Application of the Ferry Clip Approach to Multi-party and Interoperability Testing

By

HENDRA DANY

B.Sc., University of Toronto, Canada, 1982

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(DEPARTMENT OF COMPUTER SCIENCE)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 1990

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date August 28, 1990

Abstract

As communications protocols are becoming more complex and sophisticated, developing a test system that has the ability to provide a controlled environment for comprehensive protocol testing is essential to achieve a "real open system". This thesis advocates the need for a *multi-party* test method as currently identified by ISO, and discusses two important aspects of protocol testing: *Conformance* and *Interoperability*. They are complementary to each other and are necessary to ensure the conformity and interoperability of a protocol implementation. The proposed ferry clip based test architecture is presented. Both the concepts and design principles employed to achieve a flexible and generalized test system and the specific components which comprise the Ferry Clip based Test System are described. The test system is general and flexible not only with respect to the test configurations and test methods but also with respect to the protocol to be tested, the system under test, and the underlying communication system. Applications of the ferry clip approach to multi-party conformance and interoperability testing are discussed, followed by an example of MHS conformance testing which demonstrates the applicability of the ferry clip approach to multi-party testing.

Contents

List of Tables

List of Figures

Terms and Acronyms

ASN.1	Abstract Syntax Notation One
ASP	Abstract Service Primitive
AFC	Active Ferry Clip
BER	Basic Encoding Rules
CCITT	International Telegraph and Telephone Consulative Committee
FCP	Ferry Control Protocol
FCTS	Ferry Clip based Test System
FSM	Finite State Machine
FTMP	Ferry Transfer Medium Protocol
FY-CNTL	Ferry Control
FY-DATA	Ferry Data
IPC	Interprocess Communication
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
IUT	Implementation Under Test
LAPB	Link Access Procedure B
LMAP	Lower Mapping Module
LT	Lower Tester
MHS	Message Handling System
MPTM	Multi-party Test Method
MTAE	Message Transfer Agent Entity
OSI	Open Systems Interconnection
PCO	Points of Control and Observation
PDU	Protocol Data Unit
PFC	Passive Ferry Clip
PLP	X.25 Packet Layer Protocol
PTE	Protocol Testing Environment
SAP	Service Access Point
SIA	Service Interface Adapter

SUT	System Under Test
TCP/IP	Transmission Control Protocol/Internet Protocol
TM	Test Manager
TMP	Test Management Protocol
TP	Transaction Processing Protocol
TTCN	Tree and Tabular Combined Notation
UBC-IDACOM	University of British Columbia–IDACOM Electronics
UAE	User Agent Entity
UT	Upper Tester
X.25	Packet-switching protocol and standard (CCITT)
X.400	Message Handling Systems (CCITT)

Acknowledgement

First of all, I would like to express my sincere thanks to my thesis supervisors, Dr. Samuel T. Chanson and Dr. Son Vuong, for their guidance, commitment, and support throughout the course of my research.

I am indebted to a number of people whom I have consulted on many occasions during my research work. Especially, I would like to thank Helen see, Bernard Lee, Brian Smith, Dennis Lo, Viola Lee and Mike Sample for always finding the time to help me when I most needed it.

I would also like to extend my gratitude to my best friend, Carson Woo, who is always willing to listen and offer his constructive opinion. I hope I will continue to benefit from his friendship and wisdom.

I gratefully acknowledge the financial assistance and computing facilities provided by the IDACOM Electronics and Department of Computer Science, University of British Columbia during my graduate studies.

Last but not least, a heartfelt thanks to my parents for their love and support, and for always giving me the freedom to pursue my own goals and interests.

Chapter 1

Introduction

This chapter presents the motivation of the research reported in this thesis. An introduction to the role of protocols and the role of protocol testing help set the the discussion in the appropriate context. The importance of the two different aspects of protocol testing is discussed, followed by the thesis outline.

1.1 Motivation

As Open Systems Interconnection (OSI) protocol standards mature, many new implementations based on these standards are being developed and appear in the industry. Therefore, facilities to test these protocol implementations must be provided in order to evaluate whether they conform to the industry-standard specifications, pass appropriate conformance tests, and are able to interoperate effectively in a multi-vendor environment.

Protocol testing plays a key role in this process of evaluation for it helps to enhances interoperability between systems from multiple vendors and service providers. To meet the user requirements that products are able to interwork effectively, ISO and CCITT are jointly developing conformance testing standards with the aim that OSI protocols would be consistently tested worldwide. The current testing standards [ISO-1] focus primarily on single-party testing, and it was only recently that the need for a multi-party test method [ISO-2] has been identified and investigated by ISO.

The need for a multi-party testing arises in the situations where an action (or an instance of communication) of the protocol entity to be tested causes subsequent actions at other protocol entities to take place. The protocol entities which are influenced by the action may be distributed across the network involving multiple independent parties. In addition, protocol implementation of this type usually does not offer a directly usable service; that is there are no explicit points of control and observation within or above the protocol implementations. Under these circumstances, the results of the action may only be observed and reported by third parties. Hence, the requirement of testing such a protocol suggests the need for a test method which may involve multiple peer entities distributed among several systems, each communicating with the Implementation Under Test (IUT) through a unique association or connection, *i.e.,* the IUT concurrently utilizes more than one association or connection for communicating with its peer entities. This also implies that the test method may require both direct and indirect control and observation in order to carry out test control and management functions, *i.e.,* to achieve test co-ordination procedures and observation of the IUT. Implementations of Message Handling System (MHS), Routing Protocol, ISDN, Distributed Transaction Processing and Directory Services are typical examples which require multi-party testing capability.

Testing OSI protocol implementations before being deployed to production fields provides a measure of confidence in them. A protocol implementation is said to conform to the relevant standards if conformance testing yields a positive verdict. This, however, does not necessary guarantee it is able to interwork properly when deployed as part of a complete working system. Due to the complexity of protocols, rigorous conformance testing is faced with practical limits on both technical and economical grounds. In addition, there is a growing demand from the marketplace to have products that are interoperable in a multi-vendor environment. Thus, interoperability testing is receiving attention in the field of protocol testing as it is a direct and pragmatic test approach to assess the interoperability of the products users install [BON89].

The Ferry Clip [ZENG88a, ZENG88b, ZENG89] concept was recently introduced as a technique for realizing all the ISO defined abstract test methods. Due to its recent introduction, little work has been done to study the applicability of adopting the ferry

clip test approach to multi-party testing. The purpose of this research is investigate how the ferry clip test approach can be extended for new area of protocol testing, *i.e.*, *multi-party conformance testing* and *multi-party interoperability testing*; and explore the possible benefits with this approach. To accomplish the purpose of this research, one of the most important goals is to design and implement a flexible and generalized Ferry Clip based Test System which provides a controlled environment for comprehensive protocol testing. This leads us to examine the design issues and the architectural principles that are required to guide the test system design and implementation.

1.2 The Role of Protocols

As computer and communication networks are becoming more sophisticated and more pervasive. Users are increasingly demanding to have equipment and software from different vendors interconnected forming seamless networks of diverse components that work reliably and efficiently. They want to manage their networks easily and most importantly, add new components and replace existing ones without making their previous investment obsolete [AHO90].

To satisfy these user requirements, products from a single vendor must be able to communicate with products manufactured by other vendors. It is no longer acceptable to have proprietary products that can only communicate amongst themselves. The translation of these user requirements to products is realized through communication protocols. A protocol is a set of rules or conventions by which two components or entities within a network communicates with one another.

Communication protocols play a major role in fostering the era of *open systems* as it provides the means to interconnect disparate software systems and it also allows migration from predominantly hierarchical (mainframe-centered) computer communications to nonhierarchical (client/server centered), peer-to-peer communications. The protocols will make possible a more graceful, predictable migration to tomorrow's world of communication [BERN90].

1.3 The Role of Protocol Testing

To describe the role of protocol testing in the appropriate context, it is useful to examine the system development life cycle of a protocol implementation (*e.g.*, X.25, MHS) which evolves from system requirement analysis phase to maintenance testing.

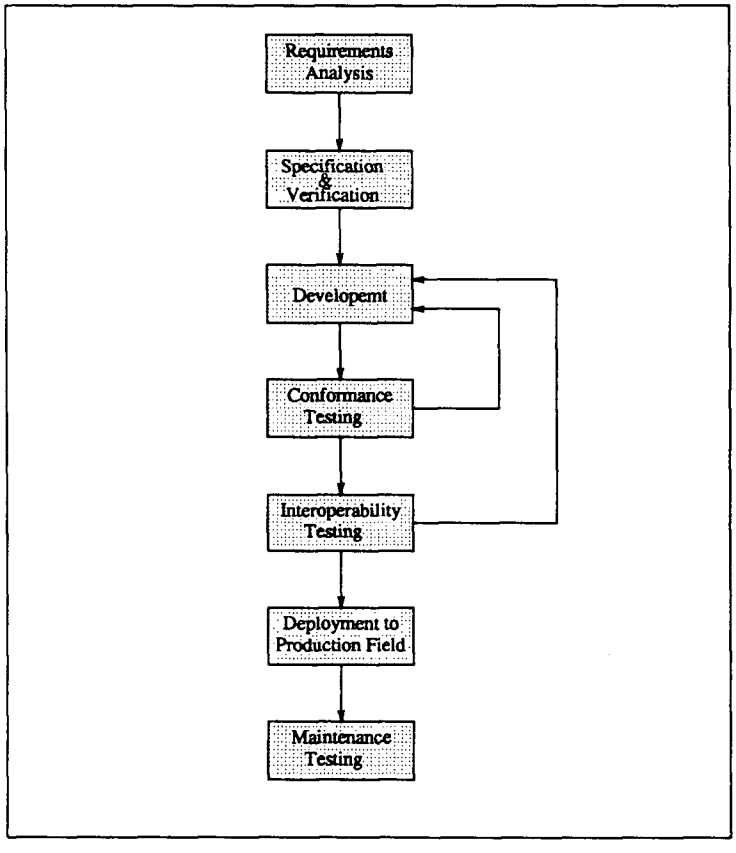


Figure 1.1: The development life cycle of a protocol implementation

As shown in Figure 1.1[BERN90], the first phase is *system requirement analysis.* In this phase, the overall behaviors and features of a protocol are specified informally in high-level representation. This phase is followed by a *Specification and Verification* phase which involves developing a complete functional specification (*i.e.*, Service and Protocol specifications) of the protocol. The *service specification* defines the communication services provided to the user entities residing in the layer above. The *protocol specification*

defines the protocol entity's behavior which is required to provide the communication services defined in the service specification.

Before a protocol is implemented, the specifications must be subject to *verification* to ensure that they provide the expected services and that it is free of logical and functional errors. In the simplest case, verification ensures that the protocol has the desired *safety* and *liveness* properties. The safety properties of a protocol address the detection of deadlocks and functional errors in its design; while the liveness properties address the detection of livelocks, that is no progress toward providing the desired services will take place. In other words, safety assures that *bad things* (*e.g.,* deadlocks) *will not happen*, and liveness assures that *good things* (*e.g.,* connection will be established) *will happen.* Recent research also deals with the verification of the semantic correctness of a protocol specification. These protocol specifications are then used to guide the development of protocol implementations.

In *Conformance testing*, the protocol to be tested is referred to as the *implementation under test* (IUT), which is viewed as a *black box* whose behavior is characterized by a set of observable actions that are generated by applying a set of externally controllable test inputs. The task of conformance testing is to verify whether the external behavior of the IUT complies with the requirements as defined in its specifications. Its purpose is to increase the probability that different implementations are able to interwork in an open communication architecture. However, conformance testing does not guarantee that the implementation fully conforms to its specifications as it detects errors rather than their absence, aspects that are not specified will generally not be tested.

Interoperability testing supplements conformance testing and provides another level of assurance that the implementation will be able to interoperate in a multi-vendor environment. Its task is to verify the end-to-end behavior of a protocol implementation by determining whether different vendors' implementations interoperate under the conditions simulated in a specified test configuration. This tests assumes that each vendor's implementation has passed the conformance tests as described above.

Even after the protocol implementation has passed both the conformance and in-

teroperability tests, there is no guarantee that no errors will be encountered when it is deployed to the actual production field. Errors may arise due to untested configurations, or any incompatibilities between the implementation and the local operating environment. Thus, *maintenance testing* may still be needed to troubleshoot and resolve any protocol and functional errors while installing new services or equipment, upgrading software, etc [BERT90].

Through the process of protocol verification, conformance testing, and interoperability testing, reliable communication products that satisfy user demands can be achieved. Without reliable, efficient, and interoperable products, users cannot embrace open standards. Consequently, the ultimate goal of OSI – *global connectivity* – will not be achieved as users are forced to depend on proprietary solutions [AHO90].

1.4 Conformance Testing versus Interoperability Testing

To achieve the goal of a truly "open system interconnection", two different aspects of protocol testing are necessary:

- *Conformance Testing:* As stated in [ISO-1], conformance testing verifies whether the external behavior of a protocol implementation complies with the relevant OSI protocol standards. The purpose of this testing is to increase the "probability" that different implementations are able to interwork.

- *Interoperability Testing:* This testing verifies the end-to-end behavior of a protocol implementation. It enhances confidence that the tested protocol implementation will interwork with different implementations of the same protocol assuming that they comply to the same protocol standards. While conformance testing may be done on a single layer basis, interoperability testing is always done with stacks of protocol layers representing actual interconnections between two or more different implementations.

Although it is generally agreed that conformance testing facilitates interworking, this does not necessarily imply that conformance guarantees interoperability. Similarly, interoperability does not necessarily guarantee conformance [BON89]. This raises the issue of cost-benefit relationship between conformance testing and interoperability testing, which requires further investigations. However, the need for different aspects of protocol testing are largely driven by the growing demand from the marketplace to have products not only conform to the standards but are also able to interoperate in a multi-vendor environment. To meet this demand, conformance testing or interoperability testing alone is insufficient for several reasons. First, due to the complexity of protocols and the fact that these protocols are usually not formally specified, they are likely to contain ambiguities and are subject to misinterpretations. Thus, it is possible for two different implementations of the same protocol that conform to the standards but are not able to interoperate; likewise, it is possible for two implementations to be interoperable but do not conform to the standards. Second, protocol specifications usually define only the functional requirements and may not address the end-to-end behavior in a complex configuration [BERT90]. Third, each protocol usually provides a range of options that the protocol is able to support. These options may result in incompatibility between different implementations. Interoperability testing helps to ensure that a common subset of options are supported by the communicating entities, thereby eliminating the mutual incompatibility. Thus, conformance and interoperability testing are both necessary and complementary if OSI standards are to be embraced by the users. However, it should be pointed out that there is no absolute guarantee for an implementation that has passed both conformance and interoperability testing not to encounter any problem after being deployed to the real operating environment. This is possible due to the fact that interoperability testing is conducted in a simulated end-to-end testing environment, any untested configurations, unexpected conditions, or local peculiarities about the actual operating environment may introduce unforeseen errors.

1.5 Thesis Outline

This thesis first identifies the need for a multi-party test method and describes the research objectives. A brief introduction to the role of protocol and protocol testing is presented to set the discussion in this thesis in the appropriate context. This is followed by a discussion on conformance testing versus interoperability testing. The importance of having the two different aspects of protocol testing is emphasized as they help ensure interoperability.

Chapter 2 describes the evolution of the ferry clip concept and characterizes the ferry clip test architecture. The application of the ferry clip to realize the ISO abstract test methods are discussed. The error detection power of the ferry clip test architecture is compared to that of the conventional test approaches.

Chapter 3 presents the design goals and architectural principles that are used to guide the implementation of a generalized ferry clip based test system. The major components which comprise the ferry clip based test system are described. The use of ASN.1 for representing the information exchanged between the test system and the SUT is presented, followed by a discussion on the derivation of executable test cases based on the dynamic behavior of a TTCN abstract test suite.

Chapter 4 describes the implementation of the proposed ferry clip based test system in the OSI-PTE environment. The scheme for structuring the ferry clip components within the OSI-PTE, and the implementation of the ASN.1 support tools and the derivation of executable test suite based on a TTCN abstract test suite are described.

Chapter 5 addresses the general issues and requirements of multi-party conformance testing. The remainder of this chapter presents an overview of the Message Handling System (MHS), identifies the testing requirements, and illustrates how the proposed ferry clip based test architecture can be configured to meet these requirements and achieve the purpose of multi-party conformance testing within the scope of OSI. The experiences gained and results obtained from the conformance testing of a MHS implementation are presented.

Chapter 6 addresses issues that are related to interoperability testing, and highlights why interoperability testing enhances the confidence that different protocol implementations will interwork. The remainder of this chapter presents how the ferry clip based test architecture can be extended to perform multi-party interoperability testing for a single- or multi-layer IUT.

Chapter 7 summarizes the thesis and highlights some of the areas that remain open for further research.

An attempt has been made to present the content of this thesis in a reasonably self-contained fashion, but familiarity with the basic concepts of protocol testing is assumed. In particular, an understanding of the OSI Reference Model, ASN.1 abstract syntax and its associated encoding/decoding rules, and the concepts of Message Handling System (MHS) would be useful.

Chapter 2

The Ferry Clip Approaches in Protocol Testing

This chapter describes the evolution of the ferry clip concept and clarifies some of the terminologies used in this thesis. The application of the ferry clip to realization of the ISO abstract test methods are discussed and the advantages of the ferry clip over conventional test approaches are highlighted. The error detection capability provided by the ferry clip test approaches is examined and compared to that of the conventional realization test approaches.

2.1 The Evolution of the Ferry Clip Concept

The *ferry principle* was originally introduced by Zeng [ZENG86] in 1985 with the aims to reduce the amount of test software to be included in the SUT, and to simplify and enhance synchronization between the upper tester (UT) and lower tester (LT). The initial application of the ferry principle was to transfer test data over a ferry channel between the UT and the IUT. This allowed the UT to be moved from the system under test (SUT) into the same machine where the lower tester (LT) resides, thereby reducing the amount of test software in the SUT, and simplifying the synchronization problem between the UT and LT. The original ferry model has a limitation, that is the point of control and observation (PCO) at the lower service interface of the IUT is obtained indirectly through

the service provider at the test system; therefore, application of the ferry principle has been extended so that it is possible to remotely access both the upper and lower service interfaces of the IUT. This new application has been termed the *ferry clip* [ZENG88a].

The term "ferry" was borrowed from real life to express the concept of transparently shipping test data between a remote tester and an IUT via "ferryboats" [ZENG86]. The term *"ferry principle"* is used to describe the employment of a bidirectional data transfer channel to deliver test data between a remote tester and an IUT; while the *"ferry clip"* is used as a generic term for all applications of the ferry principle. Hence, all test approaches that apply the ferry principle for the realization of the ISO abstract test methods will hereafter be referred to as *ferry clip test approaches.* The name "ferry clip" comes from the fact that a ferry clip has two arms that clamp on an IUT like a "clip", and these two arms can be utilized to access the upper and/or the lower service interfaces of an IUT. With this classification, the original application of the ferry principle in a distributed test system can be viewed as an extension of the ferry clip application where the ferry clip is only used to access the upper service interface of an IUT (see Figure 2.1).

To test an IUT, two ferry clips are required: the *active ferry clip* (AFC) in the test system, and the *passive ferry clip* (PFC) in the SUT. The arms of the passive ferry clip are attached to the upper and lower service interfaces (if they are both accessible) of the IUT to allow the testers to have remote control and observation of each service interface. The arms of the active ferry clip are attached to the UT and LT in the test system.

