTR Number : 24/04/1988							
Test Laboratory Manager : Robert L. Probert							
Test Manager Signature : R. L. P.							
(ii) IUT Introduction							
IUT Name : Data Link Layer IUT							
IUT Version : 1.1							
Protocol Standard : ISO Draft International Standard for X.25 LAPB							
PICS : Data Link Layer PICS							
Previous PCTR : None							
(iii) Testing Environment							
PIXIT : Data Link Layer PIXIT							
Abstract Test Suite Standard : Data Link Layer ATS							
Abstract Test Method : Remote test system							
Real Tester id : NTS							
Executable Test Suite id : Data Link layer ETS							
Protocol layer information : None							
Protocol Sub-layer information : None							
Dates of testing : 24/04/1988							
Test Operator(s) : Shahram Akhavan							
Test Analyst(s): Shahram Akhavan							
Conformance Log: X.25-DTE/RES							
(iv) Limits and Reservations							
Comments: None							
Section2: Static Conformance Summary							
Comments: IUT does/does not conform statically to the specified protocol standard.							
Section3: Dynamic Conformance Summary							
Comments: A descriptive summary given on the results of groups of tests.							
Section4: Static Conformance Review Report							
Comments: Itemization of the mismatches between PICS and SCR.							
Section5: Test Campaign Report							
ATS Ref	**SATS?**	**PETS?**	**Run?**	**Log Ref**	**Verdict**	**Unforeseen**	**Obs.**
....	....	....	....	....	....	....	....
DL4_205	Yes	Yes	Yes	X.25-DTE/ LOG/ DL4_205	PASS	None	None
....	....	....	....	....	....	....	....
Section6: Overall Observations							
Comments: None							

Figure 3.17 - Protocol Conformance Test Report for LAPB ATR

b) Test Results Parameter Tables

For this test case no global constant is used, therefore the test results global constants tables does not exist. However there are two test suite parameters which take their values from the given executable test environment files (not shown here) and they are recorded in the test environment parameters tables (see figure 3.18).

Test Environment Parameters	
Name	Value
T1	3
N2	5

Figure 3.18 - Test Environment Parameters Table for LAPB ATR

c) Test Results Behaviour Table

The test results behaviour presented in Figure 3.19 is the behaviour resulted from executing the DL4_205 test case and its corresponding test steps (Appendix 'B'). This example of test results behaviour contains a sequence of events (i.e. preamble) to bring the IUT to a stable state, where the sequence of test purpose events can be applied (i.e. test body). Test body is executed and an undefined command (Hex) is sent (T1 timer is started), which is received as a frame reject (T1 timer is cancelled). The test purpose is successfully achieved. Hence, to complete the test a sequence of postamble test events is applied to put the IUT back into a stable state. Test results verify that this step is also successfully undertaken, therefore a "pass" verdict is assigned. During this test a sequence of events was repeated, so the number of retries equals one. The times of start and end of the test case execution is also recorded, in case they are required for later analysis.

Test Results Behaviour						
Reference :	X.25-DTE/LOG/LOG_DL4_205					
Identifier :	LOG_DL4_205					
Global Table Id. :	DL4_205_GBL					
Purpose :	Verify DTE sends FRMR/W=1, CR=0 in response to undefined command received in L4 state.					
Log Behaviour Description		**E**	**ATS event Location**	**Segments Start**	**End**	**Behaviour References**
DL4_205 [L]				TS		
DL4_STATE [L]			DL4_205	PA		
DL2_STATE [L]			DL4_STATE			
DL1_STATE [L]			DL2_STATE			
(Hyper_DTE := FALSE)			DL1_STATE			
(Resp := FALSE)			DL1_STATE			
L! DISC		*(1)	DL1_STATE	R1		DISC(S0)
(START T1)			DL1_STATE			T1(S0)
? TIMEOUT			DL1_STATE			T1(S0)
L! DISC		*(2)	DL1_STATE			DISC(S1)
(START T1)			DL1_STATE			T1(S1)
L? UA			DL1_STATE			UA(S0)
(Resp := TRUE)			DL1_STATE			
(CANCEL T1)			DL1_STATE			T1(S1)
[Resp = TRUE]			DL1_STATE		R1	
(Resp := FALSE)			DL2_STATE			
L! DM		*(1)	DL2_STATE	R2		DM(S0)
(START T1)			DL2_STATE			T1(S2)
L? SABM			DL2_STATE			SABM(S0)
(Resp := TRUE)			DL2_STATE			
(CANCEL T1)			DL2_STATE			T1(S2)
[Resp = TRUE]			DL2_STATE		R2	
L! UA			DL4_STATE			UA(S1)
(START T1)			DL4_STATE			T1(S3)
L? I			DL4_STATE			I(S0)
(CANCEL T1)			DL4_STATE			T1(S3)
L! I			DL4_STATE			I(S1)
(START T1)			DL4_STATE			T1(S4)
(Hyper_DTE := TRUE)			DL4_STSTE			
L? RR			DL4_STATE			RR(S0)
(CANCEL T1)			DL4_STATE		PA	T1(S4)
L! HEX			DL4_205	TB		HEX(S0)
(START T1)			DL4_205			T1(S5)
L? FRMR [Hyper_DTE = TRUE]			DL4_205			FRMR(S0)
(CANCEL T1)			DL4_205		TB	T1(S5)
DL5_CHK [L]			DL4_205	PO		
L! RRC			DL5_CHK			RRC(S0)
(START T1)			DL5_CHK			T1(S6)
L? FRMR			DL5_CHK			FRMR(S1)
(CANCEL T1)			DL5_CHK		PO	T1(S6)
DL4_205 [L]				TE		
Test Case Start Time :	15:48:34.79					
Test Case End Time :	15:48:45.39					
Num of Retries :	1					
Verdict :	PASS					
Comment :	None					

Figure 3.19 (a) - Test Results Behaviour Table for DL4_205 Test Case

Test results Global Variables		
Reference:	X.25-DTE/LOG/LOG_DL4_205	
Global Table Id.:	GBL_DL4_205	
Behaviour Table Id:	LOG_DL4_205	
Name	**Initial Value**	**Final Value**
Resp	FALSE	TRUE
Hyper_DTE	FALSE	TRUE

Figure 3.19 (b) - Test Results Global Variables Table for DL4_205 Test Case

d) Test Results Behaviour Reference Tables

The behaviour reference tables are separated into two groups (i.e. parameter reference tables and timer reference tables), represented in Figure 3.20(a-j) and Figure 3.21 respectively.

PDU References		
PDU:	DISC (Disconnect mode)	
Behaviour Table Id:	LOG_DL4_205	
List Name	**Field Name**	
	A	P
S0	'03'H	'1'B
S1	'03'H	'1'B

(a)

PDU References		
PDU:	SABM (Set Asynchronous Balanced Mode)	
Behaviour Table Id:	LOG_DL4_205	
List Name	**Field Name**	
	A	P
S0	'01'H	'1'B

(b)

PDU References		
PDU:	UA (Unnumbered Acknowledgement)	
Behaviour Table Id:	LOG_DL4_205	
List Name	**Field Name**	
	A	F
S0	'03'H	'1'B
S1	'01'H	'1'B

(c)

Figure 3.20 (a-c) - Test Results Behaviour Reference Tables for DL4_205 Test Case (Parameter Reference Tables for ASP's, and PDU's)

PDU References		
PDU: DM (Disconnected Mode) **Behaviour Table Id:** LOG_DL4_205		
List Name	**Field Name**	
	A	F
S0	'01'H	'0'B

(d)

PDU References						
PDU: I (Information frame) **Behaviour Table Id:** LOG_DL4_205						
List Name	**Field Name**					
	A	P	UD	N(S)	N((R)	LENGTH
S0	'03'H	'0'B	'4142'H	'001'B	'000'B	1080
S1	'01'H	'0'B	'4142'H	'001'B	'000'B	1080

(e)

PDU References									
PDU: FRMR (FRMe Reject) **Behaviour Table Id:** LOG_DL4_205									
List Name	**Field Name**								
	A	F	V(R)	V(S)	C/R	W	X	Y	Z
S0	'03'H	'0'B	'001'B	'001'B	'0'B	R0			
S1	'03'H	'0'B	'001'B	'001'B	'0'B	R1			

(f)

PDU References				
PDU: REF **Behaviour Table Id:** LOG_DL4_205				
List Name	**Field Name**			
	W	X	Y	Z
R0	'1'B	'0'B	'0'B	'0'B
R1	'0'B	'0'B	'0'B	'0'B

(g)

PDU References			
PDU: RRC (Receive Ready Command) **Behaviour Table Id:** LOG_DL4_205			
List Name	**Field Name**		
	AD	F	N(R)
S0	'03'H	'1'B	'001'B

(h)

Figure 3.20 (d-h) - Test Results Behaviour Reference Tables for DL4_205 Test Case (Parameter Reference Tables for ASP's, and PDU's)

PDU References			
PDU: RR (Receive Ready Response)			
Behaviour Table Id: LOG_DL4_205			
List Name	**Field Name**		
	AD	F	N(R)
S0	'03'H	'1'B	'001'B

(i)

PDU References	
PDU: HEX (Hex string in frame)	
Behaviour Table Id: LOG_DL4_205	
List Name	**Field Name**
	Hstring
S0	'01FF'H

(j)

Figure 3.20 (i-j) - Test Results Behaviour Reference Tables for DL4_205 Test Case (Parameter Reference Tables for ASP's, and PDU's)

Timer References		
Timer: T1		
Behaviour Table Id: LOG_DL4_205		
List Name	**Field Name**	
	Start Time	**End Time**
S0	15:48:35.79	15:48:35.99
S1	15:48:36.23	15:48:38.52
S2	15:48:39.42	15:48:41.01
S3	15:48:41.33	15:48:41.80
S4	15:48:42.90	15:48:43.45
S5	15:48:44.05	15:48:44.65
S6	15:48.44.89	15:48.45.26

Figure 3.21 - Test Results Behaviour Reference Table for DL4_205 Test Case (Timer Reference Table)

From this detailed example, the following points should now be clear to the reader:

i) Events involving explicit timer references (send with timeouts) are explicitly time-stamped in the ATR timer reference table for each test case.

ii) Important segments of test results behaviour is highlighted using the segment columns (e.g. preamble, test body, postamble, repeat sequences etc.)

iii) ATS event location identifies exactly which parts of the ATS is referenced in the test results behaviour (In chapter 4 we show the usefulness of such information while investigating the relationships between different test result behaviours).

iv) Parameter settings and control codes for all the PDU's or ASP's, sent or received, are recorded in an informative way.

v) Finally, in general it can be seen that test results are more readable when they are organized in this manner.

Thus, ATR's provide a complete, auditable test execution report in a format which lends itself to various types of automated results analysis, discussed in the next chapter. Moreover, the derivation of ATR's from ETR's is straightforward, and can be automated (in a test system- dependent manner, of course).

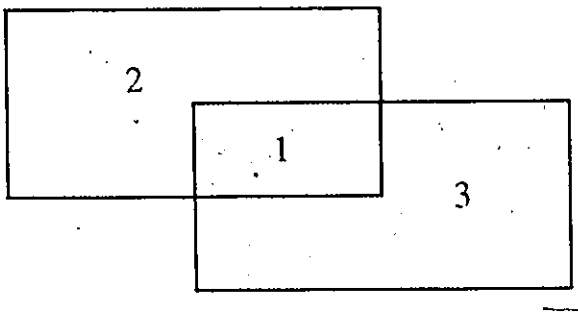
Chapter 4

High Level Test Result Analysis

4.1 Objectives

In section 2.2 we stated the standards view of test results analysis. The common view of test results analysis varies somewhat from the notion presented in the ISO draft proposal [ISO9646-1]. In particular, common view of test results analysis has a broader concept of auditing test results. It includes a detailed study of the testcases declared non-conforming to ensure that their failure is not due to existence of errors in the test specification, test environment or the protocol specification, itself.

ISO, of course, is not so concerned with diagnosis of test results, but rather concentrates on validation of test results in terms of repeatability and comparability of test results between test laboratories. Figure 4.1 provides an overview of the variety of activities and interests in the area of test results analysis.



1 - Comparing foreseen outcomes with observed outcomes (Common aspects of test results analysis) in order to detect non-conformance.

2 - Cross validating protocol specifications, test specifications and test environment (i.e. with a broader (industry) definition of auditability).

3 - Validation of test results by means of repeatability of results or comparability of results (i.e. standards definition of auditability - simple walkthrough of the results)

FIGURE 4.1 - Overall View of Test Results Analysis

It is the overall view of test results analysis which we are considering in this chapter. We do not necessarily claim that the whole process of test results analysis could be automated. Nevertheless, our primary objective is to show the usefulness of our approach in development of tools for automating some of these steps in test results analysis.

In the remainder of this chapter, we primarily define a framework for test results analysis. Further, we introduce methods for verifying ATR's against protocol specifications and introduce some graphical models for diagnosing test results in a semi-automated manner.

4.2 High Level Test Results Analysis Framework

Test results analysis is currently conducted in an adhoc manner. Analysis is always initiated at a low level (see section 2.4). It is therefore useful to identify some of the critical steps in analysis of test results and introduce a framework which may be followed at a higher level. This is accomplished in this section.

In high level test results analysis, the analysis takes place at the test specification level (ATS) rather than at the test execution level. The ATR's are derived (from ETR's according to the procedure given in Chapter 3) and verified using the standard abstract test suite. The ATR's are very suitable for cross validation against formalized protocol specifications (section 4.3); therefore high level analysis is more accurate. At this level all

the related information for the test environment is contained in the PICS and PIXIT. Also as observed in Chapter 3, the format of the conformance log in ATR's is very similar to the draft standard TTCN test specification language. This facilitates the relating of ATR's to the ATS.

In ATR's, test result verdicts are "pass", "fail" or "inconclusive". The ATR's contain the observed interaction sequences for each test case, which can be mapped onto a sequence of events in the abstract test case (ATC) specification. The verdicts assigned to these observed ATR outcomes are those associated with the matching foreseen outcomes in the ATC. If the observed outcome is not specified in the ATC, then the ATS will state what default verdict should be assigned.

In the test results behaviour tables of ATR's, where the observed outcomes are recorded, the preamble, test body and postamble segments have been explicitly distinguished. The "fail" and "inconclusive" verdicts may be assigned in any of these three test case segments; only if all three components complete without incident can a verdict of pass be assigned. "Fail" and "inconclusive" verdicts can be further qualified numerically (1, 2, 3 respectively) to indicate whether the exception occurred in the preamble, test body, or postamble. This facilitates detailed analysis as discussed in section 4.4.

As is done in practice for ETR's [Prob88b], pass verdicts will not be analyzed. Only the "fail" and "inconclusive" verdicts are analyzed to verify that the verdict is assigned legitimately. Also, some actual behaviours may not have been explicitly foreseen. Such **unforeseen** behaviours are usually captured by an ?OTHERWISE statement in TTCN; in these cases some consistency checks are made and the result of the investigations is recorded in the "unforeseen" column of the PCTR of ATR's [ISO9646-5].

Once the ATR's are derived, data summarization reports can be obtained (section 4.4) for stepwise auditability and diagnosis of test results. Test result inter dependency reports may be created automatically from the exception ATR (an abbreviated ATR which contains a record of only those test cases which were executed and terminated with a "fail" or "inconclusive" verdict). The relations between all the failed and inconclusive tests can then be displayed graphically and studied. Another source for diagnostic data is cross reference charts, where the exact location of an error may be indicated. These charts and reports can be generated automatically from the ATR's. The test analyst may study this data and estimate the complexity of the problem, before even looking at detailed test results behaviours. Often, solutions to major problems can be identified at this stage; others may require a more detailed study of the conformance log. One important potential benefit of course is that it can serve as a productivity aid for the test analyst.

In order to validate conformance tests using the same ATS at different test sites, the respective ATR's can be compared to check for consistency. The machine processable ATR's facilitate porting between test laboratories for comparability. The comparing of ATR's may be performed in two steps. Initially the PCTR's may be compared to check if exactly the same verdicts are assigned for different test cases. This however would not be a complete comparison, as fail verdicts in different ATR's for the same test case may have been caused by different observed behaviours. Hence, the second step of comparing the test results behaviours becomes a necessity. These possibilities should also be considered by ISO while developing standards for comparing test results reports. Thus, using ATR's to represent the test results can greatly facilitate the development of tools to automate these types of test result analysis activities.

So far by defining a framework, we have identified some of the main activities in high level test results analysis. We believe that ATR's can serve as a basis for semi-automatically pursuing some of these activities. In the rest of this chapter we propose methods for carrying out some of the above identified activities.

4.3 Theoretical Results : Verification of ATR's Against Protocol Specifications

In section 3.3 a high-level notation for representation of test results was introduced (TTCN-TR). One of the purposes of such a representation was to be able to directly relate the test results to the specification of the protocol. Specifications may be defined using Formal Description Techniques (FDT's). Using FDT's, desired behaviours of the system can be expressed using formal syntax and semantics. FDT's are specially important for analyzing test results, since they are complete, consistent, precise, concise and unambiguous. There are three FDT's under consideration for standardization, LOTOS (Language Of Temporal Ordering Specification) [ISO8807, Brink87], ESTELLE [ISO9074] and SDL (Specification and Description Language) [SDL85].

A protocol specification may contain non-observable behaviours. But, the log behaviours in the ATR's only contain one of the trace sequences from the test case specification (in TTCN), therefore the non-observable behaviours (i.e. transient transitions) would not be represented in the ATR's. Our approach can be extended in the future to verify such non-observable traces which normally exist in the protocol specifications.

In section 4.3.1 we show how observable behaviours of protocols expressed as FSM's are representable as ATR's expressed in TTCN-TR notation. This one to one

correspondence between test results and the behaviours of the protocol is essential for validating ATR's. In section 4.3.2 we extend our claim and show the TTCN-TR coverage of Normal Form Specifications (NFS's) and extended finite state machines given in ESTELLE [ISO9074]. In section 4.3.3 we discuss how this cohesion helps in analyzing test results, specially when there is a need to relate test results to the original test purposes in the formal specification.

4.3.1 ATR's and Finite State Machines (FSM's)

Finite State Machines (FSM's) are widely used in formal specification of protocols. In order to demonstrate the usefulness of TTCN-TR in representing test results, we need to show how any trace of a protocol FSM specification can be mapped onto a test case result in TTCN-TR contained in the ATR's (i.e. the captured trace). To do this, we first define the particular semantics we require to be associated with FSM's, FSM traces, and behaviour descriptions of ATR's. Then, we state and prove the main theorem, namely "Any trace (sequences of transitions) of a given protocol FSM can be represented in ATR notation". Finally, we present an example to illustrate the main concepts.

For FSM's the following is defined:

Definition 4.1: A **Finite State Machine** M is a quintuple M = (S, I, O, δ, λ), where:

- S is a finite, non empty set of states, $S = \{s_1, s_2, \ldots, s_{m1}\}$.
- I is a finite, non empty set of inputs, $I = \{i_1, i_2, \ldots, i_{m2}\}$.
- O is a finite, non empty set of outputs, $O = \{o_1, o_2, \ldots, o_{m3}\}$.
- $\delta : I \times S \text{ -----> } S$ is the state transition function.
- λ is the output function such that:

$$\lambda : I \times S \quad \text{-----> } O$$

(Note: This is a Mealy machine model. Moore machines are not realistic for specifying protocols, since the protocol output is always associated with both the state and the input. Also, we assume that input and output interactions can be identified as belonging to either the upper or lower (test) interface)

Definition 4.2: In any protocol FSM, a **trace (T)** is a sequence of state transitions which starts at a state s_1 and terminates at a state s_n. Such a sequence is represented by (t_1, t_2, ... , t_k, ... , t_{n-1}), $n \geq 2$, where t_1 is a transition defined at state s_1 and t_{n-1} takes the FSM to state s_n.

Definition 4.3: An Abstract Test Results report(ATR) consists of a summary type of report which the standard denotes as a Protocol Conformance Test Report (PCTR), and a more extended report often referred to as a **conformance results log** which records all interactions between the Implementation Under Test (IUT) and the complete testing environment. The **conformance results log description R** within the test results behaviour description for a single test case is formally defined as **R** = {**r** | **r** is a results log **entry**}, where each entry consists of the following five components:

a) DESCRIPTION[r], b) ENUM[r], c) LOCATION[r], d) START-END[r],
e) REFERENCE [r]

These components of an R entry r are defined as follows ([ ] in the statements means may or may not be present):

a) **DESCRIPTION[r]** (log behaviour description) is one event line consisting of one or in some cases more (e.g. a send or receive with a boolean expression) of the following statements:
   - The test case name or test step name, $name[y_1, \ldots, y_m][x_1, \ldots, x_n]$, where, x_i, y_j represent actual and PCO parameters respectively.
   - Send Events, $[x_1, \ldots, x_n]$, followed by *"!"* and data identification, d_k, where d_k belongs to a set $DAT = \{d_1, \ldots, d_n\}$.
     Receive Events, $[x_1, \ldots, x_n]$, followed by *"?"* and data identification d_k.
     It is possible to have some predicates (i.e. Boolean Expressions) and assignment statements following the above.
   - Timer operations, $to(t_i)$, where t_i belongs to a set $TM = \{t_1, \ldots, t_n\}$ of timer identifiers.
   - Boolean Expressions $(b_1, \ldots, b_n)$ or Assignment statements, $(a_1, \ldots, a_n)$.

b) **ENUM[r]** is either nil or an identification character (e.g. '*') together with an integer value (z) (e.g. Can be used to note repetition of blocks of entries, as illustrated in Figures 3.6 and 3.8 of Chapter 3).

c) **LOCATION[r]** represents the test case name or test step name at which DESCRIPTION[r] is referenced (e.g. In figure 4.2(b), LOCATION for "L! a" is STATE A).

d) **START-END[r]** is either nil or an identification for recognizing the start and end of a block of entries (r_1, ..., r_n)

e) **REFERENCE[r]** is a reference for d_k in the send and receive events, or for t_j for timer operations.

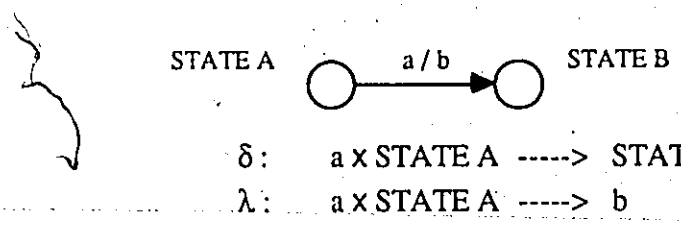


Figure 4.2(a)- An Example of a State Transition in a Specification

Test Results Behaviour						
Reference :	ATR/LOG/EXAMPLE					
Identifier :	EXAMPLE					
Purpose :	To exercise an observable transition (a send and a receive)					
Log Behaviour Description		E	ATS event Location	Segments Start	End	Behaviour References
MAIN-TREE[L,U] U! a L? b MAIN-TREE[L,U]			 STATE A STATE A	TS TB	 TB TE	 a(S0) b(S0)
Test Case Start Time :	12:00:00.00					
Test Case End Time :	12:00:00.30					
Num of Retries :	0					
Verdict	PASS					

Figure 4.2(b)- An Example of a Test Results Behaviour Description in TTCN-TR

Theorem 4.1: Any trace (sequences of transitions) of a given protocol FSM can be represented in ATR notation (TTCN-TR).
(Note that we are proving this in order to show that TTCN-TR is a complete notation for this kind of trace analysis. Moreover, from the proof it can be seen that mechanisms in TTCN-TR are simple, as well as sufficient.)

Proof: We try to prove this by induction. As the basis to our proof, we have to show that TTCN-TR can cover an FSM trace of length 1 (e.g. t_1). We show this by examining all possible cases.
For all '-' (unobservable) inputs and outputs (Figure 4.3) the test result behaviour of the ATR does not show anything (i.e., behaviours are unobservable), simply because no observable data is exchanged (i.e., These may correspond to transient transitions contained in the implementations). For each input and/or output present in a transition, the DESCRIPTION[r] would contain a send and/or a receive statement respectively. Also, the (start) state at which the transition was initiated should be mapped onto the LOCATION[r]. (e.g., the transition associated with "L! a" is mapped onto STATE A in Figure 4.2(b)). Thus, the behaviour description for a single transition contains a tree name, upto two event

lines corresponding to the events (sent and/or received) associated with the transitions input and output respectively, and values correponding to event lines (i.e., ATS event Location, Behaviour References, etc.). Figure 4.2(b) represents the transition in Figure 4.2(a).

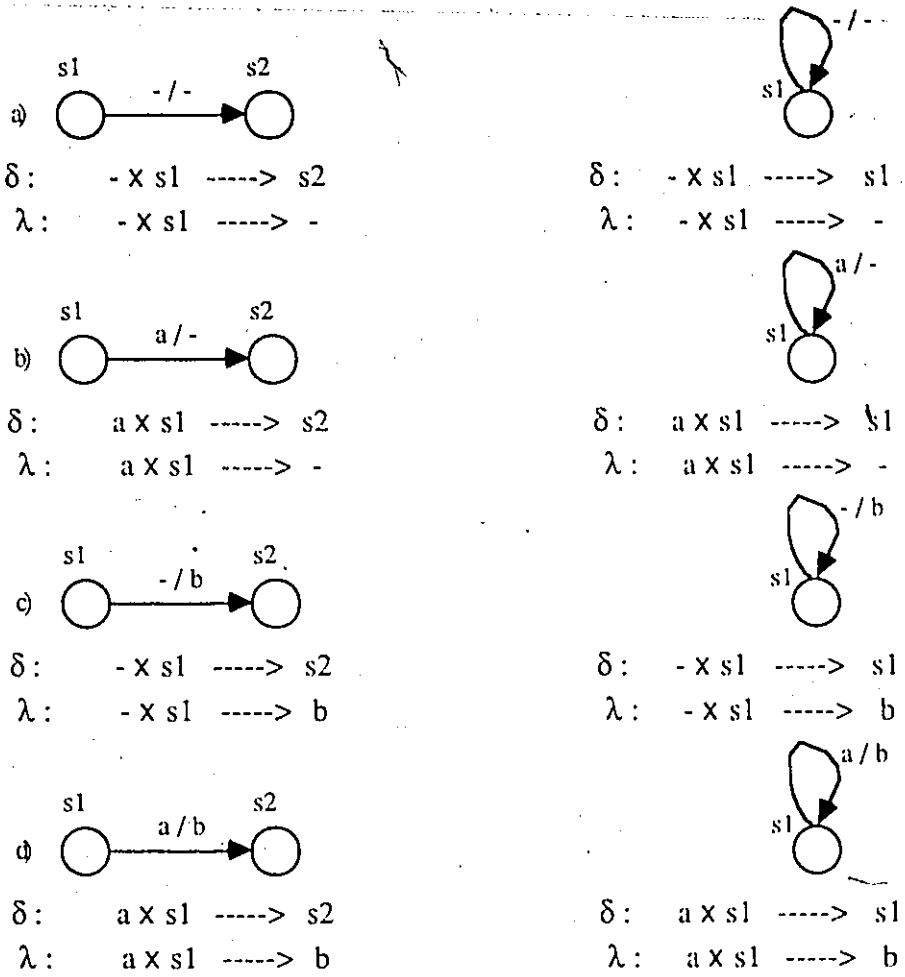


FIGURE 4.3 - Possible Scenarios for Traces of Length 1 Transition

Now for **induction**, we have to prove this is true for traces of length k ($k \geq 2$) assuming it is true for traces of length k-1. By proving that t_k transition is also representable in TTCN-TR, we will prove that all transitions in a trace can also be adequately represented in TTCN-TR.

We call the states connected by t_k transition (s_{k1} and s_{k2}). Above we have proved that a length one trace consisting of a single transition $t_1 : s_1$ -----> s_2 or $t_1 : s_1$ -----> s_1 can be

represented. Now we need to consider how the trace of length k was constructed from one of length k-1. This trace contains an additional transition, say $t_k : s_{k1}$ -----> s_{k2}, where we may have two conditions, k1<k2 (going to a new state) and k1=k2 (going to the same state) which are handled exactly the same way as above. In particular, the start state of t_k will be shown in LOCATION field and will be the target state of t_{k-1}. The additional case involves, k1>k2 (going to a state already visited). In a way similar to the above discussion, the observable interactions are encoded as additional event lines in the ATR behaviour description.

Moreover, in TTCN-TR, where there are transition cycles in the FSM trace (i.e. regular expressions such as $(t_1t_2t_3)^*$), a counter can be used to keep track of the number of times such cycles are repeated, using NUM[r] and BEGIN-END[r] in the log description R.

Some FSM specifications of protocols can be specified using classes of predicates [CaSi86]. These also can easily be represented using the boolean expressions within DESCRIPTION[r] in a natural way.

We have shown that TTCN-TR covers transition traces of protocol FSM's. The FSM's have no mechanisms in the their basic forms through which we could specify synchronization, default behaviours, etc. Therefore it is unnecessary to map such features. Hence any trace of the FSM's can be represented in TTCN-TR.

Example 4.1: Here we demonstrate the mapping of FSM's onto ATR's represented in TTCN-TR notation. The example uses the transport layer class(0) FSM specification in Figure 4.4 [ISO8073, Ural87]. The selected trace of the FSM (equivalent to the test result behaviour in the ATR) uses underline to represent PDU's and ASP's associated with the transitions. In order to be more specific in terms of what is sent or received at each the testers (lower and upper tester), we represent our results in the context of a local test method configuration. Figure 1.2(a) in Chapter 1 provides details of the local test architecture. A sample ATR which would be generated while testing an implementation of such a specification is given in Figure 4.5 (the PCTR is similar to the one presented in section 3.4.2, and is omitted here. Other features of TTCN-TR are not needed for this example but may be included in the real ATR, such as the time stamps of test results behaviours.

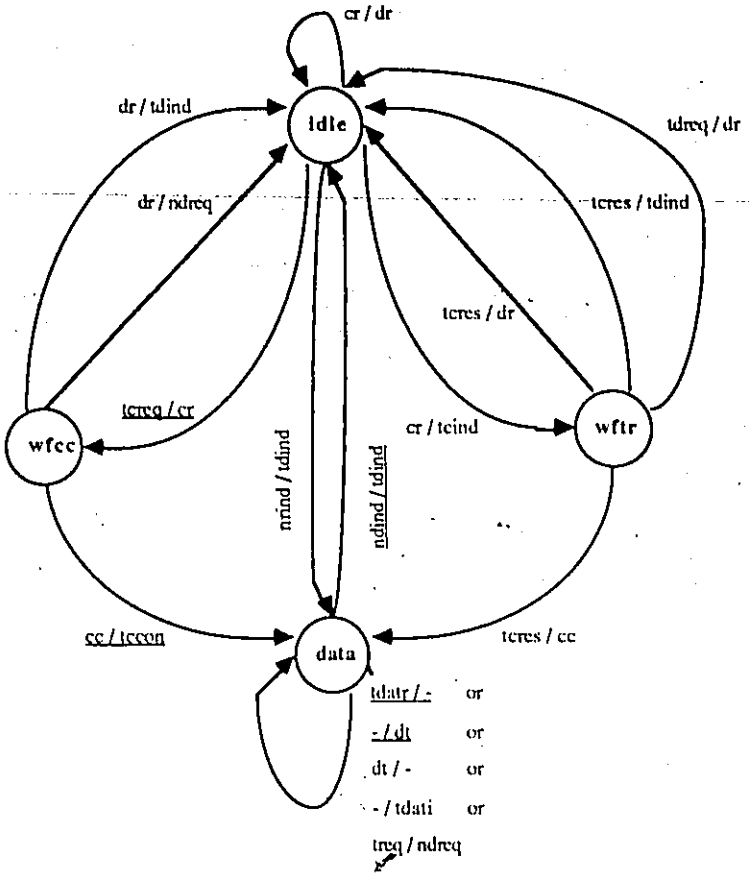


Figure 4.4 - An FSM Specification of a Class(0) Transport Layer Protocol

Test Results Behaviour						
Reference :	TRANSPORT/CLASS0/LOG/EXAMPLE					
Identifier :	LOG_EXAMPLE					
Purpose :	Connection establishment					
Log Behaviour Description		**E**	**ATS event Location**	**Segments Start**	**End**	**Behaviour References**
MAIN-TREE [L,U]				TS		
UT! TCREQ				TB		
LT? CR						
LT! CC						
UT? TCCON						
UT! TDATR						
LT? DT						
LT! NDIND						
UT? TDIND					TB	
MAIN-TREE [L,U]					TE	
Test Case Start Time :	12:00:00.00					
Test Case End Time :	12:00:00.30					
Num of Retries :	0					
Verdict	PASS					

Figure 4.5 - ATR of a Trace in Class(0) Transport Layer Protocol FSM Specification

It can be seen from this example that using ATR's simplifies verification of test results when FSM's are used for specifying protocols.