Figure 2.1 illustrates an application of the ferry clip approach to realize the distributed or coordinated test methods. In this test configuration, the passive clip is used only for remote access to the upper service interface of the IUT. The ferry clips uses the *ferry control protocol* (FCP) [ZENG89] to provide a transparent data transfer between their users (*i.e.,* the IUT and UT). To allow the exchange of ferry control protocol data units between the two ferry clips, a Ferry Transfer Service (FTP) [ZENG89] is needed. This service can utilize any existing reliable data transfer services (*e.g.,* X.25), which shall hereafter be referred to as *ferry transfer medium protocol* (FTMP), available in the test system and SUT.

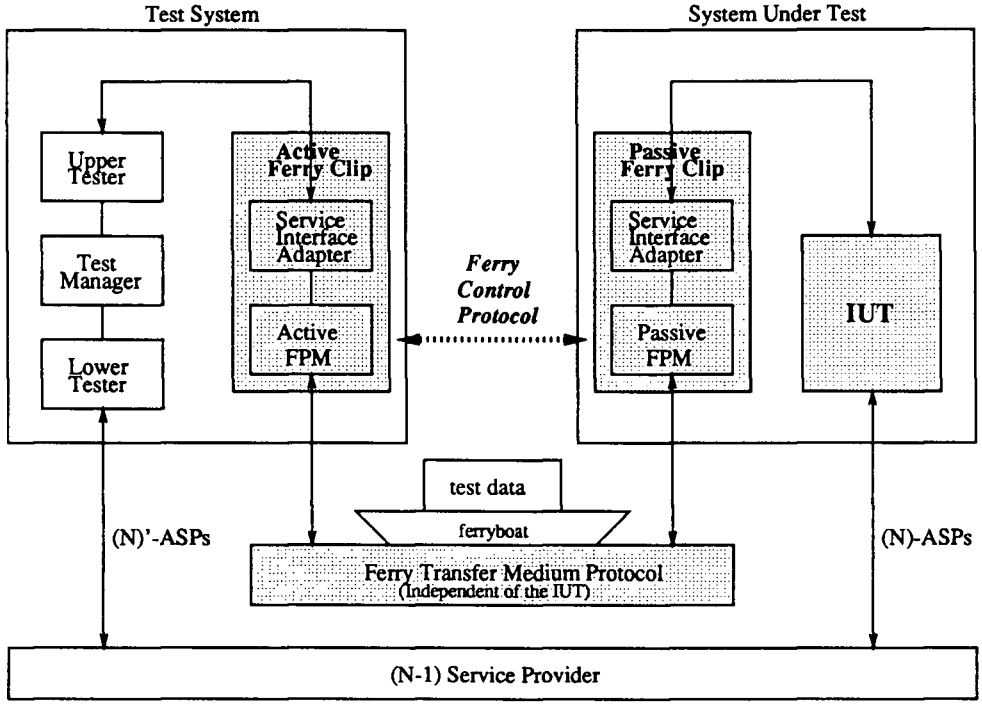


Figure 2.1: A Ferry Clip Test Configuration

Since the representation of abstract service primitives (ASPs) is IUT dependent, *i.e.*, different IUT implementations may represent ASPs in different ways, a ferry Service Interface Adapter is needed to convert IUT service primitives between IUT-dependent and tester dependent formats. The problem of mapping ASPs to specific IUT-dependent format is not unique to the ferry clip test approaches; it is common to all test approaches whenever direct access to a service interface is needed. For example, a service interface converter is required for the astride test responder in the conventional distributed test approach [ZENG89].

The ferry clip test approaches offer many advantages as compared to conventional realizations of the ISO abstract test methods. It is more powerful, more flexible and simpler to implement. These characteristics are discussed in detail in [ZENG89]. The local test method has been considered the most powerful abstract test method identified by the ISO. This test method is preferred if both service interfaces of the IUT are ac-

cessible, but it has never been considered a practical approach because of the difficulties associated with implementing the UT and LT in the SUT. With the application of ferry clip, this method has been made viable for third-party conformance testing as well as in-house diagnostic testing as illustrated in [PAR89].

2.2 Realization of the ISO Abstract Test Methods

There are four abstract test methods defined by ISO [ISO-1], namely the *Local, Distributed, Coordinated and Remote* test methods. For each test method, there exists three variant forms: *single-layer, multi-layer and embedded.* These test methods fall into two main classes, *Local* and *External*, with respect to the Points of Control and Observation (PCOs) above and/or below the service boundary of the IUT. Furthermore, the three types of external test methods, *i.e.*, Distributed, Coordinated and Remote, vary according to their ability to express test coordination procedures between the *lower tester* (LT) and *upper tester* (UT). Figure 2.2 [ISO-1] shows the conventional test architectures for the realization of these four abstract test methods.

The purpose for the standardization of these test methods is to enable test suite designers to use the most appropriate method depending on the circumstances such as the accessibility of the IUT interfaces, rather than to restrict them to a few inappropriate test methods. However, the distinctive difference of each test method together with the possible misinterpretation of the OSI standards may lead a test system designer to believe that there would have to be a different type of test system, one for each distinct test method, in order to perform tests in accordance with the OSI regulations.

The existence of different test methods does not necessarily imply that a different test tool is required for the realization of each abstract test method. Instead, it is desirable in practice that a test tool should not restrict, but rather should provide the flexibility of selecting a most appropriate test method within a single test tool for the purpose of conformance testing. An applicable test method could be determined based on the availability of abstract test suites and the accessibility of IUT interfaces.

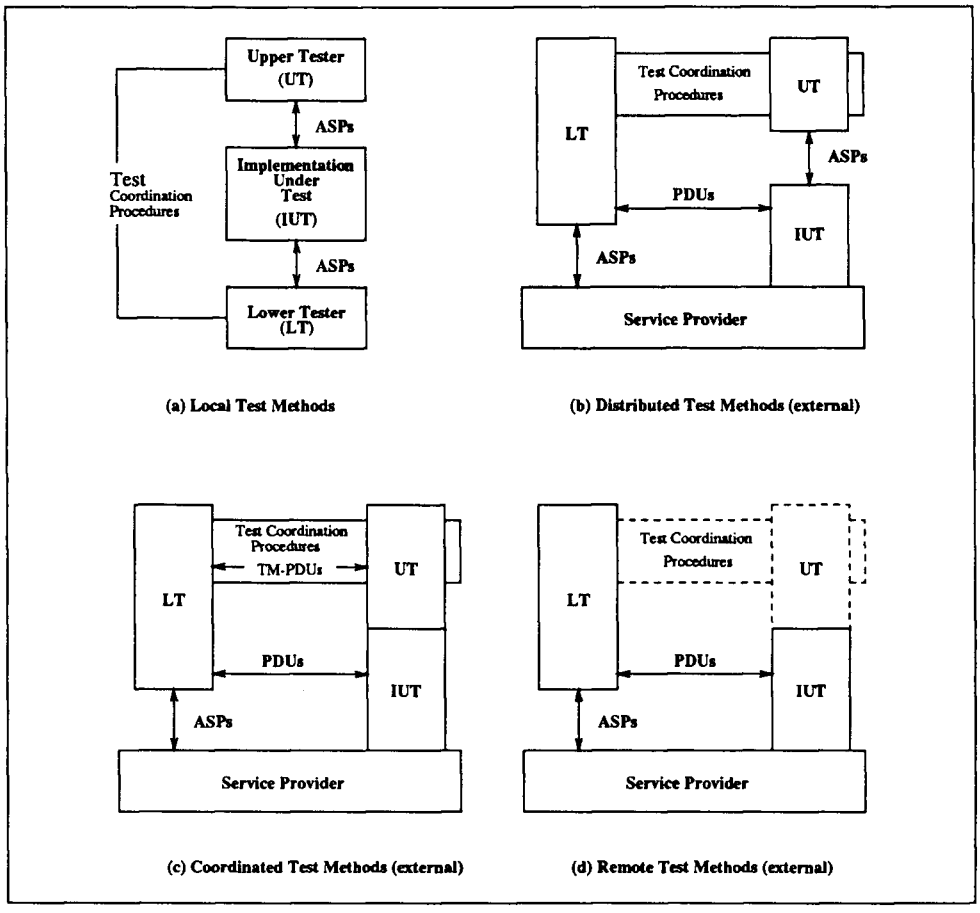


Figure 2.2: ISO Abstract Test Methods for Conformance Testing

For the realization of OSI abstract test methods within a single test tool, a Ferry Clip based test architecture is proposed as shown in Figure 2.3. Compared with conventional realizations of the ISO abstract test methods, the use of the ferry clip approach is considered to be superior in that it is more powerful, more flexible, and simpler to implement. These advantages are discussed in detail in [ZENG89].

The following subsections present the approaches to realize the ISO abstract test methods using the ferry clip based test architecture. Each test method can be realized by configuring the ferry clip to achieve different types of access to the upper and lower IUT service interfaces without the loss of desired functionality on the system under test

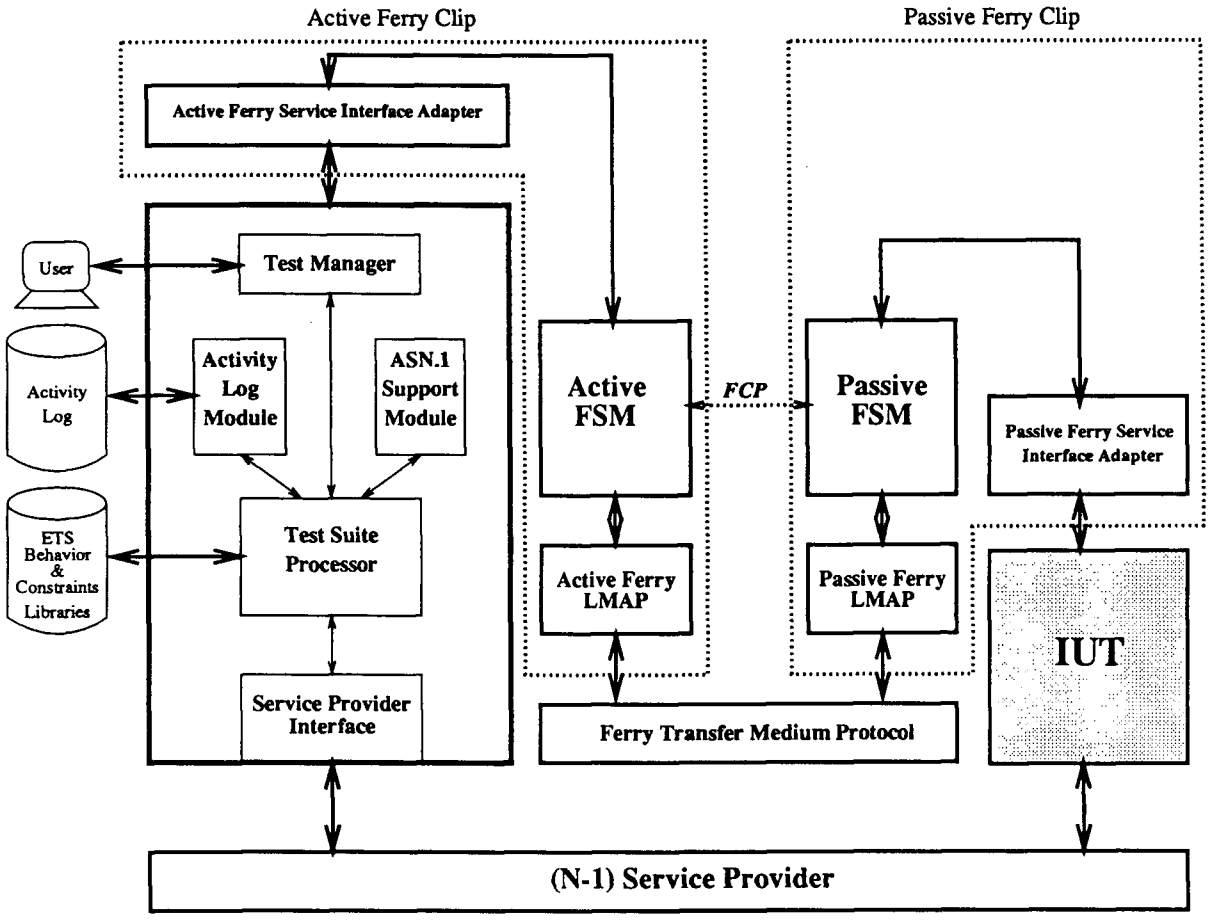


Figure 2.3: Overview of the Ferry Clip based Test Architecture

(SUT).

2.2.1 Realization of the Local Test Method

The *local* test method is the most powerful abstract test method of the four identified by the ISO for diagnostic testing purposes. Since the tester has access to both the upper and lower service interfaces of the IUT, this test architecture offers a complete error detection capability. With the passive ferry clip (PFC) residing in the SUT, and the active ferry clip (AFC), LT and UT located in the test system, the UT and LT can remotely control and observe both the upper and the lower service interfaces of the IUT.

2.2.2 Realization of the Distributed or Coordinated Test Methods

To realize the *distributed* or the *coordinated* test method, the ferry clip test approach only makes use of the passive ferry clip to control and observe the upper IUT service interface. Access to the lower IUT service interface is achieved indirectly through the underlying communication service provider.

The difference between the distributed and coordinated test methods is in the level of standardization imposed upon the synchronization procedures between the upper and lower testers, and in the requirements on the access to the service interface above the IUT. The coordinated test method uses a set of *test management protocols* (TMP) to achieve coordination between the upper and lower testers; whereas the upper tester in the distributed method directly accesses the upper service interface of the IUT. One noticeable drawback of the coordinated test method is that separate TMPs must be defined for each protocol layer to be tested.

If ferry clip test approach is used in the realization of these test methods, the difference between the distributed and coordinated test methods is only visible within the test system, but is transparent to the SUT.

2.2.3 Realization of the Remote Test Method

In the realization of the *remote* test method, the ferry clip test approach only makes use of the passive ferry clip to control and observe the lower service interface of the IUT. Since this approach has direct access to the lower IUT service interface, it is possible to provoke errors by sending test events which cannot be handled by the conventional realization of the remote test method. Therefore, the ferry clip approach is regarded to be more powerful than its conventional counterpart.

Thus, the use of ferry clip test approach for the realization of the ISO abstract test methods renders a uniform platform to all aspects of OSI protocol testing.

2.2.4 Test Architecture and Error Detection Capability

In any protocol testing activity, it is assumed that the tester is able to determine whether a given input/output interaction sequence observed from the IUT is valid or not. Error(s) are said to occur in an IUT if the observed sequence is not valid, *i.e.*, it does not conform to the protocol specification. The error detection capability of protocol testers depends on the test architecture, test suite applied, and the design of the testers [SARI89]. In this section, we discuss the error detection capability of the *local* and *external* test architectures as compared to the *ferry clip* based test architecture, without addressing issues that are related to the selection of test cases. Although the selection of test cases is an important problem since the applied test inputs determine to a large extent whether and what type of errors can be detected, the test case selection problem can be treated separately from the problem of deciding whether the IUT exhibits behavior conforming to the specification for a particular test case [BOCH89].

In the local test architecture, the tester has a global view of the interactions performed by the IUT. Thus, all possible erroneous behavior can be detected in principle, as long as the applied test input selected leads the IUT to exhibit these behavior.

In the conventional external test architectures, however, the upper and lower testers of, say, a distributed test architecture only have a partial view of the interactions per-

formed by the IUT. Since each tester only observes the interactions at one of the interfaces, the error detection power is limited. It has been shown that *local observers*, observing only the interactions at one of the interfaces of the IUT, do not detect all errors [BOCH89]. It is conceivable that an IUT which exhibits a faulty behavior, may go undetected by a local observer since the sequence of interactions observed by this local observer may simply conform to its reference specification. The error detection power of local observers in most test systems can be improved by using a so-called global analyzer which bases its error detection diagnosis solely on information received from the local observers [BOCH89].

In the ferry clip test architecture shown in Figure 2.3, the tester controls all test input to the IUT and observes all behavior exhibited by the IUT. Thus, it provides greater error detection power as compared with the conventional realization of external test architectures. However, one potential disadvantage of the ferry clip approach is that the relative order of interactions at the upper and the lower interfaces of the IUT may not be known to the tester because of the communication delays between the remote tester and the SUT. One way to reduce the timing uncertainty is to obtain more precise timing information about the interactions occurring within the SUT by using a timestamp derived from the local clock within the SUT. The clocks in the SUT and the test system must be well synchronized so that certain errors related to the relative timing between interactions at the upper and lower interfaces can be detected. However, clock synchronization in a distributed environment is a well-known nontrivial problem [LAMP78].

Chapter 3

System Architecture Design Overview

An architecture for test systems consists of a style of implementation based on carefully chosen design principles, techniques for applying these principles to test systems, and tools that support those techniques. This chapter identifies the design goals and objectives, and describes the design principles that are used to guide the development of a generalized ferry clip based test system. The overall test system architecture is presented with high level description on the major components that comprise the ferry clip based test system.

3.1 Design Goals and Objectives

From past experience, reliability and flexibility of a test system are found to be of utmost importance [LINN85, LINN86, STOL89, VEL89]. Using these criteria as the basis for the development of a generalized conformance test system, the following objectives are identified:

- *Integrated Test Environment*
  The main objective is to achieve an integrated and automated test system that supports the overall process of derivation of executable test suites, test execution

and result analysis.

- *Compliance with OSI Testing Methodology*
  The OSI Conformance Testing Methodology as defined in [ISO-1] should be applied as closely as possible with the objective of achieving a uniform approach to all aspects of OSI protocol testing.

- *Protocol Independence*
  A test system should not be restricted to testing a specific protocol, but rather should allow the testing of any protocol layer and any IUT implementation in general. The test architecture should enforce a clear separation and encapsulation of the protocol dependent components. These components should be separated into modules which are easily customizable for the purpose of testing different protocol layers and different protocol implementations.

- *Test Configuration Independence*
  A test system should not be tuned to a single test configuration so as to realize a particular test method. Instead, it is highly desirable in practice that a test system offers mechanisms for adapting to different test configurations depending on the availability of standardized test suites and the accessibility of IUT interfaces. For example, if both interfaces are accessible, then the realization of local test method can be used, otherwise the distributed or remote test methods can be employed.

- *Communication System and System Under Test (SUT) Independence*
  Due to the heterogeneous nature of open testing environment, ideally a test system should be independent of the system which embeds the protocol implementation to be tested and should also allow easy interfacing with various underlying communication systems. This independence would help enhance the portability of the software components to be included in the SUT and relax the restrictive assumptions about the local operating system.

3.2 Test Architecture Overview

To meet the design goals outlined above, the overall test architecture enforces a clear separation and encapsulation of protocol dependency, test suite dependency and communication system dependency.

Figure 2.3 shows the proposed ferry clip based test architecture which is configured for the realization of the distributed test method. The following describes the major components of the ferry clip based test architecture, and highlights how the design goals described above can be achieved. Further details on certain components can be obtained from [CHANS89, ZENG89].

3.2.1 The Ferry Control Protocol (FCP)

The protocol that is used for the interactions between the active and passive ferry clips is known as the Ferry Control Protocol (FCP). The FCP is responsible for coordinating data transfer between the active and passive ferry clips using the reliable data transfer services provided by the Ferry Transfer Medium Protocol (FTMP). The Ferry Control Protocol is implemented within each ferry clip, and it can be defined in terms of the ferry clip services and the ferry control protocol data units as described below.

Ferry Clip Services

There are three types of ferry clip services [ZENG89], namely, *ferry data service*, *ferry management service*, and *ferry transfer service*. The abstract service primitives (ASPs) of these ferry clip services are shown in Table A.1 in Appendix A. Figure 3.1 illustrates the usage of the ferry clip services in the ferry clip based test architecture.