□

4.3.2 ATR's and Normal Form Specifications (NFS's)

Since FSM's have limited capabilities for representing specifications of protocols, we now extend our theorem to prove that a similar mapping can be made from traces of Normal Form Specifications [Sarik85] onto ATR's. We assume protocol specifications are given in ESTELLE (Extended State Transition Language [ISO9074]). The usefulness of such a mapping can be observed in section 4.3.3, where we show how to relate test results to test purposes using formal specifications. Following the same structure as section 4.3.1, we first define NFS's and traces of NFS's.

In Estelle [ISO9074], a protocol may be specified in terms of the state space and the possible state transitions of the module. In general, protocol specifications in Estelle may include intermodule interactions and relatively complex transition definitions. Various transformations proposed in [Sarik85] eliminate intermodule interactions and simplify transition definitions in order to obtain easy-to-analyze, single-module specifications (i.e. Normal Form Specifications) from the original specifications given in Estelle [Ural87]. NFS is formally defined as follows [Ural87]:

Definition 4.4: A **Normal Form Specification** (NFS) describes the externally observable behaviour of a module (e.g., a protocol entity) in terms of **Normal Form Transitions (NFT)**. Formally, a NFS **T** is defined as **T** = {t | t is a NFT} where each NFT t consists of the following five components:

a) WHEN[t], b) FROM[t], c) PROVIDED[t], d) TO[t] e) BEGIN-END[t]

The components of an NFT t are defined as follows:

a) Let **WHEN[t]** be nil for a spontaneous t and let **WHEN[t]** for a t that is not spontaneous be an input interaction i which is in the form $ii\ (X_1, \ldots, X_n)$, where ii stands for interaction identifier and $X_1, \ldots, X_n$ ($n \geq 0$) are interaction parameters. Then, for a given NFS T, the set of input interactions I is defined as $I = \cup_{t \in T}\{WHEN[t]\}$.

b), d) Let **FROM[t]** and **TO[t]** be present state and the next state of a t, respectively. Then, for a given NFS **T**, the set of states **S** is defined as $S = \cup_{t \in T}\{FROM[t], TO[t]\}$. It is assumed that $idle \in S$ and that there exist at least one t_1 and one $t_2 \in S$ such that $FROM[t_1] = idle$, $TO[t_2] = idle$, and $t_1 \neq t_2$.

c) Let **PROVIDED[t]** be either nil or an *n-ary* predicate $p(X_1, \ldots, X_n)$, where $X_1, \ldots, X_n$ $(n > 0)$ are variables.

e) Let **BEGIN-END** [t] be a sequentially ordered statement block b = ($d_1, \ldots, d_m, o_1, \ldots, o_n$), where:

- d_i, $i = 1, \ldots, m$, $m \geq 0$, is an assignment statement a or a procedure call c.
- o_j, $j = 1, \ldots, n$, $n \geq 0$, is an output statement o.

- An assignment statement a is in the form $Y := f(X_1, \ldots, X_n)$, where $n \geq 0$ and $Y, X_1, \ldots, X_n$ are variables.
- An output statement o is in the form out $ii(X_1, \ldots, X_n)$, where out indicates that the statement is an output statement. *ii* stands for interaction identifier, and $X_1, \ldots, X_n$ $(n \geq 0)$ are interaction parameters.
- A procedure call c is in the form $pn(X_1, \ldots X_k, Y_1, \ldots, Y_m, Z_1, \ldots, Z_n)$, where pn is the procedure name and $X_1, \ldots, X_k$ $(k \geq 0)$; $Y_1, \ldots, Y_m$ $(m \geq 0)$; and $Z_1, \ldots, Z_n$ $(n \geq 0)$ are variables that stand for actual in-out, output, and input parameters, respectively.

Definition 4.5: In any NFS, a **trace (T)** is a sequence of NFT's which starts at a state s_1 and terminates at a state s_n. Such a sequence is represented by $(t_1, t_2, \ldots, t_k, \ldots, t_{n-1})$, $n \geq 2$, where s_1 is FROM[t] state of t_1 and s_n is TO[t] state of t_{n-1} and for all the intermediate t's such as t_k, TO[t] state of t_k is the same as FROM[t] state of t_{k+1}.

Theorem 4.2: Any observable part of a trace (sequences of NFT's (Normal Form Transitions)) of a given NFS can be represented in ATR notation (TTCN-TR). In fact, non-observable transitions can be represented as well, but usually can not be directly monitored in practical. Further discussions are given in Chapter 5.
(Note that we are proving this in order to show that TTCN-TR is a complete notation for this kind of trace analysis. Moreover, from the proof it can be seen that mechanisms in TTCN-TR are simple, as well as sufficient.)

Proof: As in theorem 4.1, this can be proved using proof by induction on length of traces. The only essential difference between representing NFS traces and FSM traces by ATR is the additional requirement to handling enabling transition predicates (in PROVIDED

clauses). As a **basis** to our proof, we have to show that TTCN-TR can represent an NFS trace of length 1 (i.e. a trace consisting of only one NFT t, e.g., t_1). The following accounts for all the possibilities for representing a complete single NFT trace in TTCN-TR:

1) t is not a spontaneous transition:
   a) Representation of WHEN[t]:
      WHEN[t] can be represented by a send event statement in the DESCRIPTION[r] to represent the sending of a message from either the upper tester or the lower tester. The interaction parametersfor the event statement can be specified using REFERENCE[r].
   b) Representation of FROM[t]:
      FROM[t] construct may be directly mapped to LOCATION[r] of send statements of DESCRIPTION[r]. However, for this to be true each NFT should be specified separately while specifying test cases. Otherwise, the mapping between results and specification would not be consistent.
   c) TO[t] construct is not represented in the ATR's specifically. It is always included in the information of the following transition (i.e. FROM[t+1], where t+1 is the next observable transition).
   d) PROVIDED[t] could be nil in which case test result behaviour does not show anything additional. Otherwise, a corresponding boolean expression is inserted into the event line following the send event description in DESCRIPTION[r].
   e) In BEGIN-END[t] the output statements (if existed) are represented by receive events of DESCRIPTION[r] in a r. If output statements do not exist (i.e. output is unobservable), then nothing is shown in the ATR results behaviour for t. Assignment statements which set up predicates for future events to occur may also be presented by DESCRIPTION[r].
2) t is a spontaneous transition:
   WHEN[t] for a spontaneous t is nil. Therefore it would not be represented in the ATR. However, the other components of the transition, if observable, are represented exactly the same as for non-spontaneous transitions (e.g. The output statements may still be treated the same as receive events in the ATR's)

It can be seen that a single NFT t_1 can be mapped onto its equivalent ATR behaviours. Also, for t_1 FROM[t] may be considered the same as s_1 in the FSM specification. Similarly, TO[t] may be considered as s_2 in the FSM specification. Therefore, the possibilities are that $s_1=s_2$ (going to same state) or $s_1<s_2$ (going to a new state). In theorem

4.1, we already proved that ATR's can represent observable behaviours for both of the above behaviours.

Above we have proved that single NFT t_1 can be represented in ATR's (similar to the FSM proof). Now for **induction**, we have to prove this is true for traces of length k ($k \geq 2$) given it is true for traces of length k-1. As we prove that t_k NFT is also representable in TTCN-TR, we will prove that any trace of an NFS specification an be adequately represented in TTCN-TR.

Above we have proved the cases where FROM[t]<TO[t] (going to a new state) and FROM[t]=TO[t] (going to the same state). Here we have an additional case where FROM[t]>TO[t] (going to a state already visited). This is handled exactly the same as above. Moreover, in TTCN-TR in such cases where there are NFT cycles (i.e. expressions such as $(t_1t_2t_3)^*$), a counter can be used to keep track of the number of times we go through such NFT cycles. This is represented using Num[r] and START-END[r] in the R. Hence observable parts of any trace in normal form specifications can also be represented in TTCN-TR.

Example 4.2: Given the normal form specification of transport layer class(0) (Figure 4.6), we select a trace and represent the possible ATR (test results behaviour and test results behaviour reference tables) which may result from testing an implementation of such a specification (Figure 4.7). In order to be more specific in terms of what is sent or received at each of the testers (lower and upper tester), we represent our results in the context of a local test method configuration. Figure 1.2(a) in Chapter 1 provides details of the local test architecture. The selected trace is the sequence t_1, t_6, t_{13}, t_{15}, t_{18}. The PCTR is similar to the one presented in section 3.4.2, therefore it is not presented in this example. Test results parameter tables are not needed here.

Here we briefly describe how the chosen trace in the NFS specification can be reconstructed from the ATR fragments in Figure 4.7. Figure 4.7 does not represent a full ATR (for example it does not have the information for time stamps. Since the specification does not have time constraints these information are ignored in this example). The trace starts with t_1 which sends a connection request from the upper tester to the IUT which has a boolean expression associated with it (same as the PROVIDED[t] predicate in the NFS). The detail of the parameters belonging to this ASP is recorded in a separate table, which is

used to record all the captured values of the parameters of TCreq ASP. Later the lower tester receives a CR (connection request) PDU which also has some parameters associated with it, also recorded in a separate table for referencing. Same thing is done for t_6 and t_{18}, however t_{13} and t_{15} are mapped differently because they contain an unobservable receive and send event respectively. It can be seen from the ATR that such unobservable behaviours are omitted. It would be useful if ATR's could represent such behaviours as well, however this is one of our suggestions for extension of our approach in representing test results, which is outside the scope of this thesis. Hence, from the example the ability of the ATR's to represent the observable parts of NFS traces can be easily observed.

```
WHEN tcreq(to.t.addr, from.t.addr, qts.req)
FROM idle
PROVIDED tcreq.in.qts.req = ok
TO WFCC
t1:     BEGIN
        local.refer := ...;
        tpdu.size := ...;
        calling.t.addr := tcreq.in.from.t.addr;
        called.t.addr := tcreq.in.to.addr;
        cr.out.source.ref := local.refer;
        cr.out.option := 'normal';
        cr.out.calling.t.addr := calling.t.addr;
        cr.out.called.t.addr := called.t.addr;
        cr.out.max.tpdu.size := tpdu.size;
        out cr(source.ref, option, calling.t.addr,
                called.t.addr, max.tpdu.size);
        END;
----------
WHEN tcreq(to.t.addr, from.t.addr, qts.req)
FROM idle
PROVIDED tcreq.in.qts.req <> ok
TO idle
t2:     BEGIN
        tdind.out.ts.disc.reason := ...;
        tdind.out.ts.user.reason := ...;
        out tdind(ts.disc.reason, ts.user.reason);
        END;
----------
WHEN cr(source.ref, option, calling.t.addr,
        called.t.addr, max.tpdu.size)
FROM idle
PROVIDED cr.in.max.tpdu.size = nil
        and cr.in.option = ok
TO wftr
t3:     BEGIN
        remote.refer := cr.in.source.ref;
        tpdu.size := cr.in.max.tpdu.size;
        calling.t.addr := cr.in.calling.t.addr;
        called.t.addr := cr.in.called.t.addr;
        qts.estimate := ...;
        tcind.out.to.t.addr := called.t.addr;
        tcind.out.from.t.addr := calling.t.addr;
        tcind.out.qts.pro := qts.estimate;
        out tcind(to.t.addr, from.t.addr, qts.pro);
        END;
----------
WHEN cr(source.ref, option, calling.t.addr,
        called.t.addr, max.tpdu.size)
FROM idle
PROVIDED cr.in.max.tpdu.size <> nil
        and cr.in.option = ok
TO wftr
t4:     BEGIN
        remote.refer := cr.in.source.ref;
        tpdu.size := ...;
        calling.t.addr := cr.in.calling.t.addr;
        called.t.addr := cr.in.called.t.addr;
        qts.estimate := ...;
        tcind.out.to.t.addr := called.addr;
        tcind.out.from.t.addr := calling.t.addr;
        tcind.out.qts.pro := qts.estimate;
        out tcind(to.t.addr, from.t.addr, qts.pro);
        END;
----------
WHEN cr(source.ref, option, calling.t.addr,
        called.t.addr, max.tpdu.size)
FROM idle
PROVIDED cr.in.option <> ok
TO idle
t5:     BEGIN
        dr.out.dest.refer := cr.in.source.ref;
        dr.out.disconnect.reason := ...;
        out dr(dest.refer, disconnect.reason);
        END;
----------
WHEN cc(max.tpdu.size)
FROM wfcc
PROVIDED cc.in.max.tpdu.size <> nil
TO data
t6:     BEGIN
        qts.estimate := ...;
        tccon.out.qts.res := qts.estimate;
        in.buffer := nil;
        out.buffer := nil;
        out tccon(qts.res);
        END;
----------
WHEN cc(max.tpdu.size)
FROM wfcc
PROVIDED cc.in.max.tpdu.size = nil
TO data
t7:     BEGIN
        qts.estimate := ...;
        in.buffer := nil;
        out.buffer := nil;
        tccon.out.qts.res := qts.estimate;
        out tccon(qts.res);
        END;
----------
WHEN dr(disconnect.reason, add.clear.reason)
FROM wfcc
PROVIDED dr.in.disconnect.reason = "user.init"
TO idle
t8:     BEGIN
        ndreq.out.disc.reason := dr.in.disconnect.
                                 reason;
        tdind.out.ts.disc.reason := dr.in.disconnect.
                                 reason;
        tdind.out.ts.user.reason := dr.in.add.clear.
                                 reason;
        out ndreq(disc.reason);
        out tdind(ts.user.reason, ts.disc.reason);
        END;
----------
WHEN dr(disconnect.reason, add.clear.reason)
FROM wfcc
PROVIDED dr.in.disconnect.reason = "user.init"
TO idle
```

Figure 4.6 - NFS for OSI Class (0) Transport Protocol

```
t9:     BEGIN
        ndreq.out.disc.reason := dr.in.disconnect.
                                 reason;
        tdind.out.ts.disc.reason := dr.in.disconnect.
                                 reason;
        out ndreq(disc.reason);
        out tdind(ts.disc.reason);
        END;
----------
WHEN tcres(qts.req)
FROM wftr
PROVIDED tcres.in.qts.req <= qts.estimate
TO data
t10:    BEGIN
        local.refer := ...;
        cc.out.dest.refer := remote.refer;
        cc.out.source.ref := local.refer;
        cc.out.calling.t.addr := calling.t.addr;
        cc.out.called.t.addr := called.t.addr;
        cc.out.max.tpdu.size := tpdu.size;
        in.buffer := nil;
        out.buffer := nil;
        out cc(dest.refer, source.ref, calling.t.addr,
                called.t.addr, max.tpdu.size);
        END;
----------
WHEN tcres(qts.req)
FROM wftr
PROVIDED tcres(qts.req)
TO idle
t11:    BEGIN
        dr.out.dest.refer := remote.refer;
        dr.out.disconnect.reason := ...;
        dr.out.add.clear.reason := ...;
        tdind.in.ts.disc.reason := ...;
        out dr(dest.refer, disconnect.reason,
                add.clear.reason);
        out tdind(ts.disc.reason);
        END;
----------
WHEN tdreq(ts.user.reason)
FROM wftr
TO idle
t12:    BEGIN
        dr.out.disconnect.reason := ...;
        dr.out.add.clear.reason := tdreq.in.ts.user.
                                 reason;
        dr.out.dest.refer := remote.refer;
        out dr(dest.refer, disconnect.reason,
                add.clear.reason);
        END;
----------
WHEN tdatr(tsdu.fragment)
FROM data
TO data
t13:    BEGIN
        insert(out.buffer, tdatr.in.tsdu.fragment);
        END;
----------
FROM data
PROVIDED out.buffer <> nil
TO data
t14:    BEGIN
        remove(out.buffer, dt.out.user.data);
        out dt(user.data);
        END;
----------
WHEN dt(user.data)
FROM data
TO data
t15:    BEGIN
        insert(in.buffer, dt.in.user.data);
        END;
----------
FROM data
PROVIDED in.buffer <> nil
TO data
t16:    BEGIN
        remove(in.buffer, tdati.out.tsdu:fragment);
        out tdata(tsdu.fragment);
        END;
----------
WHEN tdreq(ts.user.reason)
FROM data
TO idle
t17:    BEGIN
        ndreq.out.disc.reason := tdreq.in.ts.user.
                                 reason;
        out ndreq(disc.reason);
        END;
----------
WHEN ndind()
FROM data
TO idle
t18:    BEGIN
        tdind.out.ts.disc.reason := ...;
        out tdind(ts.disc.reason);
        END;
----------
WHEN nrind()
FROM data
TO idle
t19:    BEGIN
        tdind.out.ts.disc.reason := ...;
        out tdind(ts.disc.reason);
        END;
```

Figure 4.6 (Contin.) - NFS for OSI Class (0) Transport Protocol

Test Results Behaviour					
Reference :	TRANSPORT/CLASS0/LOG/EXAMPLE				
Identifier :	LOG_EXAMPLE				
Purpose :	Connection establishment				
Log Behaviour Description	**E**	**ATS event Location**	**Segments Start**	**End**	**Behaviour References**
MAIN-TREE [L,U]			TS		
UT! TCREQ [tcreq.in.qts.req = ok]			TB		TCREQ(S0)
LT? CR					CR(S0)
LT! CC [cc.in.max.tpdu.size ≠ nil]					CC(S0)
UT? TCCON					TCCON(S0)
UT! TDATR					TDATR(S0)
LT? DT					DT(S0)
LT! NDIND					NDIND(S0)
UT? TDIND				TB	TDIND(S0)
MAIN-TREE [L,U]				TE	
Test Case Start Time :	12:00:00.00				
Test Case End Time :	12:00:00.30				
Num of Retries :	0				
Verdict	PASS				

(a)

PDU References								
PDU:	CR (Connection Request)							
Behaviour Table Id:	LOG_EXAMPLE							
List Name	**Field Name**							
	LI	CDT	TSAP	DST	SRC	DT-NR	ED-NR	YR-NR
S0	<observed values are printed for each of these PDU parameters>							

(b)

PDU References								
PDU:	CC (Connection Confirm)							
Behaviour Table Id:	LOG_EXAMPLE							
List Name	**Field Name**							
	LI	CDT	TSAP	DST	SRC	DT-NR	ED-NR	YR-NR
S0	<observed values are printed for each of these PDU parameters>							

(c)

PDU References								
PDU:	DT (Data Transfer)							
Behaviour Table Id:	LOG_EXAMPLE							
List Name	**Field Name**							
	LI	CDT	TSAP	DST	SRC	DT-NR	ED-NR	YR-NR
S0	<observed values are printed for each of these PDU parameters>							

(d)

Figure 4.7 - ATR of a Trace in Class(0) Transport Layer Protocol NFS Specification
(a) is test results behaviour and (b-d) are test results behaviour reference tables

ASP References			
ASP: TCREQ (Transport Connection Request) Behaviour Table Id: LOG_EXAMPLE			
List Name	Field Name		
	to.t.addr	from.t.addr	qts
S0	<observed values of ASP parameters>		

(e)

ASP References		
ASP: TCCON (Transport Connection Confirmation) Behaviour Table Id: LOG_EXAMPLE		
List Name	Field Name	
	resp.t.addr	qts
S0	<observed values of ASP parameters>	

(f)

ASP References	
ASP: TDATR (Transport Data Request) Behaviour Table Id: LOG_EXAMPLE	
List Name	Field Name
	TS User-Data
S0	<observed values of ASP parameters>

(g)

ASP References		
ASP: TDIND (Transport Disconnection Indication) Behaviour Table Id: LOG_EXAMPLE		
List Name	Field Name	
	Disc-reason	TS User-Data
S0	<observed values of ASP parameters>	

(h)

ASP References	
ASP: NDIND (Network Disconnection Indication) Behaviour Table Id: LOG_EXAMPLE	
List Name	Field Name
	NS User-Data
S0	<observed values of ASP parameters>

(i)

Figure 4.7 (Contin.) - ATR of a Trace in Class(0) Transport Layer Protocol NFS Specification - (e-i) are test results behaviour reference tables

4.3.3 Discussion: Test Results Verification Using Protocol Specification

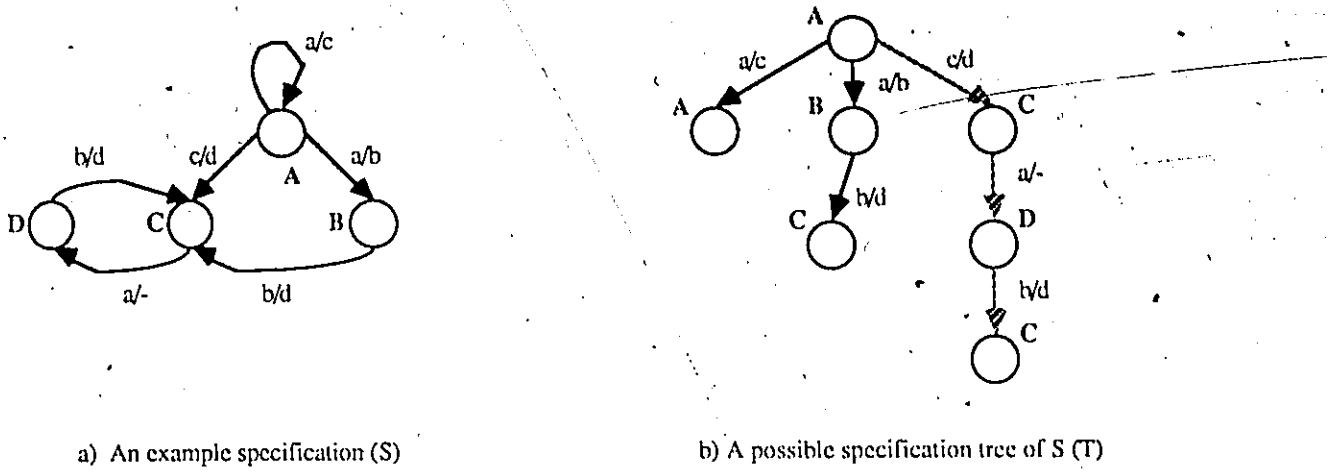
In Chapter 3, we showed how ATR's are used in validating the test specification and the test environment files (i.e. PICS and PIXIT). Here we show how, in parallel, ATR's can be cross-validated against the protocol specification. To demonstrate this, we assume Formal Description Techniques (FDT's) are used in specifying the protocol. Using FDT's, protocol specifications may be represented as specification trees [Turn87, TuUr88]. In fact for FDT's such as LOTOS, there are tools available for generating specification trees from a formal protocol specification [GuLo88]. In each protocol specification, a set of mandatory and optional capabilities is specified. The implemented capabilities are stated in a PICS. These capabilities can be mapped to a set of test purposes and to a sequence of test events in the test case (i.e. Test body). Each of these test purposes are represented by a set of subtrees in the specification tree.

Also, for some categories of test cases (e.g. transition tests) a preamble subtree and postamble subtree may be appended to the beginning and end of the test purpose subtree. In ATR's, the test results behaviours section identifies the test body fragment. Therefore, one can directly map the test result behaviours to the related specification tree segment and the related specification segment itself.

Suppose we have a FSM specification 'S' given in figure 4.8 (a). A finite distinguishing set of sequences from such a specification can be represented by a specification tree 'T' in figure 4.8 (b). The detail of the input and output parameters are not given in these specifications. Therefore, when one is required to verify the results against the specification, he/she does not want to be overwhelmed by the detail of the executed test result.

Assume the transition test objective (test purpose) is represented by the dotted trace (in this case there is no preamble or postamble sequence) in the specification tree 'T'. Then a successful test result for such a test objective may look like the one in figure 4.9. Information such as timer operations in the log behaviour description and individual behaviour reference tables etc. are not included in this example, but there is enough information to see how traces in the specification can be reconstructed from the ATR's. In this case the verdict of the results is "pass", therefore it completely matches the specification trace, but sometimes an illegal behaviour may happen, which can be also easily identified. Note that this is only a simple example; in testing of complex protocols

such an organization of test output results would be of a great help while verifying test results against the protocol specification.



a) An example specification (S)

b) A possible specification tree of S (T)

Figure 4.8 (a-b) - A FSM Specification Fragment and its Generated Specification Tree Fragment

Test Results Behaviour						
Reference :	EXAMPLE/TRACE					
Identifier :	TRACE					
Purpose :	To show that implementation conforms to the dotted trace in 'T'.					
Log Behaviour Description		**E**	**ATS event Location**	**Segments Start**	**End**	**Behaviour References**
MAIN-TREE [L,U]				TS		
U! c				TB		c(S0)
L?'d						d(S0)
U! a						a(S0)
U! b						b(S0)
L? d					TB	d(S1)
MAIN-TREE [L,U]					TE	
Test Case Start Time :	12:00:00.00					
Test Case End Time :	12:00:00.30					
Num of Retries :	0					
Verdict	PASS					

Figure 4.9 - An ATR Fragment for The Dotted Trace in 'T' (A,C,D,C)

If ATR's are not used, the amount of detail in executed test reports makes the relating of test results to the protocol specification an impossible task. The ATR's make this activity possible, albeit, in a manual way. The development of tools to automate this process is a logical next step, but beyond the scope of this thesis.

Thus, use of ATR's is recommended for verifying results against the formal specifications. In the next section we suggest methods for summarizing test results in preparation for test result diagnosis.

4.4 Analysis and Diagnosis of Test Results

Due to the large amount of data available for detailed diagnosis of test results, there is always a need to summarize such data in some form of understandable reports. Using ATR's, data summarization can be handled in two manners. One way is showing the interdependencies among test results, using a test dependency model (section 4.4.1). Alternatively, the results can be summarized using tables, which are useful for cross referencing between the results (section 4.4.2). From these cross-reference reports diagnostic inferences may be drawn. As well, as an extension of the concept of high-level test reporting new techniques and tools for auditing the test results may be developed.

In section 3.2 we introduced some enhancements for TTCN as a test specification language. Here we use such enhancements for generating these summarized data reports.

4.4.1 Test Dependency Model

Test analysts would likely prefer to initially study the PCTR's of ATR's and identify the complexity of problems, before even looking at other detailed informations such as the conformance log. This was observed in a study of expert analysis of X.25 test results [Prob88b]. At this stage, they may also be able to do a high level diagnosis. For such a high level diagnosis, all they need is an efficient mechanism to quickly discover the symptoms of failures common to general test case results. A mechanism is needed which would enable them to visualize the inter-dependencies between the test results in a higher level, graphical manner. Such a time saving mechanism could be used to automatically generate inter-dependency reports from the conformance logs. This test dependency model may be useful as an aid to discovery of errors.

In addition, because this analysis shows interdependencies between testcases (i.e. if the above concept is applied to the ATS), it may also play an important role in the test selection phase. In an online mode of test selection for execution, observation of failure in a test step in the preamble, test body or postamble may allow avoidance of unproductive execution of other following dependent test cases. Currently test operators can investigate such inter-dependencies manually (both for test results and test cases), but this can be a very time consuming activity.

Test results within an ATR are usually inter-dependent, often simply because parts of the test data used in a test case are used in other test cases. Therefore the obtained test result traces are inter related. In Section 3.2, we introduced a mechanism for structuring

test suites. We required the test specifier to identify preamble and postamble(s) in the identification section of each test case. Using the given syntax, the preamble, test body and postamble events and their location in the ATS can be captured in the test results using ATR's. The inter relationship between the test results can be graphically presented in a set of Test Dependency Graphs (TDG's), and it can be further refined into Test Dependency Trees (TDT's) as follows:

Definition 4.6: We define a T as a finite, non empty ordered set of test results as following:

$T = \{TR_1, TR_2, ..., TR_i, ..., TR_j, ..., TR_n\}$

Each of the test results $TR_1, ..., TR_n$ may reference one preamble test step, one or more test steps for the test body and a postamble test step from their corresponding test case. If these test steps are expanded and searched, a more complete list of test steps can be created. we define these lists for each TR as follows (any of the following sets could be empty, e.g., when there is no preamble, postambles, or test body test steps, the test body is contained in the main behaviour of the test case itself):

$PA \text{ (preamble)} = \{pa_1, pa_2, ..., pa_{m1}\}$ $\quad |PA| \geq$ number of states

$TB \text{ (test body)} = \{tb_1, tb_2, ..., tb_{m2}\}$ $\quad |TB| \geq$ number of states

$PO \text{ (postamble)} = \{po_1, po_2, ..., po_{m3}\}$ $\quad |PO| \geq$ number of states

$TR_i \in PA \times TB \times PO$

The test steps referenced within the test results may be defined as follows:

$PBTS = \{pb_1, pb_2, ..., pb_k, ..., pb_{m4}\}$ (Public test steps - referenced in two or more TR)

$PRTS = \{pr_1, pr_2, ..., pr_l, ..., pr_{m5}\}$ (private test steps - referenced in only one TR)

$(PBTS \cup PRTS) \supset PA$, $(PBTS \cup PRTS) \supset TB$ and $(PBTS \cup PRTS) \supset PO$

Now, if there exists $s \in TR_i$, $t \in TR_j$, such that $i \neq j$ and $s = t = pb_k$

$\Rightarrow TR_i$ and TR_j are potentially inter-dependent

AND, if $s \in PA_i$ Then TR_i is 'Preamble Dependent' on TR_j .....(1)

AND, if $s \in TB_i$ Then TR_i is 'Test Body Dependent' on TR_j .....(2)

AND, if $s \in PO_i$ Then TR_i is 'Postamble Dependent' on TR_j .....(3)

and similarly for TR_j. TR_i and TR_j may be inter-dependent in more than one test case component.

Test Dependency Graphs (TDG's) show the relationships between the test results by a graphical means. Test dependency graphs can be created using the relations (1), (2) and (3). To show the relationship (i.e. bidirectional dependence) of a test result 'A' to a test result 'B' in a graph we use a three digit binary number. Digits from left to right identify the existence of relations (1) to (3) respectively, '0' for each digit means relation does not exist, where as '1' signifies a relationship. We refer to this number as a relationship code. The relationship code can be interpreted as follows:

e.g. A ----110---> B means that if 'A' failed in the preamble or test body, its failure is expected to be related to failure of 'B'. However, if 'A' failed in the postamble, one can not relate it to 'B'.

Similarly, we may have a relationship code to show the relationship of a test result 'B' to test result 'A'

e.g. A <---001---- B means that if 'B' failed in the postamble, its failure is expected to be related to failure of 'A'. However, if 'B' failed in the preamble or test body, one can not relate it to 'A'.

If the relationship code for a pair of test results is '000', it is not shown in the graph. If the relationship code is '000' in both directions, the edge between those test results is simply omitted from the TDG (i.e. means the pair of test results are independent). Therefore as a result we might have a group of disjoint TDG's to represent the relationships between test results in an ATR.

Algorithm 4.1: Algorithm for generating TDG's using the ATR's.

1) For each and every combination pairs of the test results in the ATR follow instructions (2) to (6).
2) Initially the relationship code for showing the relationship between one test result to another is set to '000'.
3) For each of the test results, parse the list of preamble and postamble. Create hierarchial lists of public test steps used in preamble and postambles by expanding the original list.
4) Once all the public test steps used in the preamble and postamble are identified, create a similar lists for the public test steps referenced in the test bodies of the test results.

5) Starting from top of the hierarchial lists of public test steps for preamble, test body and postamble, investigate the commonality in the test steps. Follow the conditions given below:

   a) If a test step in the list of public test steps for preamble of a test result is used in the other test result, stop the investigation and set the first digit of the relationship code to '1'.

   b) If a test step in the list of public test steps for postamble of a test result is used in the other test result, stop the investigation and set the second digit of the relationship code to '1'.

   c) If a test step in the list of public test steps for test body of a test result is used in the other test result, stop the investigation and set the third digit of the relationship code to '1'.

6) If the relationship code is not equal to '000', update the TDG's.

These graphs may be used to generate **Test dependency Trees (TDT's)** for any chosen test result within the ATR. In [Chow78] the algorithm for generating trees from FSM's (a type of graph) is given. We use a similar logic in producing our TDT's from TDG's. The algorithm is given as follows:

Algorithm 4.2: Algorithm for generating TDT's from TDG's.

1) Label the root of the TDT with the selected test case. This is level $k = 1$.
2) Suppose we have already built the TDT to a level k. The $(k+1)$th level is built by examining nodes in the kth level from left to right. A node at the kth level is terminated if the node has no more edges associated with it. Otherwise, expand the node according to the remaining TDG, and remove those edges which you just considered from the TDG.

Lemma: Algorithm 4.2 terminates even in presence of cycles in TDG.

Note: The above process always terminates, since there are only a finite number of test case result represented in TDG. Also, depending on the left to right order in which we place the successor nodes, a different tree may created.

Example 4.3: Given a set of test results 'A' to 'H', the list of public test steps in the preamble, test body and postamble is created (Figure 4.10), from which the TDG's are derived (Figure 4.11) using algorithm 4.1.