1. *Ferry Data Service:* The Ferry Data Service is used by the Passive and Active Ferry Service Interface Adapters (SIAs) within each ferry clip to send and receive test data. Any units of test data sent by one of the ferry service interface adapters to its associated ferry finite state machine will be delivered in the same format to the

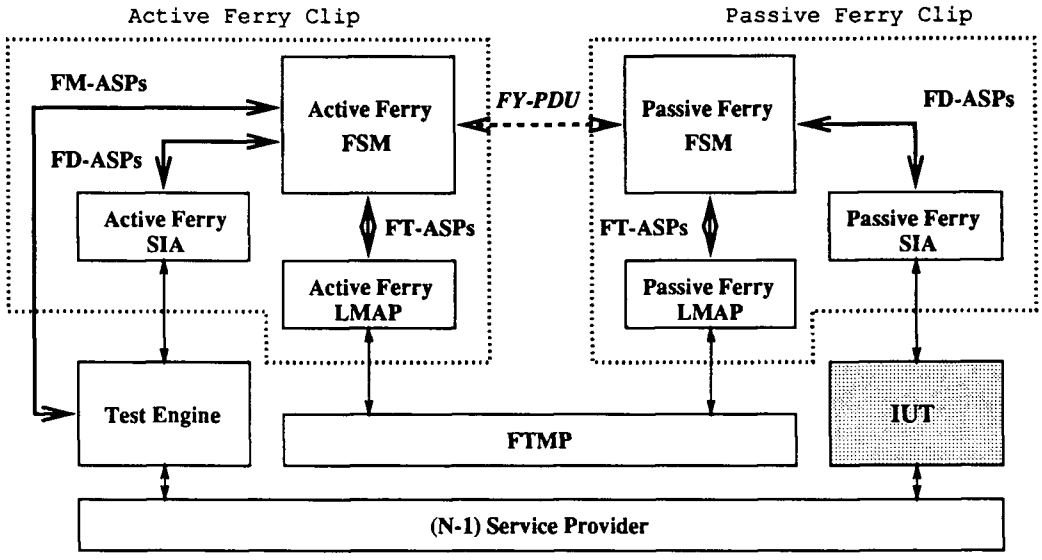


Figure 3.1: The usage of ferry clip services

other ferry service interface adapter.

2. *Ferry Management Service:* The Ferry Management Service is designed to provide the test system with the ability to control the connection between the active and passive ferry clips (*e.g.*, set up or abort ferry channel connection). This service is provided by the active ferry clip.

3. *Ferry Transfer Service:* The Ferry Transfer Service is used by the ferry finite state machines within each ferry clip for the exchange of ferry control protocol data units with its peer.

Ferry Control Protocol Data Units (FY-PDUs)

There are two types of ferry control protocol data units: FY-DATA PDU and FY-CNTL PDU. The format of these two PDUs are shown in Figure A.1 and A.2 respectively in Appendix A. Both of these PDUs are transferred between the active and passive ferry clips using the ferry data transfer service (*i.e.*, FT-DATA ASPs).

3.2.2 Passive Ferry Clip (PFC)

The main goal in designing a Passive Ferry Clip (PFC) is to keep it small, compact and portable so that it can reside in a SUT with limited memory, and usable in a variety of environments. The PFC is structured into the following modules to facilitate the replacement of both the IUT and the Ferry Transfer Medium Protocol (FTMP) (see Figure 2.3).

1. **Passive Ferry Finite State Machine (FSM)**
   The main function of this module is to implement the Passive Ferry's protocol state machine. It contains all the functions of the Passive Ferry that are independent of the IUT and the FTMP being used. It interacts with FTMP via a set of Ferry Transfer Service Primitives (FT-ASPs) and with the Service Interface Adapter via the Ferry Data Service Primitives (FD-ASPs) as described in [ZENG89].

2. **Passive Ferry Lower Mapping Module (LMAP)**
   This module maps the Passive Ferry transfer service primitives (FT-ASPs) onto the ASPs that are specific to the FTMP being used, *e.g.*, the ISO Transport or X.25 Network services. The functions contained in this module are specific to a particular FTMP, but independent of the IUT. Hence, this is the only module that needs to be modified if a different FTMP is used.

3. **Passive Ferry Service Interface Adapter (SIA)**
   The Passive Ferry Service Interface Adapter (SIA) interacts with the IUT through its upper and/or lower SAPs. It converts test input received in a ferry PDUs into a format that is specific to the IUT being tested, and converts observed output into a format that the tester understands. In order to enhance protocol independence and software portability of the passive clip, a uniform representation of the IUT's ASPs is achieved through the use of ASN.1 [ISO-8824, ISO-8825].

   The passive ferry SIA is further decomposed into two submodules: the E-node encoder/decoder and ASP converter modules. The E-node encoder transforms an E-node tree representation of ASN.1 specification into external stream-oriented

data for transmission and the decoder does the reverse. This is achieved through the ASN.1 Support Module which is described in more detail in Section 3.2.7. The ASP converter converts an E-node tree into the IUT's ASPs and vice versa. Since different implementations of the same protocol might represent the same ASP differently, only the ASP converter needs to be replaced if a different IUT is to be tested.

3.2.3 Active Ferry Clip (AFC)

The functions performed by the Active Ferry Clip (AFC), to a large extent, are very similar to those of the PFC. The AFC is also structured into three modules so as to minimize the changes necessary when a different FTMP is used.

1. **Active Ferry Finite State Machine (FSM)**
   The main function of this module is to implement the Active Ferry's protocol state machine. It contains all the functions that are independent of the IUT and the FTMP being used. It interacts with the Test Manager, FTMP and Active Ferry Service Interface Adapter by means of ferry ASPs, *i.e.,* FM-ASPs, FT-ASPs, and FD-ASPs respectively [ZENG89].

2. **Active Ferry Lower Mapping Module (LMAP)**
   This module maps the Active Ferry ASPs (FT-ASPs) onto the ASPs that are specific to the FTMP being used. By localizing the code specific to the FTMP in this module, it is possible to set up a library of Lower Mapping Modules corresponding to different FTMPs. Thus, the effort of configuring the Active Ferry to use a particular FTMP supported by the SUT is just a simple task of selecting an appropriate Lower Mapping Module from the library.

3. **Active Ferry Service Interface Adapter (SIA)**
   The active ferry SIA provides interface to the Test Manager and Active Ferry Finite State Machine. Because it is IUT dependent, this module is further decomposed into two submodules to facilitate its replacement for testing a different IUT: the

E-nodes encoder/decoder and ASP converter. The functions of these modules are similar to those described in the Passive Ferry Service Interface Adapter. Hence, only the ASP converter needs to be replaced if a different IUT is to be tested.

3.2.4 Test Suite Processor

The Test Suite Processor (TPS) contains all the logic necessary to process a test case. It is designed to communicate with a peer IUT under the direction of a test case, generate valid and invalid PDUs, verify test data received, and provide indication of pass/fail/inconclusive for a particular test case. Thus, the design of TPS is IUT independent as it simulates the protocol entity behavior by interpreting test cases. The TPS is different from a reference implementation test engine in that the TPS is capable of simulating both valid and invalid protocol behavior, and injecting semantically invalid but correctly coded PDUs. Thus, an IUT can be tested for its ability to recover from errors.

3.2.5 Test Manager

The Test Manager (TM) is the root of a hierarchy of modules that compose the ferry clip based test system (FCTS). It is responsible for synchronizing the actions of all its children modules. These actions include ferry management services, start or abort a test execution, and interface between the user and the test system. The TM communicates with the Active Ferry FSM via a set of FM-ASPs (see Table A.1 in Appendix A).

3.2.6 Service Provider Interface Module

The role of this module is to convert the service primitives from the Test Suite Processor into the ASPs specific to the underlying service provider. Similarly, it converts ASPs/PDUs received from the service provider before forwarding them to the Test Suite Processor. By localizing the service provider dependent code in this module, it is possible to set up a library of service provider interface modules corresponding to different service

providers. Thus, the task of customizing this module for interfacing with a particular service provider is reduced to selecting an appropriate interface module from the library.

3.2.7 ASN.1 Support Module

Protocol independence and software portability is achieved by enforcing a uniform representation of all relevant data structures (*i.e.,* IUT's ASPs and PDUs, ferry control PDUs) through the use of the *Abstract Syntax Notation One* (ASN.1) [ISO-8824]. We adopt a data structure called E-node to represent the abstract syntax. The E-node, originally developed for the EAN mail system [NEU86], is a tree-like data structure for representing ASN.1 data (*e.g.,* ASPs/PDUs) based on the *Basic Encoding Rules* (BER) [ISO-8825]. The use of E-nodes allows several data manipulation functions to remain protocol independent:

- *Coding and Decoding*
  Since the E-node is used to represent any BER based data, two protocol independent routines, namely *encoder* and *decoder* are required. The encoder transforms an internal tree representation (E-node) into an external stream-oriented (BER) data for transmission; whereas the decoder transforms a received BER data into an E-node tree.

- *Verification of ASPs and PDUs*
  This function is specifically developed to support the automatic test execution subsystem. By representing the constraint specification of ASPs and PDUs in an E-node format, it allows the test execution subsystem to inspect the observed ASPs/PDUs to determine whether they satisfy the constraints as specified in the test case.

3.2.8 Activity Log Module

Since testing does not always produce a simple pass or fail verdict, in order to properly evaluate the conformance of an IUT, it is important for a test system to produce a

detailed log which helps to determine whether: (1) the IUT fails, or (2) the protocol specifications contain ambiguities which lead to different interpretations, or (3) other errors which are caused by the underlying communication services or test system. The role of this module is to produce two test log files with an easy-to-read structure that can be used for further results analysis after the test execution. The first log file, called *conformance log*, contains the basic information such as the test cases selected, time of initiations, test events executed and test completion results. The second log file, called *event traced log*, contains a detailed trace of all events observed by the test system. The parameter values of ASPs/PDUs exchanged between the test system and the IUT, indications of protocol violations, etc. are logged in this file.

3.3 The Use of ASN.1 Representation

3.3.1 Representation of ASPs Using ASN.1

As pointed out by Bochmann in [BOCH88], the adaptation of abstract service primitives (ASPs) to the real service interface of an IUT is a major issue for both the portability and protocol independence of the testing software. This is important as the software should be usable for testing different protocol implementations which may operate within different SUTs. The adaptation problem arises due to the fact that the format of ASPs is not standardized, and as such they are realized in different ways within different protocol implementations to be tested, and within different test systems. This problem is also faced in the design of the Ferry Clip based Test System.

In the case of the ferry clip based test architecture, test input to the IUT (*i.e.*, information representing the service primitives to be input to the IUT) and the observed output of the IUT (*i.e.*, information representing the the service primitives initiated by the IUT) must be transferred between the active and passive ferry clips through the ferry channel. Therefore, the representation of the service primitives in the test system, at the interface between the test system and the IUT, and at the interface between the test system and the underlying communication service provider are of prime importance to

the design of the test system. To enhance protocol independence of the test system as well as the portability of the testing software (*i.e.,* passive ferry clip) to be included in the SUT, it is imperative to have a uniform and consistent representation of information exchanged between the test system and the SUT. The use of Abstract Syntax Notation One (ASN.1) [ISO-8824] and its associated Basic Encoding Rules (BER) [ISO-8825] is proposed for this purpose.

The use of ASN.1 for representing the information exchanged between the test system and the SUT offers many advantages, the notable ones are as follows:

- Different machine have different representation for basic data types, and ASN.1 provides a mechanism to formally describe data types and values without specifying any particular representation for the data being described. This ensures data compatibility between the test system and SUT.

- Since ASN.1 has been designed to be independent of any programming language or operating system, its use as a representation for the ASPs is particularly useful in those cases where the interface representation must be independent of the program structure of the interfacing software/hardware modules [BOCH88].

- ASN.1 allows complex data structures to be defined using simple data types (e.g., *Integer, Boolean, Bitstring, Octetstring, etc*). Furthermore, it has been used extensively in existing OSI application protocols as well as many new applications. The use of ASN.1 for the representation of ASPs is particularly well suited for the testing of OSI Application layer protocols, since the data exchanged at the internal interfaces of protocol implementation between different protocol sublayers (*e.g.,* interactions between MTAEs and UAEs within X.400) are encoded in ASN.1 format.

3.3.2 E-Node: A Data Structure for Representing ASN.1

All relevant data structures, *i.e.,* ASPs, PDUs, ferry PDUs, etc. are represented in a unified form internally using an adaptation of *E-nodes*, which was originally developed

for the EAN [NEU86] mail system at the University of British Columbia. The E-node is a tree-like data structure that is capable of representing any abstract syntax that are defined in ASN.1. Two forms of extended E-node data structures have been derived to support the use of ASN.1 within a protocol testing environment:

Template E-nodes

The Template E-nodes are used to represent the ASN.1 definitions of ASPs and PDUs as specified in the protocol standards. The Template E-node largely preserves the structure of the ASN.1 types, and as such, each node of a Template E-node tree corresponds to an ASN.1 type. Structured (or Constructor) types such as *SETs, SEQUENCEs, CHOICEs* are represented as trees, and the node which corresponds to a structured type is called the *parent* node; the components (or children) of the type are represented as leaf nodes, which are linked together in a sibling list, thereby preserving the order of the components of the ASN.1 definition. Each node in a Template E-node tree contains the following information:

- *E-node id:* ASN.1 tag, *i.e.,* class, primitive/constructor, and tag number
- *Type:* ASN.1 defined type, *e.g.,* BOOLEAN, INTEGER, etc.
- *Field name:* the field name specified in ASN.1 definition, *e.g.,* CountryName
- *Attribute:* attribute flag used to indicate OPTIONAL, CHOICE, SET OF, etc.
- *Content pointer:* points to an E-node subtree representing the components of a construct if type is SET, SEQUENCE, etc.
- *Next pointer:* points to a list of alternatives of a CHOICE, a list members of a SET, or elements of a SEQUENCE

Value E-nodes

The Value E-nodes are used to represent data instances of an ASN.1 definition. In order to incorporate the constraints of ASPs/PDUs specified in the constraints part of a TTCN

test suite, the Value E-node structure has been extended to allow for the specification of value constraints as well as the actual data values. The following are the types of constraints which are most useful towards the development of an automatic test system as they facilitate the process of verifying the data instances of ASPs/PDUs received during test execution:

OMIT ("–")	specifies the parameter must be absent
ANY ("?")	specifies the parameter can be any single value
ANY_OR_OMIT ("*")	specifies the parameter can be any value or absent
Bitstring	bitmasks for setting or resetting bits can be specified
Integer	upper and lower bounds can be specified
Octetstring	pattern or wildcards may be specified
SEQUENCE	special constraints (*e.g.*, "-", "*") can be specified
SET	special constraints (*e.g.*, "-", "*") can be specified

Similar to Template E-node, each structured (or Constructor) ASN.1 type corresponds to a parent node in a Value E-node tree, and its children nodes are linked together in a sibling list. However, these children (or leaf) nodes store the actual data values of types defined in the corresponding Template E-node tree. The Value E-ndoes are stored on disk and are identified by their unique tree names. To enhance reusability and reduce disk storage, a tree attachment mechanism is provided for a Value E-node; *i.e.*, a Value E-node tree can reference one or more Value E-node trees. An entire Value E-node tree can thus be built by recursively expanding the attached E-nodes. This allows a Value E-node tree to contain an arbitrary number of elements (or nodes) on the condition that no loop may exist. Each node in an Value E-node tree contains the following information:

- *E-node id:* ASN.1 tag, *i.e.*, class. primitive/constructor, and tag number
- *Length:* length of contents
- *Constraint type:* flag to indicate the type of constraint, *e.g.*, ANY (element can be any value), Integer constraint (upper and lower bound can be specified), etc.
- *Bitstring constraint:* bitmask and value bitstring can be specified
- *Integer constraint:* upper and lower bounds can be specified for integer value

- *Attached tree name:* reference an external Value E-node tree, the attached tree can be dynamically expanded during run time
- *Content pointer:* points to an E-node subtree representing the components of a construct if type is SET, SEQUENCE, SET OF, etc.
- *Next pointer:* points to a list of members of a SET, elements of a SEQUENCE

Since the Value E-node can be used to represent data instances of any ASN.1 definition in a unified form as trees, it allows several data operations to be protocol independent. This facilitates the development of a set of "generic" ASN.1 support tools as described in Section 4.5. The separation between Template and Value E-nodes provides efficiency both in space and time required for processing, since only a single copy of Template E-node tree is needed to remain in memory, allowing it to be referenced by an arbitrary number of Value E-node trees.

3.4 Executable Test Suites Design Overview

Experience has shown that one of the key decisions in designing a test system is whether to develop a complex test language versus a complex test engine [LINN85, ESWA90, MAT88]. Since there are technical merits for either approach, the underlying question is where to represent the residual complexity of a communication protocol that remains in the test system. If the complexity of a protocol is represented in a complex test language, a protocol-independent test engine can be built. The test engine simply interprets and executes under the direction of complex test cases. The complex test engine approach (*e.g.,* using reference implementation) leads to simple test cases to be represented in a simple, but protocol-dependent test language and test engine.

We have adopted the complex test language approach in the design of our test architecture. There are several reasons for choosing this approach. First, the detailed knowledge of the protocol implementation is encapsulated in the complex test cases, and is hidden from the test architecture designer. This enhances both the protocol-independency and reusability of the test system. In addition, maintenance on the test

system may be totally eliminated if there are any subsequent updates on the protocol implementation due to revisions in the protocol specifications. Second, unlike the use of reference implementation method, the complex test language approach provides greater degree of flexibility. For example, a test system may need to generate alternative correct behavior which deviate from a conforming reference implementation, inject semantically invalid but correctly coded PDUs, etc.

The complex test language approach starts from abstract test suites which are specified in a notation called Tree and Tabular Combined Notation (TTCN) [ISO-1]. To test a protocol implementation based on abstract test suites that are specified in TTCN, it is essential to define a comprehensive executable test language (or notation) that is capable of capturing both the dynamic behavior and constraints parts of a TTCN abstract test suite. The following subsections describe the major components of an *Executable Test Suite* (ETS) and their derivations based on a standard TTCN abstract test suite.

3.4.1 Executable Test Suite Behavior

The executable test suite behavior (or ETS behavior) is the realization of the dynamic behavior of a TTCN abstract test suite. The behavior is represented internally in a unified form as trees. Each node of an executable test tree represents an encoding of a single test action (*e.g.*, sending or receiving test event in TTCN), and each test step in a TTCN test case is represented by an executable test tree which is uniquely identified by its tree name. Thus, a single TTCN test case may be represented by one or more executable test trees with each tree referencing the others by storing their tree names. An entire executable test tree can therefore be built by recursively expanding the attached trees.

There are many advantages of using tree representation for the dynamic behavior of a TTCN test case. First, since TTCN itself is expressed in a tree notation, representation of TTCN in tree form naturally preserves the basic structure and the dynamic behavior of a TTCN test case; *i.e.*, execution is to progress from left to right (sequence) and from top to bottom (alternatives). Second, trees can be easily implemented in any

programming language by using dynamic memory allocation and pointers. Third, the use of tree representation provides efficiency both in time and space required for processing a test case, since each test case can be dynamically loaded into the memory during test execution and removed from the memory at the end the test execution. The algorithm required for processing a test case is also simplified since each node contains information about a test action, execution of a test case is therefore similar to traversing a tree from node to node. Fourth, tree representation of test cases fits well into the architecture of an event driven test system (*e.g.*, test system developed within OSI-PTE) since the transition from one node to another provides an excellent break point for returning control to the test system [LEE89].