Test Results	Preamble TS's	Test Body TS's	Postamble TS's	Expan. of Common Test Steps
Test Result 'A'	a	b	c	
Test Result 'B'	a	c	g	
Test Result 'C'	b	c,d	e	d ---> g
Test Result 'D'	None	None	c	
Test Result 'E'	f	None	None	
Test Result 'F'	None	None	None	
Test Result 'G'	None	None	f	
Test Result 'H'	e	None	None	

Figure 4.10 - Expanded List of Public Test Steps for Test Results 'A'-'H'

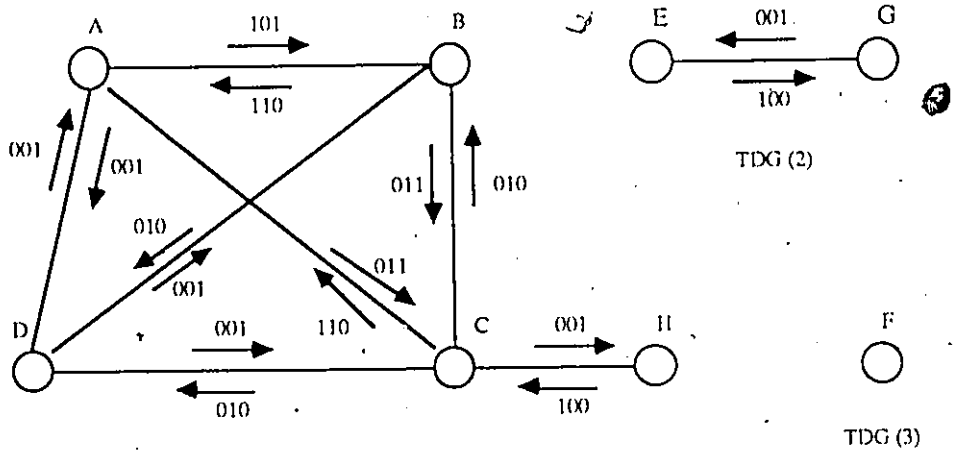
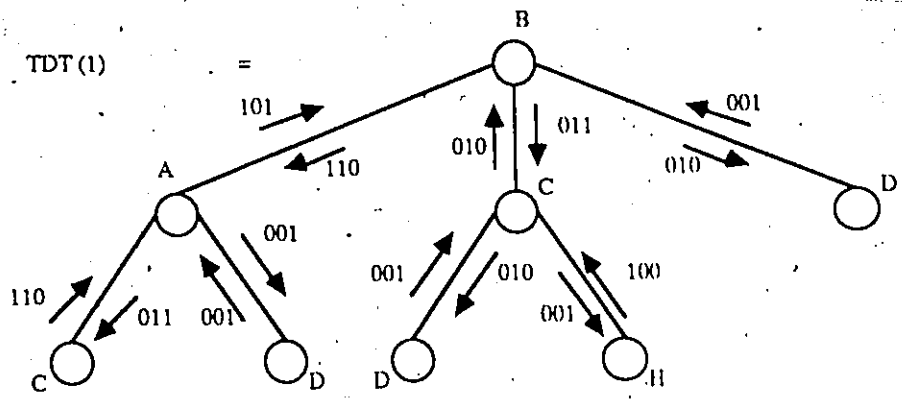
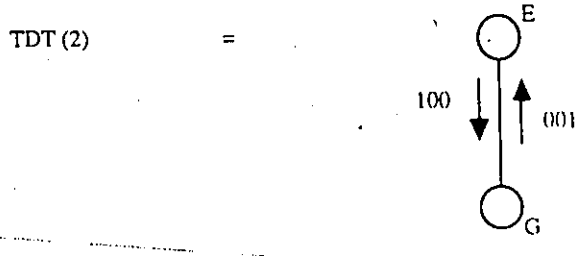


Figure 4.11 - Test Dependency Graphs for Test Results 'A'-'H'

The TDT's for test results 'A', 'E' and 'F' are constructed using the TDG's (Figure 4.11) together with algorithm 4.2 (Figure 4.12).



- Test Dependency Tree for Test Result 'B'



- Test Dependency Tree for Test Result 'E'

- Test Dependency Tree for Test Result 'F'

Figure 4.12 - Test Dependency Trees for Test Results 'B', 'E' and 'F'

□

A simple real example of the use of TDT's and TDG's may be provided using the data link layer test suite [ISO8882-2]. We select three test cases from the test suite. We then show the relationship between the possible test results which may be obtained from running these test cases.

Test Case Name	Test Purposes
DL4_205	Verify DTE sends FRMR/W=1, CR=0 in response to undefined command in the DL4_STATE.
DL4_206	Verify that DTE responds FRMR/W=1, CR=1 to undefined response in the DL4_STATE.
DL4_301	Verify that DTE initiates link reset in response to DM/F=1 in DL4_STATE.

In the case of all of these test cases, the test results may be related in the preamble, as they all need a test data sequence to bring the test to the DL4_STATE. In the case of test cases DL4_205 and DL4_206 the test results may also be inter-related in the postambles as they have to verify that they reach DL5 STATE after achieving their test purposes. In the case of DL4_301 that part of the result is independent from the other two test results and it checks for link reset instead. Thus, this model is very effective way of representing these inter-dependencies between the test results. The job of audition of test results may be simplified using this model, as TDT's and TDG's can be derived automatically from the ATR's.

4.4.2 Results Cross Referencing Model

By introducing TDG's and TDT's we provided a means of graphically showing the relationships which exists between test cases. Although this would be a good aide for an on-line or off-line test result analysis (i.e during or after execution), another alternative approach would be possible, using Cross Reference Tables (CRT's). Test analysts usually create CRT's or similar charts to investigate the major problems in test results. The fact that now we have introduced a formal syntax for test results (i.e. ATR's), these charts (i.e. CRT tables) can be created automatically using ATS's (statically) and ATR's (dynamically). The way CRT's are designed (Figure 4.13), they provide a quick reference for diagnosing test results. From the CRT's approximate location of errors can be identified. Further diagnosis (i.e. further steps in auditability of results) can be facilitated using CRT's.

Each test case may have private segments and/or public segments (i.e. global test steps). A private segments column followed by public segments columns (global test steps) headings constitute the vertical inputs of the table. Each segments is also subdivided to preamble, test body and postamble (**P, B, O**). For each failed executed test case in ATR's, there will be an entry in the CRT. Initially all the boxes are empty ( ). The boxes for which

test cases are executed and test result is pass would be marked (✓). Finally, the boxes for which test cases are executed and test result is fail would be marked (✗).

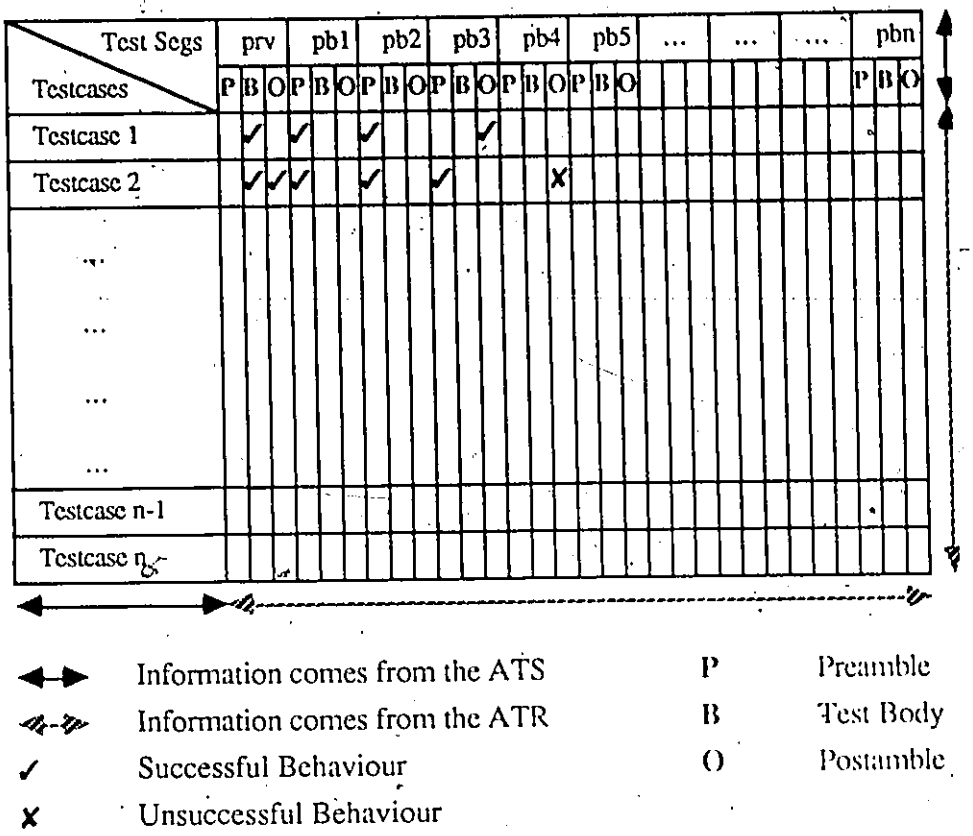


Figure 4.13 - Example of a Cross Reference Table (CRT)

Horizontal columns in the table display the testing summary for each test case. Vertical columns in the table display the information with regard to particular segments of the test suite which is referred in the test results traces. The location and seriousness of the errors in the test results may be easily identified using CRT's. CRT's are a very high level reports, which condense the history of each test in a single row in the table. They may be extremely useful in analysis of multiple runs. CRT's are also useful for recording the test session histories. The presentation and readability aspects of CRT's makes them very useful for regression testing.

4.5 Summary

In this chapter, a framework for test results analysis with respect to high level abstract test results was introduced. A mapping between the ATR's and formal protocol specifications was found to be useful for verifying test results. In addition, methods such as test dependency graphs, trees and cross reference tables aide in diagnosing test results in terms of locating the errors. Moreover, we demonstrated graphical summarization reports for test results were useful to test analysts.

In the next and final chapter, we present a summary of the thesis, together with our conclusions and give directions for future research in the area of test results analysis.

Chapter 5

Conclusions & Suggestions for Future Work

5.1 Summary of the Thesis

Given the conformance testing standardization work, in this thesis, we make an effort to introduce a new methodology for test results analysis. Test results analysis is the final step in conformance testing. To date, as conformance testing has been formalized, less effort has been focussed on this topic in comparison to areas such as test derivation, selection or test execution. However, it is expected that test results analysis will soon receive much more attention.

We started this thesis (chapter 1 & 2) by identifying the problem of test results analysis. We presented a study of the current adhoc methods for test results analysis and tried to combine these ideas and to relate them to the available standards work [ISO9646-1,5].

In chapter 3 of this thesis it is proposed that a standardized, formalized syntax for test results is the key to automating analysis of test results. To make these ideas concrete, a test results notation (TTCN-TR) based on TTCN (a standard test specification notation) was introduced. Test results represented in high level manner using TTCN-TR were shown to facilitate relating the test results to the test purposes in formal specifications in Estelle as well as to test specifications in TTCN. Finally in chapter 4 we presented methods for summarizing the large quantity of data which is obtained after each test session. This should reduce the time which test analysts spend in diagnosing test results.

It is believed that the quality of analysis which may be carried out at the test results analysis stage is directly proportional to the amount of information which is made available in the test specifications and in the conformance results logs. The more detailed and precise the information in these sources, the more detailed the analysis which can be performed. To fully automate the process of analysis of test results, this information must be made available in a formalized manner.

This thesis is based on the TTCN test notation and is limited to the functionalities which are provided using this semi-formal test notation. This is reasonable, however, since TTCN is currently the only standardized test notation. The tools which may be developed for deriving the ATR's from ETR's would be also dependent on the test method selected.

5.2 Suggestions for Future Work

The research in test results analysis may take two different directions to deal with the short or long term problems in this area. The short term solution may be achieved by continuing this work and developing the following recommended tools to achieve a semi-automatic test result analysis process:

- Abstract test result (ATR) derivation tool.
  A test deriviation tool can be used to convert an ETR to its equivalent ATR.
- Tool(s) for generating test result summary reports (i.e. in graphical form) for diagnosis of test results. The approach of section 4.4 is original and is useful for automating the process of searching for common location of errors. This process is currently conducted manually by test analysts. A similar tool can be created if the same idea is applied to the test suites rather than test results.

- Comparability and repeatability report generator.
  It is hoped that the ATR syntax suggested in this thesis (or a more improved version of this ATR syntax) becomes a standard for representation of test results. Once this objective is achieved the above tools can be implemented to automatically compare test result reports to promote universality of conformance testing.
- Tools for auditing completeness and correctness of test results.
  Similarly the machine processable version of TTCN-TR can be used as test results input to auditing tools. But again it would be better that implementation takes place once the notation is standardized.

Longer term studies may involve working in the following directions given as follows:

- Test requirement languages to formally describe the test purpose requirements. The goal would be to describe the test purpose in a simple, general, but effective way which would be useful for test result analysis.
- Estimation or deduction of possible unobservable trace segments. It was suggested as an extension that behaviour of transient transitions could in principle also be represented in the ATR's. This will require a precise mapping between test cases and the design of the protocol implementation. It would make the ATR's even more effective in verifying test results versus the protocol specifications, but is not feasible at present.
- Use of expert systems in test results analysis. In test result analysis, the test analysts usually make use of their past experiences. It should be useful to have an expert system which may based on an incremental knowledge base, making use of the knowledge which is gained for past test session test result analysis [prob88b].

Test result analysis is a very new topic, with an incompletely understood framework. This is a primary effort with very little work being done elsewhere yet, [illegible] the same time is of immense international interest. It is hoped that this work would serve as a useful framework for this important stage of conformance testing.

Appendix 'A'

Machine Processable Form of TTCN-TR

The Syntactic Metanotation for specifying the BNF grammar for TTCN-TR is given as follows::

::=	is defined to be
|	alternative
[a]	0 or 1 instances of <a>
{a}	0 or more instances of <a>
{a}+	1 or more instances of <a>
(...)	textual grouping
abc	the non-terminal symbol $abc
$abc	the terminal symbol $abc
"abc"	the terminal symbol <abc>

This annex defines the machine processable form of TTCN-TR called TTCN-TR/MP. The machine processable vesrion of TTCN-TR provides a formal syntax for the TTCN-TR/GR. It can be used to transfer the test results data between different test centres. It aen be used as basis for detailed comparability of abstract test reports.

The Conventions which are used in representing the TTCN-TR/MP is very much the same as the conventions used for representing TTCN/MP. These are explained as follows [ISO9646-2a,3]:

a) TTCN-TR keywords (terminal symbols) that belong only to the TTCN-TR/MP form start with the dollar character ($);

   EXAMPLE 1 - **$ATRid**

b) TTCN keywords (terminal symbols) that belong to both the TTCN-TR/MP and TTCN-TR/GR do not start with dollar character ($);

   EXAMPLE 2 - the TTCN-TR keyword: START

c) Productions of the kind:

   $Begin.Keyword.....$End.Keyword

   represent complete tables that are defined in the TTCN-TR/GR

EXAMPLE 3 - **$Begin.TimerReferences.....$End.TimerReferences**

d) Productions where **Begin.** is not part of the first keyword and which have a closing keyword are used to represent sets of tables, or sets of fields in a table:

EXAMPLE 4 - **$ReferencesPart.....$End.ReferencesPart**

$Refdcl.....$End.REFdcl

e) Productions where $Begin. is omitted from the first keyword and that have no closing keyword, represent individual fields in the table. These fields contain text. The syntax for this text is common to both the TTCN-TR/MP and the TTCN-TR/GR,

EXAMPLE 5 - ATRid ::= **$ATRid** identifier

NOTE: Since this grammar is not fully exercised, it is likely that few minor ambiguities might exist.

SECTION 1: Abstract Test Results Report:

1. ATRreport ::= **$ATRreport**
   ATRid PCTReport ParameterPart ResultBehaviourPart ReferencesPart
   $End.ATRreport

2. ATRid ::= **$ATRid** Identifier

SECTION 1.1: Protocol Conformance Test Report (PCTR):

3. PCTReport ::= **$Begin.PCTReport** IdentSum StaticSum
   DynamicSum StaticReview TestCampaign [OverallObs]
   $End.PCTReport

4. StaticSum ::= **$StaticSum** FreeText

5. DynamicSum ::= **$DynamicSum** FreeText

6. StaticReview ::= **$StaticReview** FreeText

7. OverallObs ::= **$OverallObs** FreeText

SECTION 1.1.1: PCTR Identification Summary:

8. IdentSum ::= **$IdentSum** PCTRintro IUTintro TestEnv LimitsRes
   $End.IdentSum

9. PCTRintro ::= **$PCTRintro** PCTRnum PCTRdate CorresSCTRnum
   CorresSCTRdate ManagerName ManagerInitial **$End.PCTRintro**

10. PCTRnum ::= **$PCTRnum** Number

11. PCTRdate ::= **$PCTRdate** Date

12. CorresSCTRnum ::= **$CorresSCTRnum** Number

13. CorresSCTRdate ::= **$CorresSCTRdate** Date

14. ManagerName ::= **$ManagerName** Name

15. ManagerInitial ::= **$ManagerInitial** Name

16. IUTintro ::= **$IUTintro** IUTname IUTversion StandardRef PICSref
    PrevPCTR **$End.IUTintro**

17. IUTname ::= **$IUTname** Identifier

18. IUTversion ::= **$IUTversion** number

19. StandardRef ::= **$StandardRef** FreeText

20. PICSref ::= **$PICSref** FreeText

21. PrevPCTR ::= **$PrevPCTR** Identifier

22. TestEnv ::= **$TestEnv** PIXITref ATSstandard ATMused RealTester ETSref ProtLayerInfo ProtSublayerInfo TestDates TestOperators TestAnalysts ConfLogRef **$End.TestEnv**

23. PIXITref ::= **$PIXITref** FreeText

24. ATSstandard ::= **$ATSstandard** FreeText

25. ATMused ::= **$ATMused** FreeText

26. RealTester::= **$RealTester** FreeText

27. ETSref ::= **$ETSref** Identifier

28. ProtLayerInfo ::= **$ProtLayerInfo** FreeText

29. ProtSublayerInfo ::= **$ProtSublayerInfo** FreeText

30. TestDates ::= **$TestDates** {Date}+

31. TestOperators ::= **$TestOperators** {Name}+

32. TestAnalysts ::= **$TestAnalysts** {Name}+

33. ConfLogref ::= **$ConfLogref** FreeText

34. LimitsRes ::= **$LimitsRes** FreeText

SECTION 1.1.2: PCTR Test Campaign Summary:

35. TestCampaign ::= **$TestCampaign** {ATSref SATSflag PETSflag Runflag TRBehaviourRef Verdict Unforeseen LogObs}+ **$End.TestCampaign**

36. ATSref ::= **$ATSref** TestCaseReference

37. TestCaseReference ::= TestGroupReference TestCaseName

38. TestGroupReference ::= SuiteIdentifier "/" {TestGroupIdentifier "/"}

39. TestCaseName ::= Identifier

40. SuiteIdentifier ::= Identifier

41. TestGroupIdentifier ::= Identifier

42. SATSflag ::= **$SATSflag** BooleanType

43. PETSflag ::= **$PETSflag** BooleanType

44. Runflag ::= **$Runflag** BooleanType

45. TRBehaviourRef ::= **$TRBehaviourRef** TestResultReference

46. TestResultReference ::= TestResultGroupReference TestResultName

47. TestResultGroupReference ::= ATRid "/" {TestResultGroupIdentifier "/"}

48. TestResultName ::= Identifier

49. TestResultGroupIdentifier ::= Identifier

50. Verdict ::= **PASS | INCONC | FAIL1 | FAIL2 | FAIL3**

51. Unforeseen ::= **$Unforeseen** identifier

52. LogObs ::= **$LogObs** FreeText

SECTION 1.2: Test Results Parameter Part:

53. ParameterPart ::= **$ParameterPart** [TRParams] [TRGlobalCons] **$End.ParameterPart**

SECTION 1.2.1: Test Results Parameter table:

54. TRParams ::= **$Begin.TRparams** {TRParamEnt}+ **$End.TRparams**

55. TRParamEnt ::= **$TRParamEnt** TRParId TRParValue **$End.TRParamEnt**

56. TRParId ::= **$TRParId** TRParIdentifier

57. TRParIdentifier ::= Identifier

58. TRParValue ::= **$TRParValue** Value

SECTION 1.2.2: Test Results Global Constant Table:

59. TRGlobalCons ::= **$Begin.TRGlobalCons** {TRGlobalConEnt}+ **$End.TRGlobalCons**

60. TRGlobalConEnt ::= **$TRGlobalConEnt** TRGblConId TRGblValue **$End.TRGlobalConEnt**

61. TRGblConId ::= **$GblId** TRGblIdentifier

62. TRGblIdentifier ::= Identifier

63. TRGblValue ::= **$TRGblValue** Value

SECTION 1.3: Test Results Behaviour Part:

64. ResultBehaviourPart ::= **$ResultBehaviourPart** {TRBehaviour} {TRGlobalVars} **$End.ResultBehaviourPart**

SECTION 1.3.1: Test Results Behaviour (includes Conformance Log Record):

65. TRBehaviour ::= **$Begin.TRBehaviour** TRBehaviourRef TRBehaviourId GlobalTableId TestPurpose LogBehaviourDescription TRStartTime TREndTime NumOfRetries Verdict ETRcomment **$End.TRBehaviour**

66. TRBehaviourId ::= **$TRBehaviourId** TRBehaviourIdentifier

67. TRBehaviourIdentifier ::= Identifier

68. GlobalTableId ::= **$GlobalTableId** GlobalTableIdentifier

69. GlobalTableIdentifier ::= Identifier

70. TestPurpose ::= **$TestPurpose** FreeText

71. LogBehaviourDescription ::= **$LogBehaviourDescription** LogHeader {LogBehaviourLine}+ LogTrailer **$End.LogBehaviourDescription**

72. LogHeader ::= **$LogHeader** MainTreeIdentifier [FormalPCOlist] [FormalPARlist] LogStartSegment **$End.LogHeader**

73. LogTrailer ::= **$LogTrailer** Indentation MainTreeIdentifier [FormalPCOlist] [FormalPARlist] LogEndSegment **$End.LogTrailer**

74. LogStartSegment ::= "TS"

75. LogEndSegment ::= "TE"

76. Delim ::= ","

77. MainTreeIdentifier ::= Identifier

78. FormalPCOlist ::= "[" PCOidentifier {Delim PCOidentifier} "]"

79. FormalPARlist ::= "(" PARidentifier {Delim PARidentifier} ")"

80. PCOidentifier ::= Identifier

81. PARidentifier ::= Identifier

82. TRStartTime ::= **$TRStartTime** Time

83. TREndTime ::= **$TREndTime** Time

84. NumOfRetries ::= **$NumOfRetries** Number

85. ETRcomment ::= **$RefEndTime** FreeText

SECTION 1.3.1.1: Entries in the Test Results Behaviour Description:

86. LogBehaviour Line ::= **$LogBehaviourLine** EventLine [Enum]
Location Segment [LogBehaviourRef] **$End.LogBehaviourLine**

87. Enum ::= **$Enum** ("*" | "+" | "<" | ">") "(" Number ")"

88. Location ::= **$Location** LocationId

89. LocationId ::= Identifier

90. Segment ::= **$Segment** [EventStartSegment] [EventEndSegment]

91. EventStartSegment ::= "PA" | "TB" | "PO" | "DB" | (("R" | "P") Number)

92. EventEndSegment ::= "PA" | "TB" | "PO" | "DB" | (("R" | "P") Number)

93. LogBehaviourRef ::= **$LogBehaviourRef** Bref {Delim Bref}

94. Bref ::= (ASPidentifier | PDUidentifier | BtimerIdentifier)
"(" BrefIdentifier ")"

95. BrefIdentifier ::= **$BrefIdentifier** identifier

96. ASPidentifier ::= Identifier

97. PDUidentifier ::= Identifier

98. BtimerIdentifier ::= Identifier

SECTION 1.3.1.2: Test Results Events:

99. EventLine ::= (Event | PseudoEvent)

100. Indentation ::= "[" Number "]"

101. Event ::= (Send | Receive | Timeout | Elapse)

102. Send ::= [PCOidentifier] "!" (ASPidentifier | PDUidentifier) [EncodedAs]
[TTCNExpression] [TimerOperation]

103. Receive ::= [PCOidentifier] "?" (ASPidentifier | PDUidentifier)
[DecodesAs] [TTCNExpression] [TimerOperation]

104. Timeout ::= **?TIMEOUT** [TimerTypeIdentifier [Delim TimerIdentifier]]
[TTCNExpression]

105. Elapse ::= **?ELAPSE** [TTCNExpression]

106. PseudoEvent ::= [TTCNExpression] [TimerOperation]

SECTION 1.3.1.3: Test Results Expressions:

107. TTCNExpression ::= {BooleanExpression} {Assignment}

108. Assignment ::= "(" SimpleAssignment {Delim SimpleAssignment} ")"

109. SimpleAssignment ::= (VARidentifier | ASP_PDUreference) ":=" (["-"] (Aterm | Term)) | BooleanExpression

110. Aterm ::= Term ArithOp Term

111. Term ::= TCVid | ASP_PDUreference | Value | ArithmeticExpression

112. ArithmeticExpression ::= "(" Aterm ")"

113. ArithOp ::= "+" | "-" | "*" | "/" | **MOD**

114. BooleanExpression ::= "[" SimpleBooleanExpression | CompoundBooleanExpression "]"

115. SimpleBooleanExpression ::= Bterm | (**NOT** "(" Bterm ")"

116. Bterm ::= Term | (Term Relop Term)

117. Relop ::= "=" | "<" | ">" | "<>" | "≥" | "≤"

118. CompoundBooleanExpression ::= "(" SimpleBooleanExpression ")" {(**AND** | **OR**) "(" SimpleBooleanExpression ")"}+

119. EncodedAs ::= "(" (ASP_PARidentifier | FieldIdentifier) "^" PDUidentifier {EncodedAs} ")"

120. DecodesAs ::= "[" (ASP_PARidentifier | FieldIdentifier) "~" PDUidentifier {DecodesAs} "]"

121. ASP_PDUreference ::= ASPreference | FieldReference

122. ASPreference ::= (ASPidentifier "." ASP_PARidentifier)

123. FieldReference ::= (ASPidentifier "." FieldIdentifier) | (PDUidentifier "." FieldIdentifier) | (FieldIdentifier "." FieldIdentifier)

124. TCVid ::= TRParIdentifier | CONidentifier | VARidentifier

125. CONidentifier ::= Identifier

126. VARidentifier ::= Identifier

127. ASP_PARidentifier ::= Identifier

128. FieldIdentifier ::= Identifier

SECTION 1.3.1.4: Timer Operations:

129. TimerOperation ::= "(" StartTimer | CancelTimer ")"

130. StartTimer ::= **START** [TimerTypeIdentifier Delim TimerIdentifier]

131. CancelTimer ::= **CANCEL** [TimerTypeIdentifier [Delim TimerIdentifier]]

132. ResumeTimer ::= **RESUME** [TimerTypeIdentifier [Delim TimerIdentifier]]

133. SuspendTimer ::= **SUSPEND** [TimerTypeIdentifier [Delim TimerIdentifier]]

134. TimerIdentifier ::= Identifier

135. TimerValue ::= IntegerValue | IntegerExpression

136. IntegerValue ::= Number | TCVid
/ where the value of TCVid is of type integer */*

137. IntegerExpression ::= "(" ArithmeticExpression ")"
/ expression containing only integer types */*

138. TimerTypeIdentifier ::= Identifier

SECTION 1.3.2: Test Results Behaviour (Global Variables Table):

139. TRGlobalVars ::= **$Begin.TRGlobalVars** GlobalTableId TRBehaviourId {TRGlobalVarEnt}+ **$End.TRGlobalVars**

140. TRGlobalVarEnt ::= **$TRGlobalVarEnt** TRGblVarId TRGblInitialValue TRGlobalEndValue **$End.TRGlobalVarEnt**

141. TRGblVarId ::= **$TRGblVarId** TRGblVarIdentifier

142. TRGblVarIdentifier ::= Identifier

143. TRGblInitialValue ::= **$TRGblInitialValue** Value

144. TRGblEndValue ::= **$TRGblEndValue** Value

SECTION 1.4: Test Result References Part:

145. ReferencesPart ::= **$ReferencesPart** {TR_ASPreferences} {TR_PDUreferences} {TR_TimerReferences} **$End.ReferencesPart**

SECTION 1.4.1: Test Result ASP References (Tabular Form):

146. TR_ASPreferences ::= **$Begin.TR_ASPreferences** ASPidentifier TRBehaviourId {RefId ASP_PV}+ **$End.TR_ASPreferences**

147. RefId ::= **$RefId** RefId&ParList

148. RefId&ParList ::= RefIdentifier [FormalParList]

149. RefIdentifier ::= Identifier {"." Identifier}

150. ASP_PV ::= **$ASP_PV** {ASP_ValEnt}+ **$End.ASP_PV**

151. ASP_ValEnt ::= **$ASP_ValEnt** ASP_ParId RefValue **$End.ASP_ValEnt**

152. RefValue ::= **$RefValue** Value

SECTION 1.4.2: Test Result PDU References (Tabular Form):

153. TR_PDUreferences ::= **$Begin.TR_PDUreferences** PDUid TRBehaviourId {RefId PDU_FV}+ **$End.TR_PDUreferences**

154. PDU_FV ::= **$PDU_FV** {PDU_ValEnt}+ **$End.PDU_FV**

155. PDU_ValEnt ::= **$PDU_ValEnt** FieldId RefValue **$End.PDU_ValEnt**

156. FieldId ::= **$FieldId** FieldId&FullId

157. FieldId&FullId ::= FieldIdentifier [FullIdentifier]

158. FullIdentifier ::= Identifier

SECTION 1.4.3: Test Result Timer References (Tabular Form):

159. TRTimerReferences ::= **$Begin.TimerReferences** TimerIdentifier TRBehaviourId {TimerField}+ **$End.TimerReferences**

160. TimerField ::= **$TimerField** TimerRefId RefStartTime RefEndTime **$End.TimerField**

161. TimerRefId ::= Identifier

162. RefStartTime ::= **$RefStartTime** Time

163. RefEndTime ::= **$RefEndTime** Time

SECTION 1.5: General Values and Types etc.:

164. Value ::= Number | Hstring | Ostring | Bstring | Cstring

165. Identifier ::= Alpha {AlphaNum | "_"}

166. AlphaNum ::= Alpha | Num

167. Alpha ::= **A** | ..... | **Z** | **a** | ..... | **z**

168. Num ::= **0** | ..... | **9**

169. ExtendedAlphaNum ::= As defined in ISO646

170. FreeText ::= [ExtendedAlphaNum {ExtendedAlphaNum}]

171. Cstring ::= """ [Char {Char}] """

172. Char ::= *a character defined by the relevant CharacterString type*

173. Bstring ::= """ [Bin {Bin}] """ **B**

174. Bin ::= **0** | **1**

175. Hstring ::= """ [Hex {Hex}] """ **H**

176. Hex ::= Num | **A** | ..... | **F** | **a** | ..... | **f**

177. Number ::= Num {Num}

178. Ostring ::= """ [Bin Bin Bin Bin Bin Bin Bin Bin] """ **O**

179. Date ::= TimeDateType

180. Name ::= {Alpha} " " {Alpha} " " {Alpha}

181. Version ::= {Num} "." {Num}

182. Time ::= TimeDateType

183. TimeDateType ::= Num Num "/" Num Num "/" Num Num

184. BooleanType ::= **TRUE** | **FALSE**

Index of TTCN-TR/MP Productions

M

N

O

P

R

S

T

U

V

Appendix 'B'

A Typical Extracted X.25-DTE Data Link Layer Test Case

Here we present a test case and all of its corresponding test steps as part of the X.25 DTE Data Link Layer test suite [ISO8882-2] (Note, this is an updated version of the test case). The selected test case is of transition test type. It consists of the preamble, test body and postamble test steps. The TTCN test specification notation [ISO9646-2a] is used to represent this test case. Only those tables of the test suite which are used by this test case are specified here.

SECTION B.1: Test Suite Overview Table:

Suite Overview			
Reference to Standards :	ISO Draft International Standard for X.25 LAPB protocol		
Reference to PICS :	PICS Reference		
Reference to PIXIT :	PIXIT Reference		
How used :	Not applicable		
Test Method :	Remote test system		
Comments :	This is only one test case from the X.25-DTE Data Link Layer Conformance Test Suite.		
Test Case Id.	**Test Case Reference**	**Page**	**Purpose**
DL4_205	X.25-DTE/DL4_205		To demonstrate a transition test

SECTION B.2: Test Suite Declaration Tables:

Test Suite Parameters		
Name	**Type**	**Comments**
T1	Integer	Timer value, as specified in PIXIT.
N2	Integer	Timer value, as specified in PIXIT.