3.4.2 Executable Test Suite Constraints

The executable test suite constraints (or ETS constraints) are the realizations of the constraints part of a TTCN abstract test suite. The constraints of test ASPs and PDUs (both send and receive) are represented using a Value E-node data structure (see Section 3.3.2). These ASPs/PDUs constraints are stored as separate files in a ETS Constraints library, and can be dynamically linked to an executable test tree during test execution. Since the dynamic behavior of a TTCN test case is represented separately and independently from its associated test ASPs/PDUs constraints, changes on the dynamic behavior of the TTCN test case will not result in any modification to the ETS constraints; similarly, modification of the ETS behavior will not be required should changes be made to the the test ASPs/PDUs constraints specifications.

3.4.3 Derivation of the Executable Test Suite

The derivation process from the "abstract" test cases to the "executable" test events can be divided into the following three phases:

1. *Pre-processing phase:* An abstract test case specified in TTCN must undergo a series of transformation in order to render them executable. First, the ASN.1

constraints specification of test ASPs and PDUs defined in the constraints part of a TTCN test suite are transformed into an internal E-node tree representation and stored in the "Executable Test Suites (ETS) constraints library". Second, the dynamic behavior of the TTCN test suites are transformed into an internal executable tree notations which are stored in the "ETS behavior library". Each node of the executable test tree represents a single test event which is defined in the dynamic behavior of a TTCN test case. The above transformation process can be automated with the aids of a compiler/parser to avoid human errors and to ease maintenance.

2. *Actualization phase:* When a test session is initiated, the Test Suite Processor dynamically loads the desired test case from the ETS behavior library into a pre-allocated block of memory. In this phase, the referenced ASPs/PDUs constraints and test suite parameters are dynamically loaded from the ETS constraints library and linked to the ETS behavior tree. The Test Suite Processor then checks the executable test event for syntax and static semantic errors after actualizing the test events with the data values pre-encoded in the E-node trees.

3. *Execution phase:* The execution of a single test event is performed by various test system components under the direction of the Test Suite Processor. The Test Suite Processor provides the information on the selected service primitives and the parameter values, this enables the test system to transform a single test event into test action, for example, sending an ASP/PDU to the IUT.

Chapter 4

Implementation

This chapter describe the implementation of the proposed ferry clip based test system based on the design requirements outlined in Chapter 3. The chapter begins with an introduction to the implementation environment, *i.e.*, OSI-PTE, followed by a detailed description of the individual components of the ferry clip based test system. The functions of the ASN.1 support tools and the derivation of executable test suites are described and illustrated using some examples.

4.1 The OSI-PTE Environment

We have applied the principles described in Chapter 3 to our implementation of the proposed ferry clip based test system as shown in Figure 2.3. The test system was developed within the *Open Systems Interconnections - Protocol Testing Environment* (OSI-PTE) [CHAN89, SMITH89] which runs on UNIX [1] 4.3BSD (SUN/OS 4.0) [2]. To enhance system independence, the entire test environment was implemented using the C programming language. The OSI-PTE is a realization of the OSI Reference Model (OSI-RM) [CCITT-4] within a single operating system process for protocol testing purposes. Besides providing an operating environment which is close to the OSI-RM, it also allows

[1] UNIX is a trademark of AT&T Bell Laboratories

[2] Sun is a trademark of Sun Microsystems, Inc.

for the incorporation of a test manager into the test system to support all the test methods defined by ISO as well as passive monitoring, logging and analysis capabilities.

In essence, the OSI-PTE is an event-driven system as entities communicate with each other by posting events. An *entity* is a basic execution unit in the OSI-PTE. Basically, there are two types of entities: *protocol entities* and *test entities.* Protocol entities are analogous to OSI-RM entities, they are implementations of supported protocols in execution. A protocol entity provides services to protocol entities at the layer above, and utilizes services provided by protocol entities at the layer below. The only means of communication between protocol entities at adjacent layers is by ASPs through SAPs. Each protocol entity defined in the OSI-PTE is uniquely identified by an *entity identifier* (NID). Test entities are user programs which utilizes the protocol entities to control and observe the IUT. They may observe, alter, or intercept events that are posted between protocol entities, or generate events and posts them to a protocol entity.

The design of OSI-PTE enforces a clear separation between the data areas and code areas of an protocol entity. All global variables used by the protocol entities are placed in control blocks, and all subroutines using these global variables take a pointer to a control block as one of their parameters. This separation between data and code allow multiple instances of a protocol entity to be created.

When an entity establishes a connection with its adjacent protocol entities, a separate *Connection Control Block* (CCB) is created and maintained by the entity. This CCB is used to contain information that is specific to the connection, it is associated with the *connection identifier* (CID) that is assigned to the connection.

Communication between protocol entities achieved by means of an event-posting scheme whereby one protocol entity posts an event to another for processing. To meet the requirements of protocol testing, seven types of events are defined in the OSI-PTE: ASP events, PDU events, timer events, state change events and protocol error events. Each event type is assigned an *event type identifier* (SID), and is accommodated by defining an event interface for a protocol entity. Figure 4.1 shows the seven event interfaces for an OSI-PTE protocol entity. Each event posted to an entity is represented by an *event*

identifier (EID) and an associated *Event Parameter Area* (EPA). An EPA is a control block which contains parameters, such as those specified in a service specification document, to be passed between two entities during an interaction. The receiving entity uses the event identifier to determine the structure of the EPA and the action necessary to process the event.

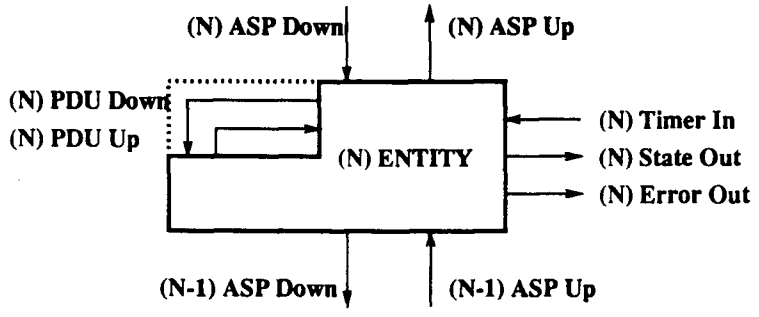


Figure 4.1: Event Interfaces for an OSI-PTE Protocol Entity

Thus, when an entity posts an event, it uses the *PostEvent* service, which requires the following five parameters:

PostEvent (nid, cid, sid, eid, epa)
where nid : identifier of the entity which the event is posted to,
cid : identifier of the connection,
sid : event interface type, *e.g.*, ASP-UP,
eid : event identifier, *e.g.*, N-CONNreq
epa : event parameter area pointer.

Since the OSI-PTE is an event-driven system, processing within the OSI-PTE is driven by the external events, such as *frame arrival*, time out and *external service request.* Thus, a *dispatcher* is introduced to translate external events into internal events. When an internal event is dispatched by the dispatcher, the entity invoked by the dispatcher may in turn invoke other entities, via events posting, to carry on the processing. The dispatcher will not regain control until all the internal events, that are caused to be generated by the initial external event, have been completely processed by the entities involved.

4.2 Overview of the Implementation

Our goal is to develop the proposed multi-party ferry clip based test system (see Figure 2.3) that would allow us to conformance test a protocol (*e.g.*, MHS) in the OSI-PTE environment. The entire testing environment, which includes the ferry clip based test system and a set of protocol implementations, has been implemented in the C programming language and runs on UNIX 4.3BSD (SUN/OS 4.0).

Communication between the SUT and test system is achieved through the use of UNIX stream sockets in the Internet domain (TCP/IP) [SECH86]. This type of socket domain has been chosen as the underlying communication service because it provides a reliable end-to-end delivery of test data. The Message Handling System (MHS), recently developed by the UBC-IDACOM project group, is used as the IUT which will be tested for conformity using the ferry clip based test system. To accomplish this task, a stack of OSI protocol implementations must be used on both the SUT and the test system to provide the required underlying OSI services. The stack comprises the following OSI protocol implementations: LAPB (Data Link layer), X.25 PLP (Network layer), Transport layer, and Session layer. Each protocol contained in the stack has been individually tested to ensure that it has met the relevant ISO or CCITT protocol specifications.

One of the challenges faced in this implementation is to develop a highly flexible and generalized test tool that meet the design requirements as outlined in Section 3.1. The following sections describe the implementation of the major components which comprise the multi-party ferry clip based test system.

4.3 The Active Ferry Clip

The *Active Ferry Clip* (AFC) is structured into three main modules, namely the Active Ferry FSM, Active Ferry LMAP, and the Active Ferry SIA as shown in Figure 3.1. These modules interact with each other using the ferry clip services – Ferry Data Service, Ferry Management Service, and Ferry Transfer Service – as shown in Table A.1 in Appendix A (see [ZENG89] for detailed description of these services). To facilitate the implementation

of the AFC within the OSI-PTE environment, the Active Ferry FSM, Active Ferry LMAP, and the Active Ferry SIA are configured as a separate OSI-PTE entities as shown in Figure 4.2. The entities communicate with one another using the events posting scheme provided by the OSI-PTE.

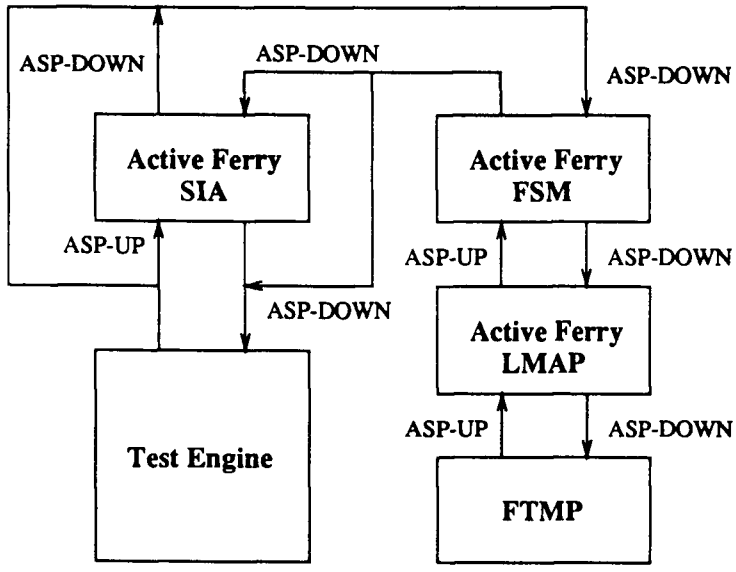


Figure 4.2: Configuration of the Active Ferry Clip in the OSI-PTE environment

4.3.1 The Active Ferry FSM

The Active Ferry FMS is configured as a single OSI-PTE protocol entity. The Ferry Control Protocol (FCP) is implemented within this entity as a simple finite state machine with three states: *idle, connecting,* and *connected.* The Active Ferry FSM entity will perform an appropriate action depending on its current state and the ferry event received based on the state transition rules defined in Table A.3 as shown in Appendix A. It communicates with its adjacent entities (*i.e.*, Active Ferry SIA, Test Engine) using the OSI-PTE events posting scheme.

Figure 4.3 shows the event interfaces for the Active Ferry FSM entity. These events are classified based on the services into the following types:

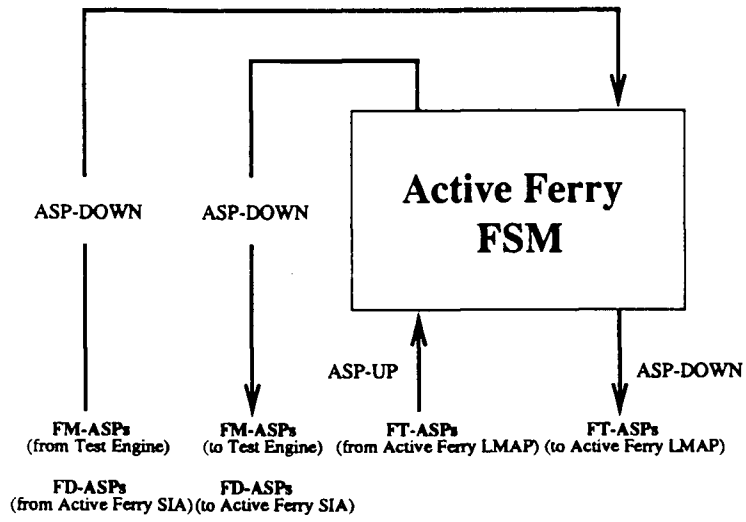


Figure 4.3: Event Interfaces for the Active Ferry FSM Entity

1. *Ferry Data Service Event*

   The Active Ferry SIA entity uses the Ferry Data Service provided by the Active Ferry FSM to send and receive test data. This is achieved through the use of *PostEvent* service (see Section 4.1) provided by the OSI-PTE system. For example, to send test data to the Active Ferry FSM, the Active Ferry SIA entity posts a FD-DATA request event to the ASP-DOWN event interface of the Active Ferry FSM entity.

2. *Ferry Management Service Event*

   The Test Engine entity uses the FM-ASPs to set up, maintain, and control the connection between the active and passive ferry clips. This is done by posting an appropriate FM-ASPs (*e.g.*, FM-CONNECT request) event to the ASP-DOWN event interface of the Active Ferry FSM entity, which will then take an appropriate action based on the transition rules defined in the the active ferry clip state transition table (see Table A.3 in Appendix A).

3. *Ferry Transfer Service Event*

   The Ferry Transfer Service is designed to provide the means by which the active and passive ferry clips exchange the ferry PDUs (*i.e.*, control or data PDUs). When a

3. *Ferry Transfer Service Event*
   The Ferry Transfer Service is designed to provide the means by which the active and passive ferry clips exchange the ferry PDUs (*i.e.*, control or data PDUs). When a ferry PDUs is to be sent to the passive ferry clip, the Active Ferry FSM entity posts an appropriate FT-ASPs (*e.g.*, FT-CONNECT request) event to the ASP-DOWN event interface of the Active Ferry LMAP entity, which will then map the FT-ASPs onto a specific FTMP transfer service (*e.g.*, FT-CONNreq will be mapped to N-CONNreq if X.25 is used as the FTMP).

4.3.2 The Active Ferry LMAP and FTMP

The Active Ferry LMAP module is configured as a separate OSI-PTE entity with event interfaces as shown in Figure 4.4. It acts as an interface adapter between the Active Ferry FSM and the FTMP, its function is to map the ferry transfer service primitives (FT-ASPs) to the ASPs that are specific to the FTMP being used. In this implementation, X.25 PLP has been used as the FTMP for several reasons. First, it provides a reliable transfer of ferry PDUs between the active and passive ferry clips; second, it is the most popular and widely available data transfer service which supports channel multiplexing facility. The mapping of FT-ASPs to X.25 Network layer services is shown in Table A.2 in Appendix A.

4.3.3 The Active Ferry SIA

The Active Ferry SIA is configured as a single OSI-PTE protocol entity which interacts with its two adjacent entities, the Active Ferry FSM and the Test Engine, using the post event service provided by the OSI-PTE system. Its main function is to convert data sent by the Test Engine in the E-node tree format into a linear BER octet stream, which will then be packed into FY-DATA PDUs (see Appendix A) and forwarded to the Active Ferry FSM for transmission to the passive ferry clip. On the reverse side, it converts data received from the Active Ferry FSM in ASN.1 byte stream format into an E-node tree before forwarding it to the Test Engine.

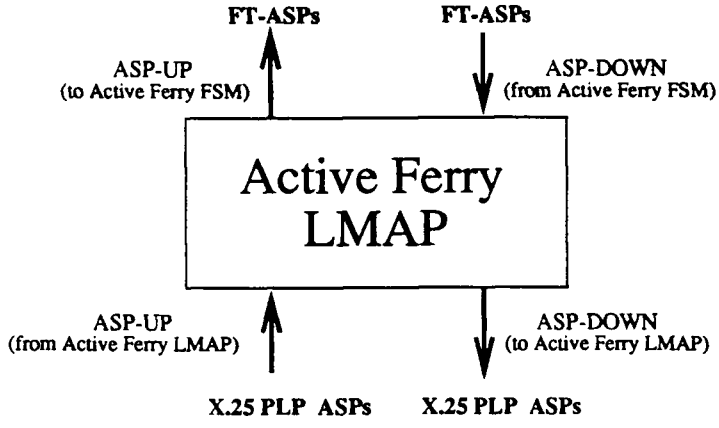


Figure 4.4: Event Interfaces for the Active Ferry LMAP Entity

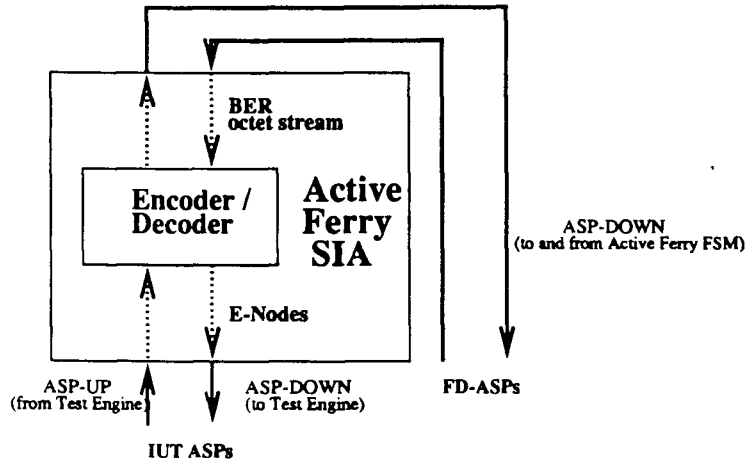


Figure 4.5: Structure of the Active Ferry Service Interface Adapter

The structure of the Active Ferry SIA is shown in Figure 4.5. To achieve protocol independence and software portability requirements, a uniform representation of information exchange between the active and passive ferry clips is enforced through the use of ASN.1. Two "generic" routines have been developed to convert data instances of any ASN.1 abstract specification between E-node tree and BER [ISO-8825] octet stream formats. The *encoder* encodes an E-node tree, which is a tree structure representation of ASN.1 [ISO-8824] data values in transfer syntax, into a linear BER octet stream; conversely, the *decoder* decodes BER octet stream into an E-node tree. The functions of the

encoder/decoder will be described in more detail later in this chapter.

For generality and reasons that are dictated by the OSI-PTE environment, the Active Ferry SIA makes two FD-DATAreq calls to the Active Ferry FSM to transfer an IUT ASPs event. The first FD-DATAreq contains an ASP event identifier, and a boolean indicator which denotes the presence or absence of the associated event parameter area (EPA). The second FD-DATAreq contains the associated ASP event parameters and/or the service data unit if either one is present, otherwise this call will not be made.

One of the requirements of multi-party testing is to test the capability of the IUT for handling several test connections in parallel, this is achieved by multiplexing multiple logical test connections onto a single ferry channel, thereby allowing multiple entities to communicate over the ferry channel between the SUT and test system. To handle this multiplexing capability, a minor change in the FY-DATA PDU's header format is required. An additional "connection-id" byte has been added to the header of FY-DATA PDU as shown in Figure A.1 in Appendix A.

4.4 The Passive Ferry Clip

The *Passive Ferry Clip* (PFC), mirrors the Active Ferry Clip, is structured into three main modules, namely the Passive Ferry FSM, Passive Ferry LMAP, and the Passive Ferry SIA as shown in Figure 3.1. These modules interact with each other using the ferry clip services (see Table A.1 in Appendix A). To facilitate the implementation of the PFC within the OSI-PTE environment, the Passive Ferry FSM, Passive Ferry LMAP, and the Passive Ferry SIA are configured as a separate OSI-PTE entities as shown in Figure 4.6. These entities communicate among with one another through the use of the event posting scheme provided by the OSI-PTE.