PCO Declarations	
Name	**Role**
L	SAP at the lower tester controlling and observing (N) protocol data units, user to physical layer service.

Global Variable Declarations		
Name	**Type**	**Comments**
Resp	Boolean	To check for responses while waiting
Hyper_DTE	Boolean	Active DTE or non-active DTE

Data Type Declaration		
PDU: DISC (Disconnect mode)	**Comments:** It should be used always with the poll bit set	
Protocol Control Information		
Field Name	**Type**	**Comments**
A	Octetstring	Address. As defined in X.25 standards.
P	Bitstring	Poll bit. Single bit within frame control field defined in X.25 standards.

Data Type Declaration		
PDU: SABM (Set Async. Balance Mode)	**Comments:** It should be used always with the poll bit set	
Protocol Control Information		
Field Name	**Type**	**Comments**
A	Octetstring	Address
P	Bitstring	Poll bit

Data Type Declaration		
PDU: UA (Unnumbered Acknowledge)	**Comments:**	
Protocol Control Information		
Field Name	**Type**	**Comments**
A	Octetstring	Address
F	Bitstring	Final bit

Data Type Declaration		
PDU: DM (Disconnected Mode)	Comments:	
Protocol Control Information		
Field Name	**Type**	**Comments**
A	Octetstring	Address
F	Bitstring	Final bit

Data Type Declaration		
PDU: FRMR (Frame Reject)	Comments:	
Protocol Control Information		
Field Name	**Type**	**Comments**
A	Octetstring	Address
P	Bitstring	Poll bit
WXYZ	Bitstring	\
V(S)	Integer (Modulo 8)	| Part of FRMR information field.
V(R)	Integer (Modulo 8)	|
C/R	Bitstring	/

Data Type Declaration		
PDU: RRC (Received Ready Command)	Comments: Used only as a command PDU	
Protocol Control Information		
Field Name	**Type**	**Comments**
A	Octetstring	Address
F	Bitstring	Final bit
N(R)	Integer (Modulo 8)	Receive Sequence Number

Data Type Declaration		
PDU: RR (Received Ready response)	Comments: Used only as a response PDU	
Protocol Control Information		
Field Name	**Type**	**Comments**
A	Octetstring	Address
F	Bitstring	Final bit
N(R)	Integer (Modulo 8)	Receive Sequence Number

Data Type Declaration	
PDU: I (Information frame)	**Comments:** User Data field may encode Packet Level information

Protocol Control Information		
Field Name	**Type**	**Comments**
A	Octetstring	Address
P	Bitstring	Poll bit
N(S)	Integer (Modulo 8)	Send sequence number.
N(R)	Integer (Modulo 8)	Receive sequence number.
UD	Character string	User Data
Length	Integer	frame length.

Data Type Declaration	
PDU: HEX	**Comments:** Used to generate improper frame control fields or adding hex (Hex string in frame) strings to control or supervisory frames to make them improper

Protocol Control Information		
Field Name	**Type**	**Comments**
Hstring	Octetstring	Improper frame in hex

Timer Declarations		
Timer Type Name	**Duration**	**Comments**
T1	As specified in the PIXIT	Mandatory Timer

SECTION B.3: Test Suite Dynamic Behaviour Tables:

SECTION B.3.1: Test Case Dynamic Behaviour Table (Test Body):

Dynamic Behaviour				
Reference :	X.25-DTE/DL4_205			
Identifier :	DL4_205			
Purpose :	Verify DTE sends FRMR/W=1, CR=0 in repsonse to undefined command received in L4 state.			
Default Reference :	None			
Preamble :	DL4_205:DL4_STATE			
Postamble(s) :	DL4_205:DL5_CHK			
Behaviour Description	**Label**	**Cons. Ref.**	**Verdict**	**Comments**
DL4_205 [L]				
+DL4_STATE [L], (Hyper_DTE)				
L! HEX		HEX(S0)		Undef'd Cmd.
(START T1)				
L? FRMR [Hyper_DTE = TRUE]		FRMR(S7)		
(CANCEL T1)				
+DL5_CHK [L]			PASS	
L? FRMR [Hyper_DTE = FALSE]		FRMR(S8)		
(CANCEL T1)				
+DL5_CHK [L]			PASS	
? TIMEOUT T1			FAIL	
L? OTHERWISE				
(CANCEL T1)			FAIL	

SECTION B.3.2: Initialization Test Steps (Preamble Test Steps):

Dynamic Behaviour				
Reference :	X.25-DTE/Common-Lib/Preamble/DL1_STATE			
Identifier :	DL1_STATE			
Purpose :	To put DTE into L1 state			
Default Reference :	None			
Behaviour Description	**Label**	**Cons. Ref.**	**Verdict**	**Comments**
DL1_STATE [L] (Hyper_DTE := FALSE) (Resp := FALSE) REPEAT DISCONNECT1 [L] Count [Resp = TRUE] [Count > N2]			FAIL	Repeat Guards After N2 retries
DISCONNECT1 [L] L! DISC		DISC(S1)		
(START T1) ? TIMEOUT T1	1			
L? UA (Resp := TRUE) (CANCEL T1)		UA(S2)		
L? DM (Resp := TRUE) CANCEL T1		DM(S2)		
L? DISC L! UA ---> 1		DISC(S2) UA(S1)		Unnumbered frame collision try Again!
L? OTHERWISE ---> 1				

Dynamic Behaviour				
Reference :	X.25-DTE/Common-Lib/Preamble/DL2_STATE			
Identifier :	DL2_STATE			
Purpose :	To put DTE into L2 state			
Default Reference :	None			
Behaviour Description	**Label**	**Cons. Ref.**	**Verdict**	**Comments**
DL2_STATE [L] +DL1_STATE [L] (Resp := FALSE) REPEAT DISCONNECT2 [L] Count [Resp = TRUE] [Count > N2]			FAIL	Repeat Guard After N2 retries
DISCONNECT2 [L] L! DM (START T1)	3	DM(S0)		
? TIMEOUT T1 L? SABM (Resp := TRUE) (CANCEL T1)	2	SABM(S2)		
L? DISC (Resp := TRUE) (CANCEL T1) L! UA ---> 2		DISC(S2) UA(S1)		
L? DM ---> 3		DM(S3)		
L? OTHERWISE			FAIL	

Dynamic Behaviour				
Reference :	X.25-DTE/Common-Lib/Preamble/DL4_STATE			
Identifier :	DL4_STATE			
Purpose :	To put DTE into L4 state			
Default Reference :	None			
Behaviour Description	**Label**	**Cons. Ref.**	**Verdict**	**Comments**
DL4_STATE [L] (Hyper_DTE) +DL2_STATE [L] L! UA (START T1)		UA(S1)		Ack link up
L? I (CANCEL T1) L! I (START T1) (Hyper_DTE := TRUE) L? RR (CANCEL T1)		I(S7) I(S3) RR(S3)		Hyper DTE
? TIMEOUT T1 (Hyper_DTE := FALSE) L? OTHERWISE			FAIL	Error Frame

SECTION B.3.2: Verification Test Steps (Postamble Test Steps):

Dynamic Behaviour				
Reference : LAPB/Common-Lib/Postamble/DL5_CHK				
Identifier : DL5_CHK				
Purpose : To verify DTE is in L5 state				
Default Reference : None				
Behaviour Description	**Label**	**Cons. Ref.**	**Verdict**	**Comments**
DL5_CHK [L] (WXYZ) L! RRC (START T1) L? FRMR # (CANCEL T1) ? TIMEOUT T1 L? OTHERWISE		RRC(S1) FRMR (S0 (WXYZ)) 	 FAIL FAIL	 Diagnostic check

SECTION B.4: Constraints Declration Tables:

Free Variables		
Name	**Type**	**Comments**
P	Bitstring	For command frames
F	Bitstring	For response frames
N_S	Integer (Modulo 8)	Send sequence number
N_R	Integer (Modulo 8)	Receive sequence number
CR	Bitstring	C/R bit in FRMR
WXYZ	Bitstring	WXYZ bits in FRMR

PDU Constraints			
PDU: DISC (Disconnect mode)			
List Name	**Field Name**		**Comments**
	A	P	
S1	'03'H	'1'B	
S2	'01'H	'1'B	

PDU Constraints			
PDU: SABM (Set Asynchronous Balanced Mode))			
List Name	**Field Name**		**Comments**
	A	P	
S2	'01'H	'1'B	

PDU Constraints			
PDU: UA (Unnumbered Acknowledgement)			
List Name	Field Name		Comments
	A	F	
S1	'01'H	'1'B	
S2	'03'H	'1'B	

PDU Constraints			
PDU: DM (Disconnected Mode)			
List Name	Field Name		Comments
	A	F	
S0	'01'H	'0'B	Link set up request from tester
S2	'03'H	'1'B	
S3	'03'H	'0'B	

PDU Constraints							
PDU: I (Information frame)							
List Name	Field Name						Comments
	A	P	UD	N(S)	N(R)	LENGTH	
S3	'03'H	'0'B	-	N_S	N_R	1080	Proper frame
S7	'01'H	'0'B	-	N_S	N_R	1080	Proper frame

PDU Constraints										
PDU: FRMR (FRMe Reject)										
List Name	Field Name									Comments
	A	F	V(R)	V(S)	C/R	W	X	Y	Z	
S0	A	F	V_S	V_R	'0'B	R0				
S7	'03'H	'0'B	'001'B	'001'B	'0'B	R1				
S8	'03'H	'0'B	'000'B	'000'B	'0'B	R1				

PDU Constraints					
PDU: REF					
List Name	Field Name				Comments
	W	X	Y	Z	
R0	'0'B	X	Y	Z	
R1	'1'B	'0'B	'0'B	'0'B	

PDU Constraints				
PDU: RRC (Receive Ready Command)				
List Name	Field Name			Comments
	AD	F	N(R)	
S1	'03'H	'1'B	N_R	NR is a free variable

PDU Constraints				
PDU: RR (Receive Ready response)				
List Name	Field Name			Comments
	AD	F	N(R)	
S3	'03'H	'1'B	'001'B	

PDU Constraints		
PDU: HEX (Hex string in frame)		
List Name	Field Name	Comments
	Hstring	
S0	'01FF'H	Error Command!

Appendix 'C'

Protocol Implementation Conformance Statement (PICS)

In this annex we give an example of a PICS proforma. This example is taken from the [ISO8882-2]. PICS proforma is always completed by the client for testing of the IUT. The completed PICS should contain the options and capabilities of the IUT. The following notation is used in the PICS example included here:

C = Conditional
F = Forbidden
M = Mandatory
O = Optional
[] = At least one of a set of requirements are applicable
- = Not specified or not applicable

The IUT should always be tested according to the PICS and only for the capabilities claimed to have been implemented and indicated in PICS.

PICS				
Status in X.25 Standards (References)			GENERAL	Check (√) and enter value where applicable, (follow guidance).
ISO 7776	X.25 1984	X.25 1980		
[M]	[M]	[M]	1. Conformance Claim 1.1 Support ISO 7776 1.2 Support CCITT X.25 1984 LAPB 1.3 Support CCITT X.25 1980 LAPB	No ☐ Yes ☐ -Skip to 2. No ☐ Yes ☐ -Skip to 3. Yes ☐ No ☐ -Test suite not applicable. Skip to 3.
O (S0.)	-	-	2. Support DTE/DTE interworking[1]	No ☐ Yes ☐
M (S0.)	M (S2.1.1)	M (S2.1.1)	3. Support DTE/DCE interworking	Yes ☐ No ☐ -Test suite not applicable
			GENERAL PDU STRUCTURE	
M (S3.)	M (S2.2)	M (S2.2)	4. General frame structure supported	Yes ☐
M (S3.8)	M (S2.3.5.3)	M (S2.2.9)	5. Invalid frames recognised	Yes ☐
M (S3.9)	M (S2.2.10)	M (S2.2.10)	6. Frame abortion supported	Yes ☐
M (S3.10)	M (S2.2.11)	M (S2.2.11)	7. Interframe time fill supported	Yes ☐
O (S3.8)	O (S2.3.5.3)	-	8. May send non-octet aligned I frames	No ☐ Yes ☐

[1]: Testing DTE/DTE interworking is for further study.

Status in X.25 Standards (References)				
ISO 7776	X.25 1984	X.25 1980	GENERAL PDU STRUCTURE	Check (✓) and enter value where applicable, (follow guidance).
O (§3.8).	O (§2.3.5.3)	-	9. Receiving non-octet aligned frames causes, 9.1 error at Data-link layer,	No ☐ Yes ☐ -IUT reaction is: ____. skip to 10.
			9.2 Restart at Packet layer	No ☐ Yes ☐ -Skip to 10.
			9.3 No specific reaction from IUT	No ☐ Yes ☐
M (§3,§4.1.1)	M (§2.1.3)	M (§2.3.2.1)	10. Support modulo 8 operation	Yes ☐ No ☐ -Must check 11. as Yes.
O (§3,§4.1.1)	O (§2.1.3)	O (§2.3.2.1)	11. Support modulo 128 operation[1]	No ☐ Yes ☐
M (§4.2, §4.1.2.2.5)	M (§2.3.3, §2.3.2.2.6)	M (§2.3.3)	12. Poll/Final (P/F) bit supported on all frames	Yes ☐
			SYSTEM PARAMETERS	
C (§5.7.3)	C (§2.4.8.5)	C (§2.4.11.3)	13. Maximum number of bits in I frame (N1) supported if, 13.1 Modulo 8 was checked with SLP (N1≥1080)	Yes ☐ - Value: ____. Skip to 14.
C (§5.7.3)	C (§2.4.8.5)	-	13.2 Modulo 128 was checked with SLP (N1≥1088)	Yes ☐ - Value: ____. Skip to 14.
C	C	-	13.3 Modulo 8 was checked with MLP (N1≥1096)[2]	Yes ☐ - Value: ____. Skip to 14.
C	C	-	13.4 Modulo 128 was checked with MLP (N1≥1104)[2]	Yes ☐ - Value: ____.
M (§5.7.2)	M (§2.4.8.4)	M (§2.4.11.3)	14. Maximum number attempts to complete a transmission (N2)	Value: ____

[1]: Testing Mod 128 is for further study
[2]: Testing MLP is for further study

PICS				
Status in X.25 Standards (References)			SYSTEM PARAMETERS	Check (✓) and enter value where applicable, (follow guidance).
ISO 7776	X.25 1984	X.25 1980		
C (§5.7.4)	-	-	15. Maximum number of outstanding I frames (k) if 15.1 support DTE/DTE interworking was checked, (2 ≤ DTE_k ≤ 7)	Yes☐ -Value: _____. skip to 16.
C	-	-	15.2 support DTE/DTE interworking was checked with modulo 128 (2 ≤ DTE_k ≤ 127)*	Yes☐
C (§5.7.4)	C (§2.4.8.6)	C (§2.4.11.3)	15.3 support DTE/DCE interworking was checked, (2 ≤ DTE_k ≤ 7)	Yes☐ -Value: _____. skip to 16.
C (§5.7.4)	C (§2.4.8.6)	-	15.4 support DTE/DCE interworking was checked with modulo 128 (2 ≤ DTE_k ≤ 127)*	Yes☐
C (§5.7.1.1) M (§5.7.1.1)	C (§2.4.8.1) M (§2.4.8.1)	C (§2.4.11.1) M (§2.4.11.1)	16.1 Range of parameter, DTE's T1 16.2 Expected range of DCE's T1	Yes☐ minimum:__ maximum:__ minimum:__ maximum:__ checked 2. as Yes.
C (§5.7.1.2)	C (§2.3.3,	C (§2.4.8.2)	17. Range of parameter T2 for DTE operation, T1>T2	Yes☐ minimum:___ maximum:___
C (§5.7.1.2)	-	-	18. Range of parameter T2 for DCE operation in case of DTE/DTE interworking, T1>T2	Yes☐ minimum:___ maximum:___
O (§5.7.1.3)	M (§2.4.8.3)	-	19. Range of parameter T3 such that T3>T1	Yes☐ Value:_____
O (§5.7.1.4)	-	-	20. Range of parameter T4 such that T3≥T4	Yes☐ Value:_____ Skip to 14.