4.4.1 The Passive Ferry FSM

The Passive Ferry FSM is configured as a single OSI-PTE protocol entity with event interfaces as shown in Figure 4.7. Functionally, the Passive Ferry FSM is very similar to

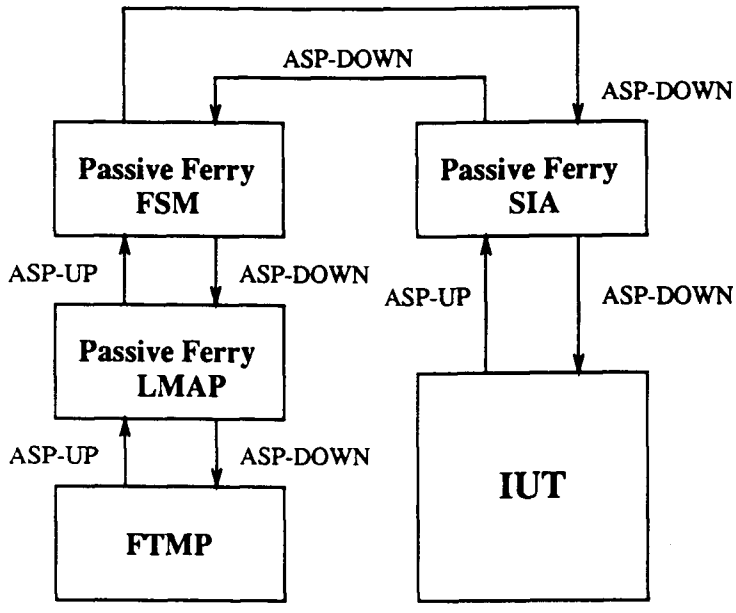


Figure 4.6: Configuration of the Passive Ferry Clip in the OSI-PTE environment

the Active Ferry FSM with the following exceptions:

- The Passive Ferry Clip's protocol state machine implemented within this entity has only two states: *idle*, and *connected*. The state transition rules defined for the PFC is shown in Table A.4 in Appendix A.
- It communicates with its adjacent entities using two of the ferry clip services: Ferry Data Service (FD-ASPs) and Ferry Transfer Service (FT-ASPs). No Ferry Management Service (FM-ASPs) is used within this entity, as it is not responsible for managing the ferry connection between the active and passive ferry clips – this is why it is termed "passive".

4.4.2 The Passive Ferry LMAP

The Passive Ferry LMAP is configured as a single OSI-PTE entity with event interfaces as shown in Figure 4.8. It has the same function as the Active Ferry LMAP, that is to

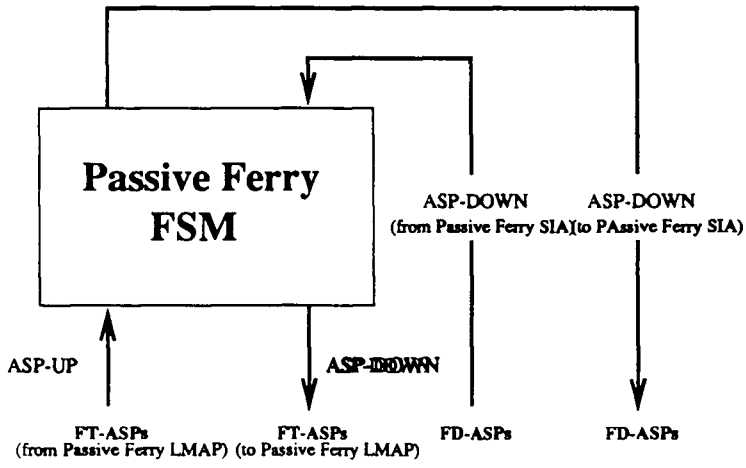


Figure 4.7: Event Interfaces for the Passive Ferry FSM Entity

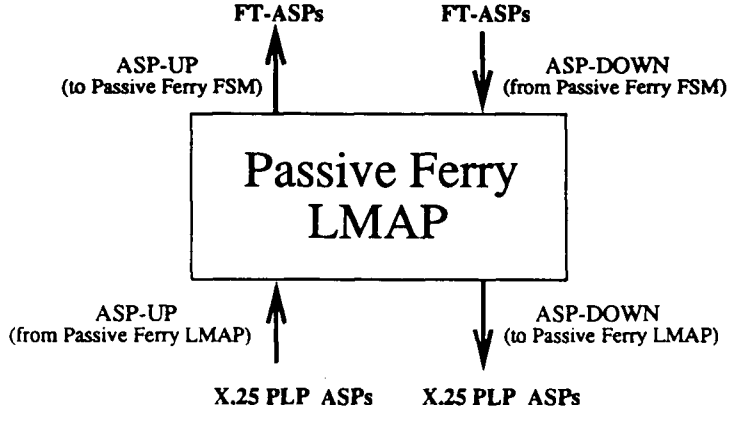


Figure 4.8: Event Interfaces for the Passive Ferry LMAP Entity

4.4.3 The Passive Ferry SIA

The Passive Ferry SIA is configured as a single OSI-PTE entity whose main function is to convert IUT service primitives between IUT-dependent and tester-dependent formats. To isolate protocol dependency, the Passive Ferry SIA is structured as shown in Figure 4.9. The Encoder/Decoder module contains two protocol independent routines – *encoder* and *decoder* – which convert data received in E-node tree format to BER octet stream and vice versa. The ASP Converter module, which is IUT dependent, converts data received in E-node tree to IUT service primitive format and vice versa. Hence, only the ASP

decoder – which convert data received in E-node tree format to BER octet stream and vice versa. The ASP Converter module, which is IUT dependent, converts data received in E-node tree to IUT service primitive format and vice versa. Hence, only the ASP Converter module needs to be replaced or modified when a different IUT is to be tested.

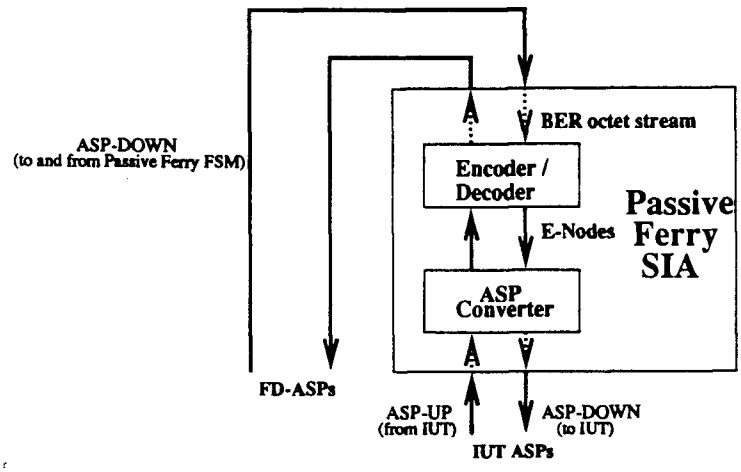


Figure 4.9: Structure of the Passive Ferry Service Interface Adapter

To support several test connections in parallel between the SUT and test system, some mechanisms must be provided for the Passive Ferry SIA to multiplex multiple logical test connections onto a single ferry channel. One simple solution is to add a "connection-id" byte to the header of a FY-DATA PDU as shown in Figure A.1 in Appendix A. The need for this additional connection-id field was first proposed in [PAR89]. This connection-id field specifies the connection which the test data is to be sent to or received from. This enables the Passive Ferry SIA to identify which connection of the IUT the test data, which is sent to the passive ferry clip via the ferry channel, is to be sent to. Similarly, data received from a particular connection of the IUT requires an identification of the test connection to be associated with the test data which is to be sent to the active ferry clip via the ferry channel.

4.5 ASN.1 Support Tools

A set of ASN.1 support tools have been developed to facilitate protocol testing within the OSI-PTE environment. These support tools are developed based on the following processing requirements:

- **Encoding and Decoding Routines**
  Since E-node is a linked tree data structure which is capable of representing any abstract syntax definition, this allows us to develop two "generic" routines – *encoder* and *decoder* – that provide the transformation between the internal BER value tree representation and the external linear BER octet stream. The encoder routine encodes an E-node tree into a linear BER octet stream which can be used for external transmission of ASPs/PDUs; whereas the decoder routine decodes a linear BER octet stream into an E-node tree representation which can be used for data instance verification. The decoder builds an E-node tree incrementally as it decodes a received PDU in BER octet stream. A complete E-node tree is created if no error is encountered. Hence, the decoding procedure also verifies the transfer syntax of the received PDU.

- **Automatic Verification of ASP/PDU Data Instances**
  In communication protocols, ASP/PDU may often take on a range of values. To manage this in protocol testing, we have extended the E-node data structure to allow for the specification of value constraints as well as the actual data values. The constraints specification of ASPs/PDUs is the realization of the constraints section of a TTCN abstract test suite, and as such, they are incorporated into the executable test scenarios. This enables the test system to perform automatic verification of a received ASP/PDU to determine whether it satisfies the constraints as specified in the E-node tree.

- **Service Primitives Conversion Routines**
  A set of routines have been developed to assist the transformation between the two different formats for representing the ASPs:

- *E-nodes to Protocol-dependent format:* There is a need to convert ASPs represented in E-node tree structure into the format required by protocol-specific service primitives. Such need arises at the interfaces between the test system and the underlying communication service provider, and between the test system and the IUT. These routines require an ASP's event parameter area (EPA) (see Section 4.1) offset table as a parameter. This table contains the offsets of each EPA field from the start of the EPA structure and its corresponding field type. The table entries are arranged in depth-first order; *i.e.,* the order of recursive traversal of an enode tree.
- *Protocol-dependent format to E-nodes:* An entire E-node tree representation of ASP is built incrementally by constructing each field in an EPA. Since ASPs are realized in different ways by different protocol implementations, a separate procedure is written for each specific ASPs being used. A node in the E-node tree can be constructed by calling the appropriate procedure which in turn invokes the ASN.1 support routines to convert the field value from an internal representation into a transfer syntax format based on the Basic Encoding Rules (BER) [ISO-8825].

As an illustration, the Value E-node data structure specified in the C programming language is shown in Appendix B. Figure C.1 in Appendix C shows an excerpt of a P1 PDU (*i.e.,* ORName) defined in ASN.1 and its corresponding data instance in E-node tree representation.

4.6 Derivation of Executable Test Suite

A set of utility routines have been developed to facilitate the translation of TTCN abstract test cases into internal executable test notations. These executable test notations are represented using a tree-like data structure. Appendix D shows the definitions of the executable test notations specified in the C programming language.

The translation of the dynamic behavior of a TTCN test case into an executable test

tree notation can best be illustrated using the example shown in Appendix E. Figure E.1 is a X.403 test case specified in TTCN, the corresponding executable test tree representation is shown in Figure E.2. It is noted that the executable test tree preserves the basic structure of the dynamic behavior of a TTCN test case. The test case is used to test whether the IUT is able to behave as relay and a recipient. The tester LT_1 sends a UMPDU (User Message Protocol Data Unit) for two recipients, one residing on the IUT and the other requires the IUT to relay the UMPDU to it. The test is passed if the IUT delivers the UMPDU to its UAE and makes a copy which is relayed to tester LT_2. The copy should have no "recipient-info" parameter concerning the IUT or that its responsibility flag is not set.

A specific Test Suite Processor has been developed to process the executable test tree. The Test Suite Processor implemented is capable of supporting the following TTCN test events and operations as defined in ISO standards [ISO-1]: (1) Sending and Receiving events, (2) Timeout event, (3) Otherwise event, (4) Tree Attachment operation, (5) Timer Start operation, (6) Timer Cancel operation, (7) Read Timer operation, and (8) Goto operation. The implementation of the Test Suite Processor contains about 160 lines of C code.

Chapter 5

The Ferry Clip Approach to Multi-party Conformance Testing

This chapter starts with a discussion of the general issues and requirements of a multi-party conformance testing. This is followed by an overview of the Message Handling System (MHS), together with its the testing requirements. The chapter proceeds to illustrate how the proposed ferry clip based test architecture can be configured to meet these requirements and achieve the purpose of multi-party conformance testing within the scope of OSI. An example using the test system to test an implementation of MHS is presented. The chapter ends with a discussion of the results and experiences obtained from the testing performed.

5.1 Issues and Requirements of Multi-party Conformance Testing

There are several issues which must be addressed during the design of a multi-party test system. The first issue concerns the structure of a multi-party configuration. The ISO has identified three multi-party configurations [ISO-2] as determined by the manner in which the IUT or IUTs are associated with its multiple peers, specifically:

- a single IUT in SUT A communicating with multiple entities in a single system B;

- a single IUT in SUT A communicating with multiple entities in different systems (one or more entities per system);
- multiple IUTs in SUT A communicating with multiple entities in a single system B.

To simplify our discussion, Figure 5.1 depicts an example of a multi-party test architecture modelled after the ISO distributed test method (see [ISO-2] for details). In this model, a set of lower testers execute in parallel (parallel lower testers or PLTs) and behave as a set of peer entities to the IUT. The activities of the PLTs are coordinated by the master lower tester (MLT). The role of the upper tester is to coordinate with its peers to achieve the test purposes.

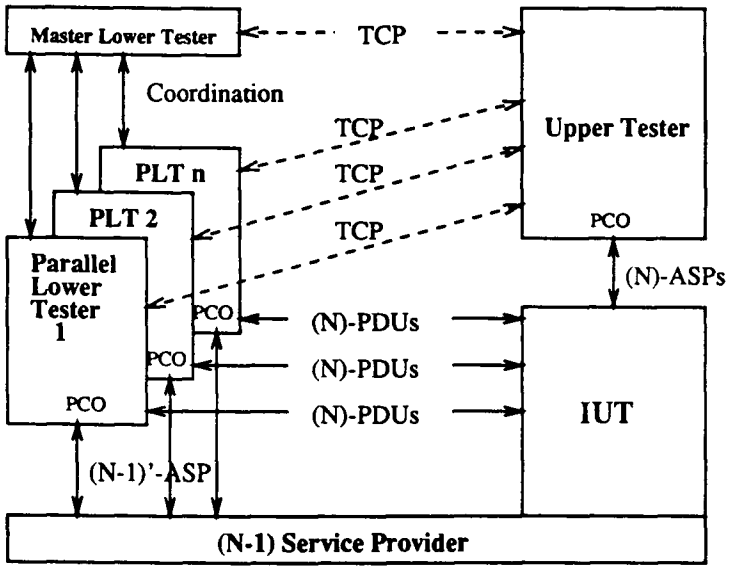


Figure 5.1: Multi-party Test Configuration. Multiple parallel lower tester communicating with a single IUT

In a multi-party environment the IUT is composed of all the components necessary to activate the protocol, and as such these components must be conformance tested. Testing of such a protocol may require direct and indirect observation because the action of this protocol may cause subsequent actions at other protocol entities involved in the network (*e.g.*, routing protocol and MHS). Under this situation, the results of these

actions may only be observed and reported by third parties. Hence, there is a need for a test method which involve multiple peers with indirect observation and control in order to observe the behavior of the IUT and synchronize the generation of inputs and observations of responses among the components of the test system. This brings up many issues concerning the Points of Control and Observation (PCOs), synchronization problem and the relationship between the upper and lower testers.

The second issue is related to the Points of Control and Observation (PCOs). One of the requirements in multi-party testing is to test the IUT whether it is capable of concurrently handling more than one connection or association for communicating with its peers. Each peer entity can be uniquely identified by the connection or association established. In the ISO distributed test method, the PCOs are between the lower tester and the (N-1) service provider and between the upper tester and the IUT. A PCO may represent a service access point, a connection end point, or both. It is the point at which a group of related interactions occur between the test system and the IUT (*i.e.,* directly) or via the underlying service provider (*i.e.,* indirectly). In a multi-party test environment, there may be one or more PCOs per lower tester. Each PCO represents a logical connection or association with its peer protocol entity within the IUT. It allows the lower tester to have remote control and observation of the ASPs and PDUs sent to or received from the IUT. Since a PCO in the ISO external test methods has the same functional characteristics as that in the multi-party test environment, it is therefore possible to extend the current ISO external test methods for multi-party testing.

The third issue relates to the classification of the system under test and the test method employed. In defining an appropriate test method for a multi-party test system, one important factor which must be considered is whether the classification of the SUT is an end-system (7-layer open or partial (N)-open system for layers 1 to N, where N is less than 7), or intermediate system (Network relay-system with layers 1 to 3, or Application relay-system with layers 1 to 7). In testing an intermediate system, for instance the routing protocol [ISO-3], there are no PCOs for control and observation above the IUT to allow for test coordination between a lower tester and an upper tester if the ISO's coordinated or distributed test method were employed. In fact, the only PCOs that can

be used to achieve test coordination and observation of the IUT are in third parties. This suggests a need for an extension to the existing external test methods in order to fulfill the test purposes. What is needed is a test method where there are multiple observers which behave as a set of peer entities to the IUT, and each uses a test management protocol to report observations to a single site.

The fourth issue is that the relationships between the lower testers (LTs) and upper testers (UTs), the LTs and IUT, and the UTs and IUT are dependent on the protocol to be tested and the test purposes of the test cases. For example, in testing the message transfer layer of MHS, three lower testers and one upper tester may be required to test whether the IUT can behave both as a relay and a recipient. In this case, the upper tester acts as a user agent, one lower tester acts as the originator and the other two act as the recipients. No upper tester, however, is required in testing the routing protocol.

In summary, the requirements of testing certain protocols indicate the need for a multi-party testing methodology which has not been addressed in Parts 1 and 2 of the OSI Conformance Testing Methodology and Framework [ISO-1]. The following section presents a case study about the extended ferry clip test approach for multi-party testing using MHS as an example.

5.2 Application of the Ferry Clip Approach to Multi-party Conformance Testing

To perform conformance testing on a MHS implementation, single-party test methods are hardly adequate due to the testing requirement for the following MHS functionalities:

- *IUT as Relay (Relay Testing)*
  Test the ability of the IUT to make use of its routing tables for relaying a message to two (or more) different management domains (*i.e.,* multi-destination delivery).

  - Lower Tester 1 originator
  - Lower Tester 2 recipient 1
  - Lower Tester 3 recipient 2

- *IUT as Relay and Recipient*
  Test the ability of the IUT to behave as a relay and a recipient. The IUT receives a message for two recipients, one on the IUT and the other requires the IUT to create a second copy and relay it to another management domain.

  - Lower Tester 1 originator
  - Upper Tester 1 recipient 1
  - Lower Tester 2 recipient 2

- *IUT as Originator and Recipient*
  Test the ability of the IUT to behave as an originator and a recipient. This test requires the IUT to send a copy of the message to its local recipient and another copy to the tester.

  - Upper Tester 1 originator
  - Upper Tester 2 recipient 1
  - Lower Tester 1 recipient 2

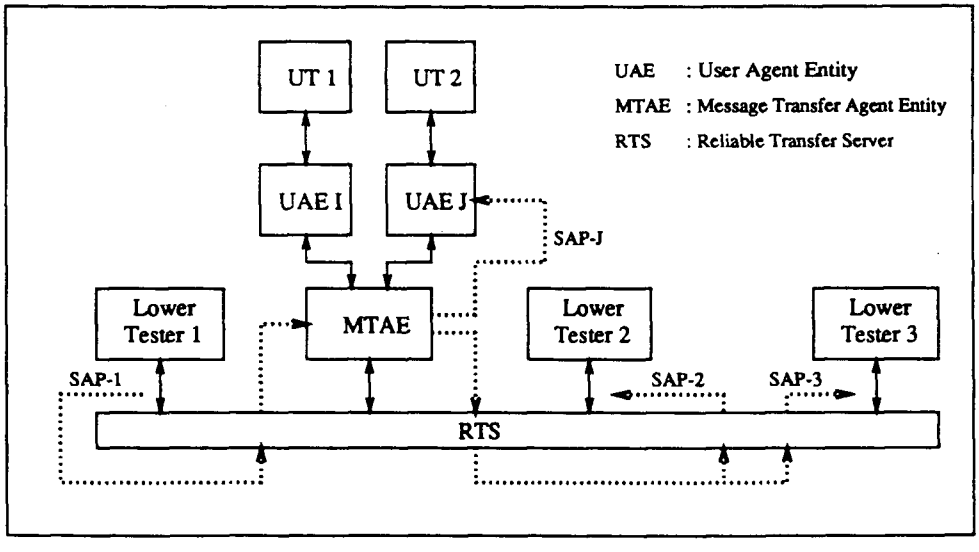


Figure 5.2: General test configuration for message transfer layer of MHS

Figure 5.2 depicts the logical view of a multi-party test configuration which can be used to fulfill the requirements for testing the message transfer layer of MHS. Any test

system that is used for testing the message transfer layer of MHS should be configurable to provide the functionalities depicted in the figure. With this general test configuration, there are many ways to perform multi-party conformance testing. It is therefore important to identify the basic mechanisms which must be supported by a test system. It is essential for a test system to:

- provide a method with indirect observation and control in order to observe the behavior of the IUT and synchronize the test execution between the components of the test system;
- support various configurations and topologies, *i.e.*, be able to simulate the change of physical topology by logically enabling or disabling the desired connections or entities within the test system;
- avoid dependency on the IUT for transmission of test management and coordination PDUs, and reporting of results observed, thereby achieving a reliable and controllable test environment.

The ferry clip based test architecture shown in Figure 2.3 is proposed for the realization of multi-party testing. In order to fulfill the requirement of multi-party testing, it is essential for the test system to support the communications between multiple entities on the SUT and the test system via the use of several test connections in parallel. The following components are involved in the support of multiple test connections in the ferry clip based test system: (1) Service Provider Interface Module, and (2) Service Interface Adapter. Multiple connections between the SUT and the test system can be achieved through the use of virtual circuit connection and multiplexing facility provided by the underlying communication service. Since each PCO represents a logical connection established by the IUT to communicate with a unique peer entity, this requires the Service Provider Module to associate each PCO with a unique identifier which is modelled as a peer entity to the IUT. Similarly, by applying downward multiplexing strategy, multiple test connections between the IUT and the Service Interface Adapter are mapped onto a

single ferry channel. Thus, an additional byte is introduced in the ferry PDU header to identify the connection the test data is associated with.

With this test architecture, a set of multi-peer entities can be simulated within the test system. The test system may simultaneously simulate several management domains as required by the test cases. For example, in testing the relay functionality of the IUT, one of the PLTs in the test system can behave as an originator in one of the simulated domains by sending a message via one of the test connections to the IUT. Another PLT in the test system simulates the behavior of the recipient in another domain waiting for the message to arrive via another test connection. Since the test system has overall control over the communication behavior of the IUT, simulation of the change of network topology can easily be achieved without any modification to the real testing environment. Since the test system has access to the service interfaces of its multiple peers, it can remotely control and observe the behavior of the IUT. Simulation of invalid peer behavior can also be achieved by injecting invalid test data under the direction of the test cases. Hence, the central control mechanism, inherited in the ferry clip based test architecture, enables the test system to remotely conduct, monitor and synchronize test components with great flexibility. Furthermore, the lower and upper testers, which are required in the conventional distributed test method, are being merged into a single tester which resides in the test system, thereby eliminating the problem of synchronization between testers residing on different machines.

5.3 Multi-party Conformance Testing of a MHS Implementation

This section illustrates how the ferry clip approach can be extended to perform multi-party testing, an example using the test system to test an implementation of MHS serves to demonstrate the power, generality and flexibility of the design concept described in Chapter 3. Conformance testing of a protocol implementation such as MHS is by nature too complex to be fully presented in this thesis, we therefore focus our attention on issues that are related to the testing aspects of the Message Transfer Layer.

5.3.1 Overview of MHS Model

The Message Handling System (MHS) model defined in X.400 (1984) series recommendation [CCITT-1] is shown in Figure 5.3. The lowest level in the MHS model is the Message Transfer System (MTS) where the Message Transfer Agents (MTA) are responsible for accepting messages from the originator (*i.e.*, User Agents), relaying the messages from MTA to MTA, and delivering them to the recipients (UAs).

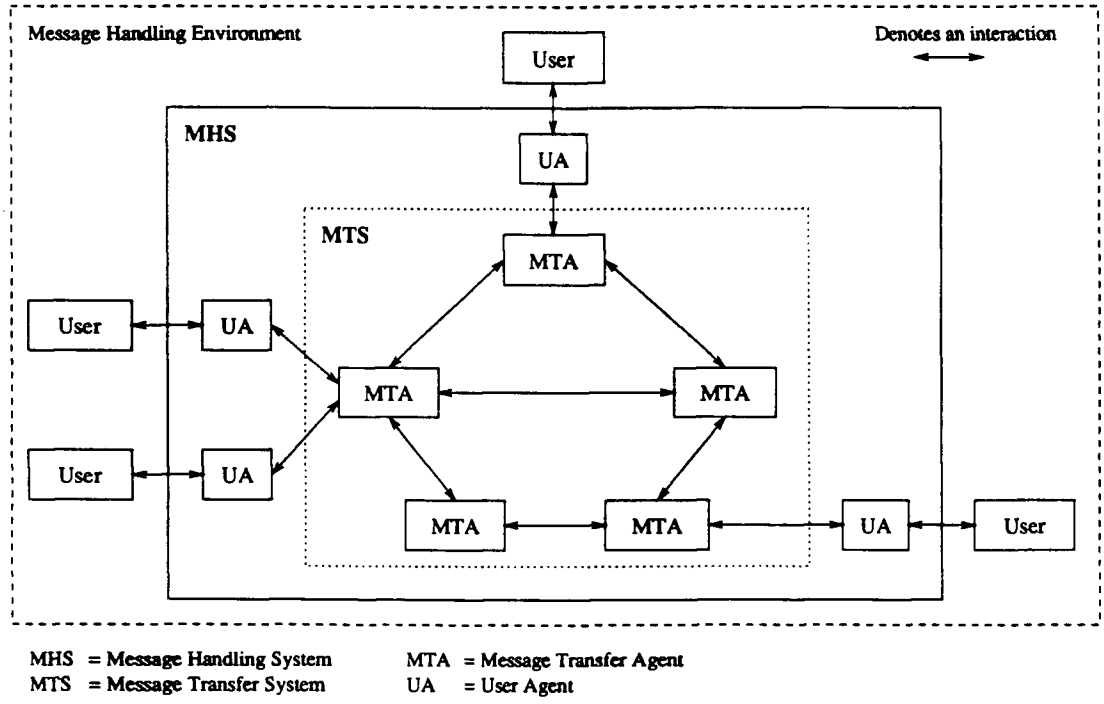


Figure 5.3: The Model of the Message Handling System

The next level, the Message Handling System (MHS), contains the User Agents (UAs) which interact directly with their users. The UAs assist the user in constructing and submitting messages to an MTA for transmission to the destination(s). In addition, the UAs also support other message functions such as filing, replying, retrieving, and forwarding.

A message that is transferred by the MTS is composed of an envelope and a content. The envelope part contains the originator and recipient addresses as well as other

necessary information that influence the transfer of the message. The content part of a message is completely transparent to the MTS except in the case where the content conversion service is requested (*e.g.*, text to facsimile).

The message handling facility is primarily an application-layer service. As such, it must rely on lower-layer entities to provide a complete communications capabilities. The architecture which incorporates the X.400 service is illustrated in Figure 5.4 [STAL89]. The application layer is divided into two sublayers: the Message Transfer Layer (MTL) and the User Agent Layer (UAL). Based on the functional model, three types of systems can be distinguished: system that contains only UA functions, system that contains only MTA functions and system that contains both UA and MTA functions.

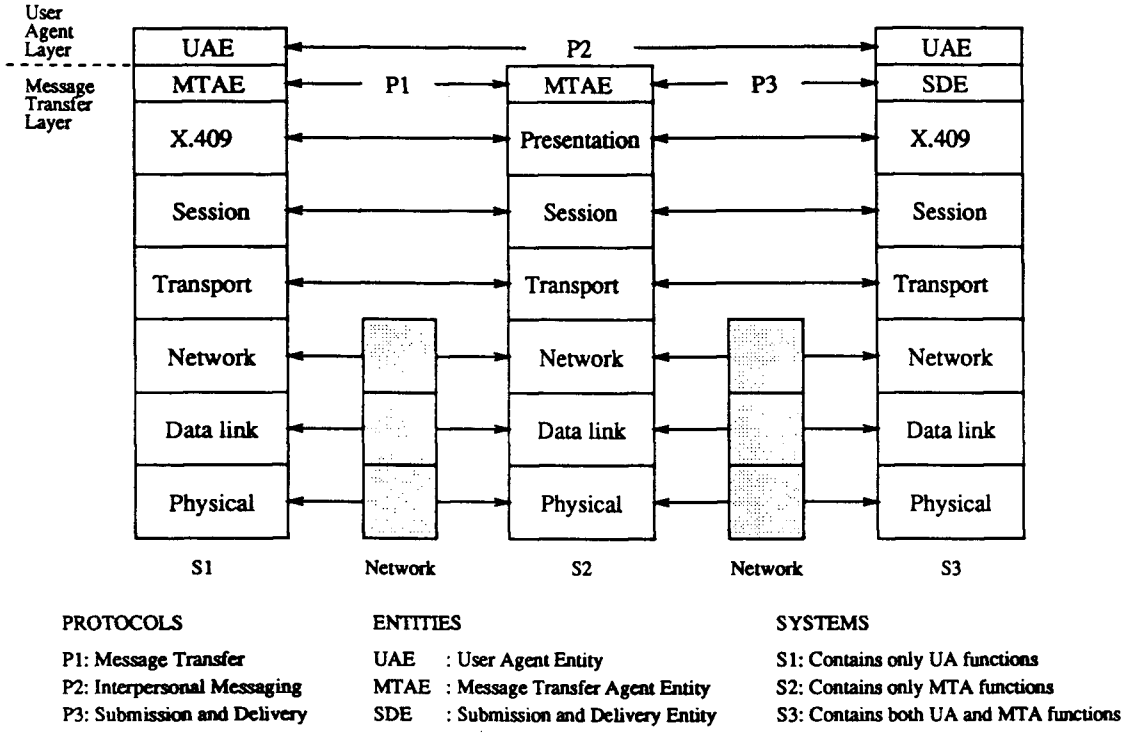


Figure 5.4: X.400 protocol architecture

The User Agent Layer contains *user agent entities* (UAEs). A UAE embodies the protocol-related functions of a single UA. UAEs cooperate with one another to support the services of the UAL known as *interpersonal messaging services* (IPMS). The protocol

that is used for interactions between UAEs is known as the *interpersonal messaging protocol* (P2). The P2 protocol essentially consists of protocol elements and rules that a UAE must follow in providing the interpersonal messaging services.

The Message Transfer Layer contains two kinds of entities: *message transfer agent entities* (MTAEs) and *submission and delivery entities* (SDEs). An MTAE embodies the functionality of a single MTA. MTAEs cooperate with one another to support the services of the MTL. If an MTAE presents in a system that contains only the MTA functions (see Figure 5.4), it acts solely as a relay. The protocol that is used for interactions between MTAEs is known as the *message transfer protocol* (P1). Basically, P1 defines the relaying of messages between MTAs and other interactions necessary to provide the MTL services.

The submission and delivery entity (SDE) makes the services of the message transfer layer available to its UAE. The SDE does not itself provide the message transfer services but rather interacts with its peer MTAE to allow the UAE to remotely invoke the services of the MTAE. The protocol that is used for interactions between MTAEs and SDEs is known as the *submission and delivery protocol* (P3). This protocol, however, achieves no significance, and hence has never been implemented in practice [SCH89].

5.3.2 Characteristics of MHS

Testing of MHS implementations, or OSI Application layer protocols in general, has certain characteristics that are not encountered in testing the OSI lower layer protocols [BOCH86, SARI89]. These characteristics are:

(a) Basically there is a one-to-one correspondence between the application PDUs and the service primitives. In addition, the rules for determining the order in which the PDUs are executed are largely independent of the rules for selecting the PDU parameter values. The same independence property also applies to the service primitives and the order of their execution. The one-to-one correspondence between the PDUs and service primitives implies that the protocol has a simple control flow, and the synchronization problem during testing is also simplified [SARI84] since the order of their (*i.e.*, ASPs/PDUs) execution are independent.

(b) In contrast to the rules for determining the order of execution for PDUs and service primitives, the rules concerning the selection of appropriate parameter values for PDUs are relatively complex. Together with the property (a) above, it is important to point out that the main concern to the test system is the structure of the PDUs, not the correlation of PDUs.

(c) ASN.1 and its associated encoding rules are used extensively within the OSI application layer protocols since the PDUs, coded in transfer syntax format, are exchanged over the boundary between different protocol sublayers. This results in the development of various ASN.1 support tools to assist and simplify the implementation and testing of application layer protocols. Hence, the proposed ferry clip based test system which uses ASN.1 for the representation of information (*i.e.*, ASPs) exchanged between the SUT and test system is well suited for testing OSI application layer protocols.

5.3.3 Test Results

The MHS implementation, based on the 1984 X.400 series of recommendations, has been successfully tested using the proposed ferry clip based test system. Most of the problems encountered were due to: (1) incompatibility in the encoding of ORNames which are used to uniquely identify the MTAs and users within an MTA; and (2) misinterpretation of the optional and default fields as defined in the ASN.1 specifications for P1/P2 PDUs. All these errors were rectified by analyzing the detailed trace log produced by the test system. A sample executable test case and the corresponding conformance log are shown in Figures E.2 and E.3 respectively in Appendix E.

5.3.4 Issues and Experiences

A number of issues have been identified from our experience in the conformance testing of a MHS implementation using the ferry clip based test system. Although many of these are recognized in the literature, they are summarized here as a result of our experiences.

- **Testing with respect to MHS**

  Based on the X.403 conformance testing specification manuals [CCITT-2], three functional groupings of services in MHS have to be tested: (1) the Interpersonal Messaging Service (P1), (2) the Message Transfer Service (P2), and (3) the Reliable Transfer Service (RTS). Properties (a) and (b) described in Section 5.3.2 are valid with respect to testing P1 and P2 protocols. The dynamic behavior of the test cases for P1 and P2 are less complex compared to that of the RTS. This means that the test system must be able to support the generation of all possible PDU structures, the detection of both syntax and semantic errors in the received PDUs.

  The main function of RTS is to provide a reliable message transfer service to its MTA. It utilizes services provided by the Session layer to maintain a "virtual association" with its peer entity and carry out the reliable transfer of application protocol data units (APDU). Thus, testing of RTS is different in several aspects from the testing of P1/P2 as well as other lower layer protocols. The main issue in RTS testing is its reliability, *i.e.*, the RTS must be tested for its ability to recover from various exception situations (*e.g.*, resume at some checkpoint after interruption or session abort). This requires the test system to have direct access to the internal behavior of the RTS implementation. There are several X.403 test cases which actually require such access in order to fulfill the testing purposes. This brings up the non-observability issue which is addressed below.

- **Non-observability**

  There are certain features of protocols which are not observable, and hence they are not testable. Such features can be classified into two categories: *pragmatically non-observable* and *theoretically non-observable* [MAT87].

  Pragmatically non-observable features are those which have a high cost associated with their testing. One example is testing the action of FTAM when a file server crashes.

  Theoretically non-observable features are those which are not directly testable by definition. Since conformance testing can be viewed as a *black box* testing, it should

not impose any requirement on the protocol behavior which cannot be triggered through some legal event at the exposed service interface of the IUT. One example is in testing the recovery mechanism of RTS, an access to the internal implementation of RTS is needed to trigger the RTS to invoke a session user abort request.

- **Misinterpretation of Protocol Specifications**

  Due to the complexity of protocols, protocols that are specified informally are likely to contain ambiguities and may lead to misinterpretations and inordinate amount of effort required to develop test cases. This implies that discrepancies between abstract test cases and the relevant protocol standards are not uncommon. For example, one discrepancy that has been identified, during the course of testing the RTS implementation based on the X.403 test case, is the **checkpointSize** parameter value specified during the RTS negotiation phase. It is stated in the X.400 series of recommendation that "a value of zero from the sending RTS invites the receiving RTS to select checkpointsize. A value of zero from the receiving RTS indicates that checkpointing will not be used. The value supplied by the receiving RTS becomes the agreed maximum value and governs both directions of transfer". However, it was assumed in the X.403 test cases that "a value of zero from the sending RTS indicates that no checkpointing will be done"; therefore, any value returned by the receiving RTS would be ignored and not used in the subsequent steps of those test cases.

  The use of formal specification techniques will likely lead to solutions in resolving specification dilemmas. As more and more protocols are specified using formal specification languages such as LOTOS or Estelle, test cases could be systematically derived (with the aids of automatic generation tool) from the specification. This reduces the probability of introducing errors into the test cases.

Chapter 6

The Ferry Clip Approach to Multi-party Interoperability Testing

As mentioned in Section 1.4, conformance testing alone is not a sufficient condition to guarantee interoperability of a protocol implementation. To further increase the probability that different implementations are able to interwork, interoperability testing can thus be considered the next pragmatic step. This chapter addresses the differences between conformance and interoperability testing in general, and describes the complementary characteristics of interoperability testing with highlights on why interoperability testing enhances confidence that different protocol implementations will interwork. The remainder of this chapter illustrates how the proposed ferry clip based test architecture can be extended to perform multi-party interoperability testing for a single- or multi-layer IUT.

6.1 Differences between Conformance and Interoperability Test Approaches

There are several aspects in which interoperability testing differs from conformance testing. The first notable difference is that interoperability testing usually involves multiple interconnected systems and networks that are functionally distinct and unique, but they provide common interfaces to allow an end-to-end connection to be established between an IUT and the involved systems. This connectivity test must be successfully completed

before full interoperability testing can be conducted. Second, one of the implementations involved is supposed to be the IUT, and the remaining participating systems are assumed to be conforming implementations which implement the same OSI protocol standards or subset of these standards. Another set up is to have the test system actively testing and monitoring the interactions between two IUTs purporting to support the same set of protocol standards. The test configuration must model as closely as possible to the actual operating environment, if testing on the users actual operation environment is considered to be infeasible. Third, as interoperability testing verifies the user-level functionalities in an end-to-end configuration, test cases required for interoperability testing are different from those which are applicable to conformance testing a particular protocol. It is therefore difficult or even impossible to expose the IUT to invalid behavior of its peers by injecting invalid test PDUs.

6.2 The Complementary Role of Interoperability Testing

This section examines the complementary characteristics of interoperability testing to conformance testing and highlights why interoperability testing enhances confidence that different protocol implementations will be able to interwork. Due to its intrinsic characteristics, interoperability testing addresses several issues which might not be covered in conformance testing.

1. *End-to-End Behavior and Applications*
   The protocol specification is developed based on the communication service to be provided by the protocol. Thus by its nature, it addresses only the required functionality between the two systems, and may not cover the end-to-end behavior and applications in complex configurations [BERT90]. In such cases, conformance testing of individual systems is insufficient to guarantee interoperability.