* : Testing Mod 128 is for further study

PICS

Status in X.25 Standards (References)			PROCEDURES	Check (✓) and enter value where applicable, (follow guidance).
ISO 7776	X.25 1984	X.25 1980		
C (§4.3.6, §5.3.3)	C (§2.3.4.6, §2.4.4,3)	C (§2.3.4.7, §2.4.5.3)	21. Disconnect procedure: Sends DISC and receives UA or DM then goes to disconnected phase.	Yes☐ No ☐ (see PIXIT item 10.)
M (§5.3.1, §5.3.3)	M (§2.3.4.6, §2.4.4,3)	M (§2.3.4.7, §2.4.5.3)	22. Disconnect procedure: Receives DISC and sends UA or DM then goes to disconnected phase.	Yes☐
[M] (§5.3.5(a) ..(b) ..(c))	[M] (§2.4.4.5.1, ..(2) ..(3))	M (§2.4.5.5.1) - -	23. Disconnect procedure: Resolves collision 23.1 After receiving UA 23.2 After sending UA 23.3 After timing out, waiting for UA	Yes☐ -Skip to 24. Yes☐ -Skip to 24. Yes
C (§4.3.5, §5.3.1)	C (§2.4.4 1, §2.4.7.2)	- (§2.4.5.1)	24. Link setup (reset) procedure: Sends SABM and receives UA.	Yes☐ No ☐
M (§5.3.1, §5.3.6)	M (§2.4.7.2, §2.4.4.2)	M (§2.3.4.6, §2.4.5.1)	25. Link setup (reset) procedure: Receives SABM and sends UA.	Yes☐
[M] (§5.3.5(a), ..(b), ..(c))	[M] (§2.4.4.5.1, ..(2) ..(3))	M (§2.4.5.5.1) - -	26. Link setup (reset): Resolves SABM collision 26.1 After receiving UA 26.2 After sending UA 26.3 After timing out, waiting for UA	Yes☐ -Skip to 27. Yes☐ -Skip to 27. Yes
O (§5.3.1)	O (§2.4.4.1, §2.4.4.4.2)	O (§2.4.5.6)	27. Link setup (reset) procedure: Sends DM to request SABM.	Yes☐ No ☐ (see PIXIT item 12)

PICS

Status in X.25 Standards (References)			PROCEDURES	Check (✓) and enter value where applicable, (follow guidance).
ISO 7776	X.25 1984	X.25 1980		
O (§5.3.1, §5.5) M (§5.3.7)	M (§2.4.4.1, §2.4.4.4.2) M (§2.4.4.7)	O (§2.4.5.6) -	28.1 Link setup (reset) procedure: Receive DM and sends SABM 28.2 Link setup (reset) procedure: Collision of DM with DM, is resolved by DTE.	Yes☐ No ☐-Skip to 29. Yes☐ No ☐
O (§4.4.2.1, §5.4.1) O (§5.4.1)	O (§2.4.5.1, §3.3,§4.4.3) O (§2.4.5,1)	O (§2.4.6.1, §3.3,§4.4.3) O (§2.3.5.4)	29. Information transfer: 29.1 DTE sends I frame for level 3 Restart/Reset. (see PIXIT item 8.) 29.2 DTE may send I frame for normal information transfer.	Yes☐ No ☐ Yes☐ No ☐
M (§5.4.2)	M (§2.4.5.2)	M (§2.4.6.2)	30. Information transfer: DTE can receive valid I frames within window (see PIXIT itsm 15).	Yes☐
[M] (§5.4.2) [M] (§5.4.2)	[M] (§2.4.5.2.1) [M] (§2.4.5.2.1)	[M] (§2.4.6.2.1) [M] (§2.4.6.2.1)	31. Information transfer: 31.1 Sends RR frames to acknowledge all I frames 31.2 Sends RR frames to acknowledge when I frames are not available.	No ☐ Yes☐-Skip to 32. No ☐ Yes☐
M (§5.4.5)	M (§2.4.5.5)	M (§2.4.6.4)	32. Information transfer: DTE can receive RR frame as acknowledgement to I frames.	Yes☐
C (§5.4.8)	C (§2.4.6.7)	C (§2.4.6.7)	33. Information transfer: DTE sends RNR frames when busy (see PIXIT item 9).	Yes☐ Yes☐
M (§5.4.7)	M (§2.4.5.7)	M (§2.4.6.6)	34. Information transfer: DTE can receive valid RNR frames as indication of DCE/ remote DTE busy condition.	Yes☐

PICS				
Status in X.25 Standards (References)			PROCEDURES	Check (√) and enter value where applicable, (follow guidance).
ISO 7776	X.25 1984	X.25 1980		
M (§5.3.4)	M (§2.4.5.4)	M (§2.4.6.3)	35. Information transfer: Sends REJ frame.	Yes ☐
M (§5.4.6)	M (§2.4.5.6)	M (§2.4.6.5)	36. Information transfer: Can receive REJ frame.	Yes ☐
M (§5.2)	M (§2.4.3)	M (§2.4.3)	37. Sends P-bit (P=1) in I frames to solicit response.	Yes ☐
O (§5.2) O (§5.3.4)	O (§2.4.3) O (§2.4.4.4.1)	O (§2.4.3) O (§2.4.5.4.1)	38.1 Sends P-bit (P=1) in supervisory frames to solicite response. 38.2 Responds with DM to supervisory command with P-bit received in disconnected phase.	Yes ☐ No ☐ Yes ☐ No ☐
M (§5.3.3, §5.3.6)	M (§2.4.4.3, §2.4.4.6)	O (§2.4.5.6)	39. Sends P-bit in DISC (P=1)	Yes ☐ No ☐
M (§5.3.1, §5.3.6)	M (§2.4.4.1, §2.4.4.6)	M (§2.4.5.6)	40. Sends P-bit in SABM (P=1)	Yes ☐ No ☐
M (§5.5, §5.6.2)	M (§2.4.6.1, §2.4.7.3)	M (§2.4.9.4, §2.4.10.1)	41. Frame reject condition: Sends FRMR frames	Yes ☐
M (§5.5, §5.6.2)	M (§2.4.6.1, §2.4.7.3)	M (§2.4.10.2, §2.4.9.4)	42. Frame reject condition: Receives FRMR as a request for link reset.	Yes ☐

PICS				
Status in X.25 Standards (References)			PROCEDURES	Check (√) and enter value where applicable. (follow guidance).
ISO 7776	X.25 1984	X.25 1980		
M (§5.4.1, §5.4.5)	M (§2.4.5.1, §2.4.5.9)	M (§2.4.6.1, §2.4.6.8)	43. Timer recovery procedure after sending I frame is supported.	Yes☐
M (§5.3.1) M (§5.3.1)	M (§2.4.4.1) M (§2.4.4.1)	M (§2.4.4.1) M (§2.4.4.1)	44. Timer recovery procedure after sending SABM SABM is supported. 44.1 After N2 attempts recovery is done at higher layer.	Yes☐ Yes☐ No☐ Specify:
M (§5.3.3) M (§5.3.3)	M (§2.4.4.3) M (§2.4.4.3)	M (§2.4.4.3) M (§2.4.4.3)	45. Timer recovery procedure after sending DISC is supported. 45.1 After N2 attempts recovery is done at higher layer.	Yes☐ Yes☐ No☐ Specify:
M (§5.6.2)	M (§2.4.7.3)	M (§2.4.9.4)	46. Timer recovery procedure after sending FRMR is supported.	Yes☐ No☐
O (§4.4.2.2)	-	-	47. Timer recovery procedure after sending REJ is supported.	Yes☐ No☐
O (§5.7.1.3)	M (§2.4.8.3)	-	48. Timer T3 procedure is supported.	No☐ Yes☐ -Specify:_____
M (§5.7.1.4, §5.3.2)	-	-	49. Timer parameter T4 is supported.	Yes☐ Yes☐ -Specify:_____

PICS				
Status In X.25 Standards (References)			PROCEDURES	Check (✓) and enter value where applicable, (follow guidance).
ISO 7776	X.25 1984	X.25 1980		
M (§1.,§5.)	M (§2.1)	M (§2.1)	50. Single Link Procedure (SLP) is supported.	Yes☐
O (§1,§5,§6)	O (§2.5)	-	51. Multilink Procedure is supported. (for further study)	No ☐ Yes☐
F	O (§2.2,§2.6, §2.7)	O (§2.4.7, §2.4.8)	52. LAP Procedure is supported. (Testing is outside scope of this standard)	No ☐ Yes☐
M (§5.)	M (§2.2,§2.3, §2.4)	M (§2.4.5, §2.4.9, §2.4.10)	53. LAPB Procedure is supported.	Yes☐ No ☐-Test suite not applicable.
O (§4.4.2.1)	-	-	54. Check point recovery is supported.	Yes☐ No ☐

Appendix 'D'

Protocol Implementation eXtra Information for Testing (PIXIT)

In this annex we give an example of a PIXIT proforma. This example is also taken from the [ISO8882-2]. PIXIT is normally used in conjunction with PICS. The informations which are deemed by the client and the test laboratory to be necessary to enable testing to be performed is documented in the PIXIT. In this example the PIXIT only provides the information about the testing environment of the IUT to the test laboratory, however PIXIT may also include the information that the client requires in order to prepare the SUT. The test laboratory incorporates this information into the PIXIT proforma that is supplied to the client for completion.

PIXIT	
GENERAL	Enter value or check where applicable.
1. The value of k used by IUT	value:___
2. The value of N1 for IUT	value:_____
3. The value of N2 parameter used by IUT	value:_____
4. The value of T1 parameter used by IUT	value:_____
5. The value of T2 parameter used by IUT	value:_____
6. The value of T3 parameter used by IUT (applicable if item 19 of PICS is Yes).	value:_____
7. The value of T4 paremeter used by IUT (applicable if item 20 of PICS is Yes).	value:_____
PROCEDURAL INFORMATION	
8. IUT will send the first I frame on performing information transfer tests.	Yes ☐ No ☐
9. IUT can be made to enter the Busy Condition	Yes ☐ No ☐ Specify minimum/maximum time:_____
10. IUT can initiate Link disconnection when required	Yes ☐ No ☐

PIXIT	
PROCEDURAL INFORMATION	Enter value or check where applicable.
11. State the time the tester must wait before determining that the IUT will not respond to a tester stimulus. This value will be tester parameter Td. Note: Td < T1.	value:___
12. IUT can be made to setup the link (applicable if PICS item 24 is Yes).	Yes ☐ No ☐ Specify how:_____
13. IUT can maintain the disconnected phase as a stable state for a period of time.	Yes ☐ No ☐ Specify minimum/maximum time:_____
14. The IUT can maintain the frame reject condition as a stable state.	Yes ☐ No ☐
I FRAME INFORMATION	
15. Specify a the contents of the information field that the IUT expects to receive in normal information transfer phase.	value:_____

References

[Brink87] E. Brinksma, "An Introduction to LOTOS", A tutorial on LOTOS specification language, May 1987.

[CaSi86] R. Castanet and R. Sijelmassi, "Methods and Semi-Automatic Tools for Preparing Distributed Testing", Protocol Specification, Testing and Verification, VI, North-Holland, IFIP.1986, Montreal, 177-188.

[Chow78] T. S. Chow, "Testing Software Design Modeled by Finite State Machines", IEEE Transactions on Software Engineering, SE-4, No. 3, 1978, pp. 178-187.

[Demil87] R. A. Demillo, W.M. McCracken, R.J. Martin, J.F. Passafiume, "Software Testing and Evaluation", Benjamin/Cummings, 1987.

[GuLo88] Renaud Guillemot and Luigi Logrippo, "Derivation of Useful Execution Trees from LOTOS Specifications Using an Interpreter", University of Ottawa, TR-88-13.

[ISO7776] ISO/TC97/SC21 "Description of the 1984 X.25 LAPB-Compatible DTE Data Link Procedures", IS7776, August 1984.

[ISO8073] ISO/TC97/SC21 "OSI Connection Oriented Transport Protocol Specification", DP8073, April 1983.

[ISO8807] ISO/TC97/SC21 "A FDT based on the Temporal Ordering of Observational Behaviour - LOTOS", DIS8807, 1987.

[ISO8824] ISO/IEC JTC1/SC21 "Specification of Abstract Syntax Notation One (ASN-1)", 1988.

[ISO8825] ISO/IEC JTC1/SC21 "Specification of Encoding Rules for Abstract Syntax Notation One (ASN-1)", 1988.

[ISO9646-1] ISO/IEC JTC1/SC21 "OSI Conformance Testing Methodology and Framework Part1: General Concepts", DP9646-1, December 1987 (Vancouver Output).

[ISO9646-2] ISO/IEC JTC1/SC21 "OSI Conformance Testing Methodology and Framework Part2: Abstract Test Suite Specification" - (Integral Part), DP9646-2, December 1987 (Vancouver Output)

[ISO9646-2a] ISO/IEC JTC1/SC21 "OSI Conformance Testing Methodology and Framework Part2: Abstract Test Suite Specification" - (Annex E & F), DP9646-2, December 1987 (Vancouver Output)

[ISO9646-3] ISO/IEC JTC1/SC21 "OSI Conformance Testing Methodology and Framework Part3: The Tree and Tabular Combined Notation", DP9646-3, July 1988 (Stockholm Input).

[ISO9646-4] ISO/IEC JTC1/SC21 "OSI Conformance Testing Methodology and Framework Part4: Test Realization", DP9646-4, May 1988 (Eastleigh Output).

[ISO9646-5] ISO/IEC JTC1/SC21 "OSI Conformance Testing Methodology and Framework Part5: Requirements on Test Laboratories and Clients for the Conformance Assessment Process", DP9646-5, May 1988 (Eastleigh Output).

[ISO8882-2] ISO/IEC JTC1/SC6 "X.25-DTE Data Link Layer Test Suite", DP8882-2, April 1987.

[ISO8882-3] ISO/IEC JTC1/SC6 "X.25-DTE Packet Layer Test Suite", DP8882-3, May 1988.

[ISO9074] ISO/TC97/SC21 "A FDT based on Extended Finite State Transition Model", DIS9074, 1987.

[ITEX88] ITEX "ITEX Test Suite Development Environment, An Introductory Overview", Swedish Telecom, Teletest, S-123 86 Farsta, Sweden, 1988.

[Matth86] R. S. Matthews, K.H. Muralidhar, M.K. Schumacher, "Conformance Testing: Operational Aspects, Tools, and Experiences", Protocol Specification, Testing and Verification, VI, North-Holland, IFIP 1986.

[Myers79] G. J. Meyers, "The Art of Software testing", John Wiley & Sons, 1979.

[Panzl78] D. J. Panzl, "Automatic Software Test Drivers", IEEE Computer, (pp 44-50), April 1978.

[Prob82] R. L. Probert, "Grey-Box (design based) Testing Techniques", Proceedings of 15th Hawaii International Conference on System Science, pp. 94-102, 1982.

[Prob88a] R. L. Probert, H. Ural and M. Hornbeek, "An Integrated Software Environment for Developing and Validating Standardized Conformance Tests", Protocol Specification, Testing and Verifivation, VIII, North-Holland, IFIP 1988, Atlantic City, pp. 242-254.

[Prob88b] R. L. Probert, "Towards. A Knowledge-Based Model For Conformance Test Results Analysis", Invited Paper, International Workshop on Protocol Testing Systems, Vancouver, B.C., October 1988.

[PrAk88] R. L. Probert and S. Akhavan-Sarraf, "Abstract Conformance Test Results for Open Systems", Second ISIIS, Interoperability Technology Association for Information Processing, Tokyo, Japan, November 1988.

[Rayn87] D. Rayner, "OSI Conformance Testing - Tutorial", Computer Networks and ISDN Systems, North-Holland, 1987.

[Sarik85] B. Sarikaya and G. v. Bochmann, "Obtaining Normal Form Specifications for Protocols", Proceedings of COMNET 85, Budapest Hungary, October 1985, pp. 6.133 - 6.149.

[SDL85] CCCITT, Functional Specification and Description Language (SDL) Recommendation Z.100-Z.104, 1985.

[Trusc88] C. Truscott, "A Conceptual Model for Test Result Analysis", Internal Document, Protocols Research Group, University of Ottawa, Ottawa, Ontario, Canada.

[Turn87] D. Turner, "Modelling of Expressed System Behaviour Towards Characterization of the Conformance Testing Problem", Master of Computer Science Thesis, University of Ottawa, Ottawa, Ontario, Canada.

[TuUr88] D. Turner, H. Ural, "Modelling of Expressed System Behaviour Towards Characterization of the Conformance Testing Problem", IEEE International Phoenix Conference on Computers and Communications, Scottsdale, Arizona, March 1988.

[Ural87] H. Ural, "A Test Derivation Method for Protocol Conformance Testing", Protocol Specification, Testing and Verification, VII, North-Holland, IFIP.1987, Zurich, June 1987.

[Wiles86] A. Wiles, "ITEX - An Interactive TTCN Editor and Executer", Proc. IEEE GlobeCom 1986.