2. *Options in a Protocol Specification*
   There is often a tradeoff between the degree of flexibility and the tightness with

which a protocol is defined. An overly flexible protocol will result in mutual incompatibility among two or more implementations developed by independent vendors. This problem is further compounded by the complexity of protocols which makes it likely for implementers to introduce errors in the implementations, and that most protocols are not formally specified and may contain ambiguities. This results in diminishing chances of interoperability among independent implementations. Conversely, a protocol that is too tightly defined may impose unnecessary restrictions and reduce the flexibilities of a protocol architecture. Thus, a pragmatic approach is to define a range of options which the protocol is required to support. However, protocol options may result in incompatibility among independent implementations. Interoperability testing helps ensure that "mutual compatible" options have been selected from among the permissible ranges of choice.

3. *Timing and Synchronization*

   There is usually a set of timers associated with a protocol. Some of the timers are essential to ensure correct protocol operation; others, such as time-out intervals for retransmission, are used for the purpose of achieving optimum performance with respect to the given network characteristics. Usually a range of timer values is specified to take into account for differences such as propagation delay and error characteristics of the medium. There is usually a synchronization pattern during the initialization of a protocol process, this synchronization pattern has an adjustable duration. Thus, two communicating entities relying on the pattern must select a synchronization duration that is agreeable by both communicating ends. Interoperability testing ensures the timer values and the duration of synchronization patterns (if any) are tested for integrity in a simulated end-to-end testing environment before being deployed to the real operating environments.

4. *Performance Characteristics*

   Another benefit of interoperability testing is that it provides a measure of performance characteristics of the implementation(s) being tested, such as its throughput and responsiveness under various conditions. Such a measure of performance char-

acteristics gives an estimate within a statistical bound of confidence. Such estimate has proved useful in assessing and ranking the performance among different implementations of the protocol.

6.3 Application of the Ferry Clip Approach to Multi-party Interoperability Testing

This section presents an approach to realization of multi-party interoperability testing using the ferry clip. Figure 6.1 illustrates how the proposed ferry clip based test architecture can be extended to allow for interoperability testing of a single- or multi-layer IUT. Using this architecture, no modification to the passive ferry clip (on the SUT site) is required. Changes to the test system site depend on the number of SUTs involved. If two SUTs are used as shown in Figure 6.1, two active ferry clips are needed with each AFC communicating with its corresponding passive ferry clip over the ferry channel. Hence, there is a one-to-one relationship between the AFC and PFC.

With this test architecture, the test system has control over the communication activities between the SUTs involved. The major advantage of adopting the ferry clip test approach is the centralized and synchronized control and management of a set of distributed observers (*i.e.*, passive ferry clips) which report observations to the central point of control. The central control mechanism provides the test system with the ability to control the distributed observers flexibly, collect observations from all the SUTs involved, analyze test results and produce test verdicts. This also allows a single test case to describe all activities required between the SUTs involved. All these activities can be automatically executed within the test system under the direction of executable test scenarios, thereby eliminating the need for operator intervention as required in other test methods where the ferry clip test approach is not used.

To enhance the error detection capability, it is possible to introduce a monitor analyzer, called *arbiter* [BOCH89], as shown in Figure 6.1, which observes the exchange of PDUs between the two protocol implementations. The use of an arbiter is to realize only

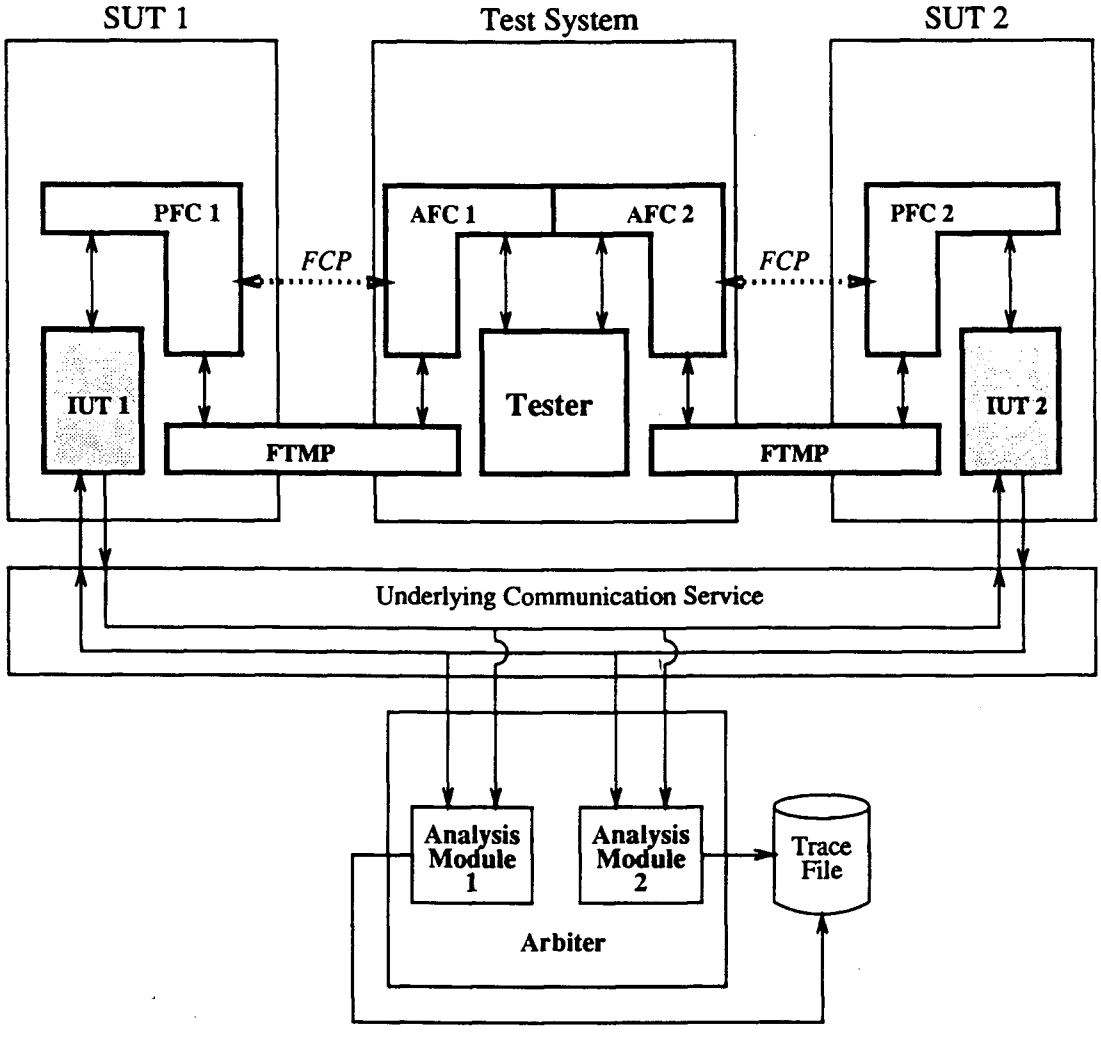


Figure 6.1: Ferry clip based test configuration for interoperability testing using two IUTs

the trace analysis function with the purpose of detecting error(s) which are not visible at the upper service interfaces of the implementations. Since only the PDUs exchanged and not the interactions at the upper service interfaces of the implementations are observed, the error detection power of the arbiter is equal to that of the remote test architecture. If two IUTs are involved as shown in the figure, the arbiter contains two trace analysis modules, one for each observed IUT. Each trace analysis module checks whether the observed trace is in contradiction to the specification of its corresponding IUT. The observed trace of PDUs is recorded in a trace file for later processing. A highly detailed log information can be produced for test results analysis, this is achieved by combining the test results captured by the ferry clip based test system together with the traces of PDUs taken by the arbiter. This is particularly useful in the case where a conclusion cannot be drawn based solely on the test interactions observed at the upper service interfaces of the implementations.

6.4 Related Work

As mentioned above, conformance testing alone does not yield a full guarantee of proper interworking, and users are demanding for an immediate and pragmatic approach to assess the interoperability of the products available in the marketplace. As such, a number of testing and research centers are starting to offer interoperability testing services. One example is the OSIRIDE-Interest initiative [1] [DILO89] which was promoted by the Italian National Research Council. Its objective is to investigate the interworking capabilities of products which implement the OSI and CCITT standards and recommendations, and are supplied by the OSIRIDE vendors (*i.e.,* Bull Digital, Hewlett-Packard, IBM, Olivetti and Unisys). This test approach assumes that each OSIRIDE vendor has already conformance tested his own products, and thus no reference testing center exists in this initiative.

Figure 6.2 depicts a conceptual model of the OSIRIDE-Interest test architecture with two SUTs (for illustrative purpose) involved. The *Scenario Generator* is in charge of

[1] OSIRIDE stands for "OSI su Rete Italian Dati Eterogenea" or "OSI on the Italian heterogeneous data networks"

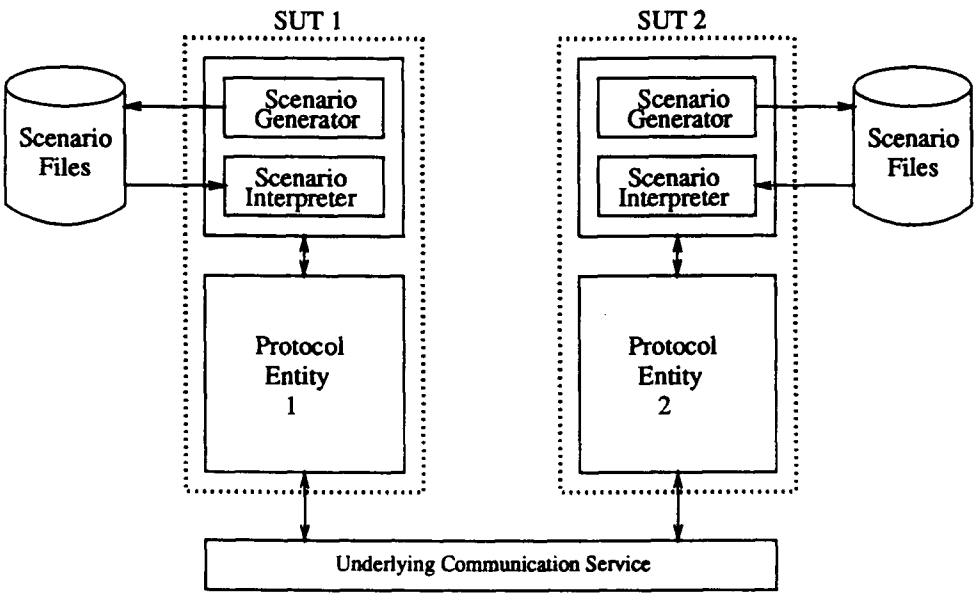


Figure 6.2: OSIRIDE-Interest test architecture for X.400 interoperability test

generating Scenarios (*i.e.*, files containing executable Test Sequences). This generation is performed off-line with respect to the Scenarios execution. The *Scenario Interpreter* takes as input the Scenario file and maps every Scenario service request into one (or more) specific service request(s) to the IUT being tested, the related parameters values given in symbolic form are also mapped into corresponding values required by the specific IUT. The interactions observed at the service boundary of the IUT are logged in a file which will be used for result analysis upon completion of each Scenario file processing. A result analysis is performed off-line by a tool named *Test Result Analyzer*, which rebuilds the sequences of service primitives observed and compares them with the expected ones.

One major disadvantage of the OSIRIDE-Interest test approach is that it does not have the central control mechanism which provides the ability to control communication activities between the SUTs, collect observed interactions at the related service interfaces of the IUTs from the distributed observers, analyze test results, and produce test verdicts within a single test system. Furthermore, base on the reconciled sequences of service primitives observed at each SUT, it is difficult and sometimes impossible to determine which IUT exhibits a deviation from the protocol specification. One possible solution

to this is to introduce a passive monitor (as described in the previous section), which observes the exchange of PDUs between the IUTs.

Many manufacturers, in addition to the national testing body described above, also seize any opportunity available to demonstrate to customers the interconnection capability of their products. One typical example is the interoperation verification services offered by IBM in Europe [BON89]. These services allow customers, vendors and other participants to verify the interoperability of their OSI implementations with the IBM licensed OSI products implementing the same OSI standards subset.

Figure 6.3 [BON89] shows the test configuration of IBM OSI-X.400 Interoperation Verification Service. The users operate and has full control over the verification process from the service terminal installed in their premises. The test cases needed for the verification service are provided by IBM. The users select those they wish to run according to the capabilities of their implementations or certain functionalities they wish to highlight. The connection between the user's and IBM's OSI implementation is achieved through the existing public telecommunications X.25 service. Traces of incoming and outgoing PDUs that flow between the user's OSI and IBM's OSI systems are taken. These traced PDUs are formatted upon completion of each test run to provide the users with an efficient problem analysis mechanism.

As indicated in [BON89], practical experience has shown that these services are extremely beneficial to the standards efforts in general, and the participants in particular as they are assured of their investment made in OSI products. As such, interoperability testing services will remain an invaluable tool in the development of open, multi-vendor systems.

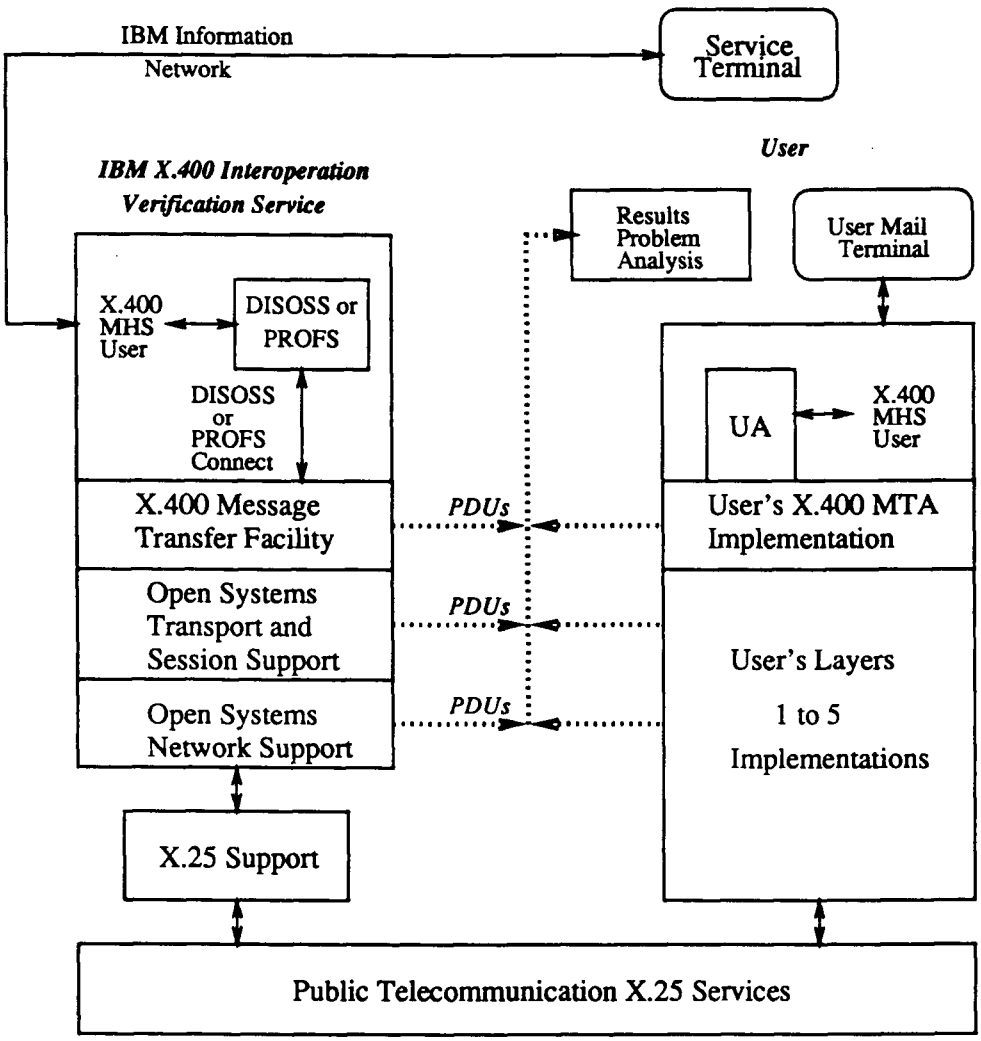


Figure 6.3: IBM OSI-X.400 Interoperation Verification Service

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis we have described the concept of multi-party testing and recognized the need for a multi-party testing methodology. In order to achieve a truly "open systems interconnection" in a multi-vendor environment, there are two different phases of protocol testing that need to be addressed: *multi-party conformance testing* and *multi-party interoperability testing.* Conformance testing verifies that the external behavior of a protocol implementation complies with the specification; while interoperability testing verifies that different implementations of the same protocol (assume they comply to the same standard) do interwork. Thus, both conformance and interoperability testing are necessary and are supplementary to each other. This calls for the design of a general test architecture that can be configured as required to meet the purpose of multi-party conformance testing and interoperability testing. The proposed ferry clip based test architecture meet this requirement.

The ferry clip based test architecture employs a set of carefully chosen design principles to achieve the goal of a generalized and flexible test tool. It is general and flexible with respect to the protocol to be tested, the test configuration and the test method to be realized, the system under test, and the underlying communication system. In addition, by adopting an ASN.1 representation of service primitives and ferry data that are to be

exchanged between the active ferry clip (in test system) and the passive ferry clip (in SUT) enhances the portability of the testing software.

The use of test system to test a MHS implementation serves to illustrate the applicability of the ferry clip test approach to multi-party conformance testing. The ferry clip test approach also offers many benefits in terms of the amount of effort saved and the cost of resources required to achieve a highly controlled test environment.

7.2 Future Work and Research Directions

Though we have implemented the ferry clip based test system for multi-party testing, the proposed extension for multi-party interoperability testing has not been implemented and tested. This would provide an interesting area for future study.

One of the objectives that has been identified in this thesis is to achieve an integrated and automated test environment where the process of deriving executable test cases, test execution and test results analysis can be automated using a set of support tools. There are several areas that remain open for future research. First, we have proposed the representation of TTCN test cases in an executable test tree notations. This allows a protocol-independent test engine to carry out the protocol testing function. The process of translating the dynamic behavior of TTCN test cases into executable test trees is done manually to a large extent, although a small set of support routines have been developed to assist the translation. The process of translation can, however, be automated with the aids of a TTCN parser. The input to the TTCN parser is a TTCN-MP (Machine Processable) source file containing the declarations, constraints specification and dynamic behavior of a test suite.

Second, the constraints of test ASPs/PDUs of a TTCN test suite are represented internally using an adaptation of E-nodes. The process of generating the E-node representation of ASPs/PDUs specified in ASN.1-MM (Modular Method) can be done automatically using a set of ASN.1 support tools. The development of a useful and complete set of ASN.1 support tools is a major undertaking, it poses an interesting area for future

work. This ASN.1 tools set shall consist of the following major components: *parser, editor, encoder/decoder,* and *analyzer.* The function of a parser is to analyze the ASN.1 specifications of ASPs/PDUs and generates the corresponding template E-node trees. The editor serves to create and manipulate the data instances for any given template E-node tree. To ease the task of editing, it would be ideal for the editor to provide a full-screen menu-driven user interface. Besides providing editing facilities for value E-node trees, it should provide mechanisms to override the structure and encoding information of a value E-node tree. These capabilities are extremely useful in protocol testing as they allow test suite designers to create semantically or syntactically invalid ASPs and PDUs for robustness testing. The generic encoder/decoder (which have been implemented and incorporated in our ferry clip based test system) basically transforms a value E-node tree into a sequence of octet stream and vice versa. The analyzer is capable of checking the syntax and semantic for a given stream of octets which is encoded based on the BER. Checking the syntax includes both the transfer syntax and the ASN.1 abstract syntax. To check for correct semantic, the ASN.1 support tool must allow the specification of constraints for a given template E-node tree by means of the editor. The constraints specification is used by the analyzer to determine the semantic correctness of the value E-node. The functions of the analyzer is particularly useful for the development of an automatic test system in which the verification of the parameter values of ASPs/PDUs can be done automatically.

The third area concerns the specification of constraints for ASPs/PDUs. Our current implementation is only capable of handling constraints that are described in the ASN.1 Modular Method of the ISO TTCN. There is still a large class of constraints which cannot be fully expressed in the current version of ASN.1; they are only stated in text part of the standards. As such, an extension of ASN.1 called "Attributed ASN.1" [BUR89] has been proposed in which constraints on instances of basic ASN.1 can be stated in a more precise and general manner. Because the extension is based on a well known formalism in language theory called " attribute grammars", it is feasible to incorporate this feature in the ASN.1 support tools. Such tools are of great help in the automatic implementation of test cases as well as the automatic execution of test cases.

Bibliography

[AHO90] A.V. Aho, B.S. Bosik, and S.J. Griesmer, "Protocol Testing and Verification within AT&T", *AT&T Technical Journal*, Vol. 69, No. 1, 1990.

[BERN90] L. Bernstein, "Protocols: Key to the Future of Computer Communication", *AT&T Technical Journal*, Vol. 69, No. 1, 1990.

[BERT90] H.V. Bertine et. al., "Overview of Protocol Testing Programs, Methodologies, and Standards", *AT&T Technical Journal*, Vol. 69, No. 1, 1990.

[BOCH86] G. v. Bochmann, M. Deslauriers and S. Bessette, "Application Layer Testing and ASN.1 Support Tools", in: *Proceedings of the GLOBECOM 86*, 1986.

[BOCH88] G.v. Bochmann, C.S He, "Ferry Approaches to Protocol Testing and Service Interface", *Proceedings of the 2nd International Symposium on Interoperable Information Systems*, Tokyo, Japan, Nov. 1988.

[BOCH89] G.v. Bochmann, R. Dssouli, J.R. Zhao, "Trace Analysis for Conformance and Arbitration Testing", *IEEE Transactions on Software Engineering*, Vol.15, No. 11, Nov. 1989.

[BON89] G. Bonnes, "OSI-X.400 Inter-operation Verification Services", *Proceedings of the 2nd International Workshop on Protocol Test System*, Berlin (west), Germany, 1989.

[BUR89] S.P. van de Burgt, P.A.J. Tilanus, "Attributed ASN.1", *Proceedings of the 2nd Formal Description Techniques for Distributed Systems and Communications Protocols (FORTE'89)*, Vancouver, Canada, Dec. 1989.

[CCITT-1] *CCITT Recommendations, Volume VIII - Fascicle VIII.7 (Red Book): Message Handling Systems*, 1984.
X.400 - MHS: System Model and Service elements
X.401 - MHS: Basic Service Elements and Optional user Facilities

X.408 - MHS: Encoded Information Type Conversion Rules
X.409 - MHS: Presentation Transfer Syntax and Notation
X.410 - MHS: Remote Operations and Reliable Transfer Server
X.411 - MHS: Message Transfer Layer
X.420 - MHS: Interpersonal Message User Agent Layer
X.430 - MHS: Access Protocol for Teletex terminals

[CCITT-2] CCITT Recommendations X.403, SG VII, "X.403 Message Handling Systems: Conformance Testing",
"MHS – Conformance Testing Specification Manual for IPMS(P2)", Nov. 1987
"MHS – Conformance Testing Specification Manual for MTS(P1)", Nov. 1987.
"MHS – Conformance Testing Specification Manual for RTS", Jan. 1988.

[CCITT-3] CCITT Recommendation X.400, Series Implementor's Guide (Version 3), 1986.

[CCITT-4] CCITT Recommendation X.200, "Reference Model of Open Systems Interconnection for CCITT Applications", 1988.

[CHAN89] R.I. Chan et. al., "A Software Environment for OSI Protocol Testing Systems", *Proceedings of the 9th IFIP Symposium on Protocol Specification, Testing, and Verification*, Enschede, Netherlands, Jun. 1989.

[CHANS89] S.T. Chanson, B.P. Lee, N.J. Parakh and H.X. Zeng, "Design and Implementation of a Ferry Clip Test System", *Proceedings of the 9th IFIP Symposium on Protocol Specification, Testing, and Verification*, Enschede, Netherlands, Jun. 1989.

[DILO89] F. Dilonardo et. al., "Experiences in the OSIRIDE-Interest Initiative: architecture and tools for X.400 interoperability tests", *Proceedings of the 2nd International Workshop on Protocol Test Systems*, Berlin (West), Germany, 1989.

[ESWA90] S. Eswara, et. al., "Towards Execution of TTCN Test Cases", *Proceedings of the 10th IFIP Symposium on Protocol Specification, Testing, and Verification*, Ottawa, Canada, Jun. 1990.

[FAV89] J. Favreau, R.J. Linn, and S. Nightingale, "A Formal Multi-Layer Test Methodology and its Applications to OSI", *Proceedings of the 2nd Formal Description Techniques for Distributed Systems and Communications Protocols (FORTE'89)*, Vancouver, Canada, Dec. 1989.

[ISO-1] ISO TC97/SC21, DIS 9646, "OSI Conformance Testing Methodology and Framework", Part 1: General Concepts, Part 2: Abstract Test Suites Specification, Part 3: The Tree and Tabular Combined Notation (TTCN), Nov. 1989.

[ISO-2] ISO/IEC JTC1/SC21/WG1, "OSI Architecture - Working Draft for Multiparty Test Methods", Florence Meeting, Nov. 1989.

[ISO-3] ISO/IEC/JTC1/SC26, End System to Intermediate System Routing Exchange Protocol for use in Conjunction with the Protocol for the Provision of the Connectionless-mode Network Service, DP 9542, Jun. 1987.
Intermediate System to Intermediate System Inter-domain Routing Exchange Protocol, Working Document, Oct. 1987.
Intermediate System to Intermediate System Intra-domain Routing Exchange Protocol, N4494, OCt. 1987.

[ISO-8824] OSI - Specification of Abstract Syntax Notation One (ASN.1), OSI DIS 8824, 1987.

[ISO-8825] OSI - Specification of Basic Encoding Rules for Abstract Syntax One (ASN.1), OSI DIS 8825, 1987.

[LAMP78] L. Lamport, "Time, clocks and the ordering of events in a distributed system", *Communication ACM*, Vol. 21, No. 4, Dec. 1978, 315-323.

[LEE89] B.P Lee, "Issues on Design and Implementation of Protocol Test System", M.Sc. thesis, Department of Computer Science, University of British Columbia, 1989.

[LINN85] R.J. Linn, "An Evaluation of the ICST Test Architecture after Testing Clas 4 Transport", *Protocol Specification, Testing, and Verification, IV*, North-Holland, 1985.

[LINN86] R.J. Linn, "Testing to Assure Interworking of Implementation of ISO/OSI Protocols", *Computer Networks and ISDN Systems*, 11 (1986) 277-286.

[MAT87] R.S. Matthews, K.H. Muralidhar, M.K. Schumacher, "Conformance Testing: Operational Aspects, Tools, and Experiences", *Protocol Specification, Testing, and Verification, VI*, North-Holland, 1987.

[MAT88] R.S. Matthews, K.H. Muralidhar, S. Sparks, "MAP 2.1 Conformance Testing Tools", *IEEE Transactions on Software Engineering*, Vol. 14, No. 3, Mar. 1988.

[NEU86] G. Neufeld, J. Demco, B. Hilpert, R. Sample, " EAN: An X.400 Message System", *Proceedings of IFIP Computer Message System'85*, Elsevier Science Publishers, North-Holland, 1986.

[PAR89] N.J. Parakh, "Design and Implementation of a Ferry Clip Test System", M.Sc. thesis, Department of Computer Science, Univerity of British Columbia, 1989.

[RAY87] D. Rayner, "OSI Conformance Testing", *Computer Networks and ISDN Systems*, 14 (1987) 79-98.

[SARI84] B. Sarikaya, G.v. Bochmann, "Synchronization and Specification in Protocol Testing", *IEEE Transactions on Communications*, Vol. COM-32, No. 4, Apr. 1984.

[SARI89] B. Sarikaya, "Conformance Testing: Architectures and Test Sequences", *Computer Networks and ISDN Systems*, 17 (1989) 111-126.

[SCH89] P. Schiker, "Message Handling System, X.400", *Message Handling Systems and Distributed Applications*, E. Stefferud, O.J. Jacobsen, P. Schiker (Editors), IFIP, 1989.

[SECH86] S. Sechrest, "An Introductory 4.3BSD Interprocess Communication Tutorial", MT XINU Manual, 4.3BSD with NFS, *Programmer's Supplementary Documents*, Vol. 1, PS1, 1986.

[SMITH89] B. Smith, "OSI Protocol Testing Environment Manual", Version 2.0, *UBC-IDACOM Project Documentation*, Mar. 1989.

[STAL89] W. Stallings, "Data and Computer Communications", Macmillan Publishing Company, New York, 1989.

[STOL89] W. Stoll, "KATE - A Test System for OSI Protocols", *Proceedings of the 2nd International Workshop on Protocol Test Systems*, Berlin (West), Germany, 1989.

[VEL89] R.J. Velthuys, J. Schneider, L.F. Mackert, "Protocol Conformance Testing with Communicating Rule Systems", *Proceedings of the 2nd International Workshop on Protocol Test Systems*, Berlin (West), Germany, 1989.

[ZENG86] H.X. Zeng, D. Rayner, "The Impact of the Ferry Concept on Protocol Testing", in: M. Diaz, *Protocol Specification, Testing, and Verification, V*, North-Holland, 1986.

[ZENG88a] H.X. Zeng, X.F. Du, C.S. He, "Promoting the *Local* Test Method with the New Concept *Ferry Clip*", *Proceedings of the 8th IFIP Symposium on Protocol Specification, Testing and Verification*, Atlantic City, Jun. 1988.

[ZENG88b] H.X. Zeng, Q. Li, X.F. Du, and C.S. He, "New Advances in Ferry Testing Approaches", *Journal of Computer Networks and ISDN Systems*, 15, 1988.

[ZENG89] H.X. Zeng, S.T. Chanson, B.R. Smith, "On Ferry Clip Approaches in Protocol Testing", *Computer Networks and ISDN Systems*, 17, 1989.

Appendix A

Ferry Clip Services and Ferry PDUs

Ferry Data Service Primitives (FD-ASPs)

FD-DATA request	(FD-DATAreq)	
FD-DATA indication	(FD-DATAind)	

Ferry Management Service Primitives (FM-ASPs)

FM-CONNECT request	(FM-CONNreq)	
FM-CONNECT confirm	(FM-CONNcnf)	
FM-DISCONNECT request	(FM-DISCreq)	
FM-DISCONNECT indication	(FM-DISCind)	
FM-CONTROL request	(FM-CNTLreq)	
FM-CONTROL confirm	(FM-CNTLcnf)	

Ferry Transfer Service Primitives (FT-ASPs)

FT-CONNECT request	(FT-CONNreq)	[test system only]
FT-CONNECT indication	(FT-CONNind)	[SUT only]
FT-CONNECT response	(FT-CONNrsp)	[SUT only]
FT-CONNECT confirm	(FT-CONNcnf)	[test system only]
FT-DISCONNECT request	(FT-DISCreq)	
FT-DISCONNECT indication	(FT-DISCind)	
FT-DATA request	(FT-DATAreq)	
FT-DATA indication	(FT-DATAind)	
FT-ERROR indication	(FT-ERRORind)	

Table A.1: Ferry Clip Service Primitives

Ferry Transfer Service	**Transport Service**	**Network Service**
FT-CONN req	T-CONN req	N-CONN req
FT-CONN ind	T-CONN ind	N-CONN ind
FT-CONN rsp	T-CONN rsp	N-CONN rsp
FT-CONNcnf	T-CONN cnf	N-CONN cnf
FT-DISC req	T-DISC req	N-DISC req
FT-DISC ind	T-DISC ind	N-DISC ind
FT-DATA req	T-DATA req	N-DATA req
FT-DATA ind	T-DATA ind	N-DATA ind
	T-EXPEDITED DATA ind	N-EXPIDITED DATA ind
FT-ERRORind		N-DATA ACK ind
		N-RESET ind
		N-RESET cnf

Table A.2: Mapping of FT-ASPs to Transport and Network Services

State	Simulating Event	Resulting Action	Next State
idle	FM-CONN req FM-DISC req	FT-CONN req none	connecting idle
	FM-CNTL req FD-DATA req	FM-DISC ind FM-DISC ind	idle idle
	FT-DISC ind FT-ERROR ind	none FT-DISC req	idle idle
connecting	FM-DISC req	FT-DISC req	idle
	FT-CONN cnf FT-DISC ind FT-ERROR ind	FM-CONN cnf FM-DISC ind EXCEPTION	connected idle idle
connected	FM-DISC req	FT-DISC req	idle
	FM-CNTL req FD-DATA req	FT-DATA req (FY-CNTL) FT-DATA req (FY-DATA)	connected connected
	FT-DATA req	P1: FD-DATA ind P2: control actions	connected connected
	FT-DISC ind FT-ERROR ind	FM-DISC ind EXCEPTION	idle idle

Notes:

. In any state, the action and transition taken for events not listed in the table are the asme as those listed for the FT-ERROR ind event, e.g. if an FM-CNTL req is received in the *connecting* state, the EXCEPTION action should be taken and the state should change to *idle*.

. EXCEPTION indicates the dual actions FT-DISC req and FM-DISC ind.

. P1 (predicate 1) -- the FT-DATA received is an FY-DATA PDU.

. P2 (predicate 2) -- the FT-DATA received is an FY-CNTL PDU; action is to process FY-CNTL flag bits and generate FM-CNTL cnf.

Table A.3: Active Ferry Clip State Transition Table

State	Simulating Event	Resulting Action	Next State
idle	FT-CONN ind	P1: FT-CONN rsp P2: FT-DISC req	connected idle
	FT-DISC ind FT-ERROR ind	none FT-DISC req	idle idle
	FD-DATA req	none	idle
connected	FT-DISC ind FT-ERROR ind	none FT-DISC req	idle idle
	FT-DATA ind	P3: FT-DATA req (loop back) P4: FD-DATA ind P5: control actions	connected connected connected
	FD-DATA req	P6: FT-DATA req (FY-DATA) P3: none	connected connected

Notes:

. In any state, the action and transition taken for events not listed in the table are the same as those listed for the FT-ERROR ind event.

. P1 (predicate 1) -- the incoming FT-CONN ind is acceptable.

. P2 -- the incoming FT-CONN ind is unacceptable

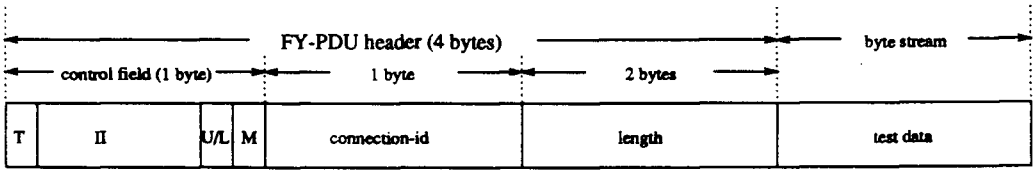
. P3 -- the passive ferry clip is in loop-back mode.

. P4 -- received data is FY-DATA PDU and the passive ferry clip is not in loop-back mode.

. P5 -- received data is FY-CNTL PDU and the passive ferry clip is not in loop-back mode. Perform appropriate actions and generate FY-CNTL (using FT-DATA req) back to active ferry clip.

. P6 -- the passive ferry clip is not in loop-back mode.

Table A.4: Passive Ferry Clip State Transition Table



T -- FY-PDU Type.
T = 0 for FY-DATA PDUs.

II -- IUT Identifier.
Identifies IUT / layer / sub-layer to which test data applies.

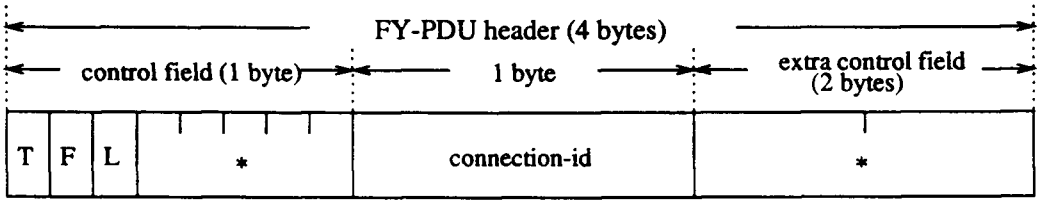
U/L -- Upper / Lower Interface Bit.
0 => test data to/from lower service interface of IUT.
1 => test data to/from upper service interface of IUT.

M -- More Segments.
0 => test data is not segmented or is last of a series of segments.
1 => at least one segment follows this one.

connection-id -- Test Connection Identifier.

length -- Number of bytes of test data within the FY-DATA PDU.

Figure A.1: FY-DATA PDU format



T -- FY-PDU Type.
T = 1 for FY-CNTL PDUs.

F -- Flow Control Bit.
0 => flow control off, 1 => flow control on.

L -- Loop-Back Bit.
0 => set passive ferry into normal (non loop-back) mode.
1 => set passive ferry into loop-back mode.

connection-id -- Test Connection Identifier.

* -- Reserved.

Figure A.2: FY-CNTL PDU format

Appendix B

The E-node Data Structure

Data Structure for the Value E-node

```
/*
 * Value Enode - used for both sending and receiving ASPs/PDUs
 */
typedef struct Enode
{
    Eid    id;                  /* class + cons/prim + type/tag */
    elen   length;              /* length of contents */
    unsigned char ctype;        /* constraint type (if it is constraint Enode) */
#   define C_NORMAL    0x00     /* normal absolute value */
#   define C_ANYSEQ    0x01     /* ANY_OR_OMIT (*), element can be any value */
                                /* or absent                                 */
#   define C_ANY       0x02     /* ANY (?), element can be any value */
#   define C_NONEXIST  0x03     /* OMIT (-), element must be absent */
#   define C_WILDCARD  0x04     /* character string contains wilcard char */
#   define C_INTEGER   0x05     /* E_INTEGER constraint */
#   define C_BITSTRING 0x06     /* E_BIT_STRING constraint */
#   define C_BOOLEAN   0x07     /* E_BOOLEAN constraint */
#   define C_RANGE     0x08     /* scalar range */
#   define C_TSP       0x09     /* test suite parameter */
#   define C_ATTACH    0x0A     /* attached constraint enode name */
#   define C_OPTIONAL  0x80     /* OPTIONAL if highest order bit = 1, */
#   define C_MANDATORY 0x00     /* MANDATORY if it is = 0. Use '|' operation */
                                /* to incorporate with the above types */

    struct Enode* next;         /* next elt (if part of constructor) */
    unsigned char cpar_flag_a;  /* constraint param no., if union a is one */
    unsigned char cpar_flag_b;  /* constraint param no., if union b is one */

    /* First pointer field: data type depends on constraint type */
    union
```

```
    {
        char*     primitive;    /* Normal Enodes: ptr to primitive */
        char*     tsp_name;     /* Constraint Test Suite param: name */
char*     bit_value;    /* Constraint Bitstring: value string */
        int       range_lo;     /* Constraint Range: lower limit */
        int       param_no;     /* Constraint param: parameter # */

        /* abbreviations for fields in this union */
#       define Prim     a.primitive
#       define tspname  a.tsp_name
#       define bitval   a.bit_value
#       define rangelo  a.range_lo
#       define parnum   a.param_no
    } a;

    union
    {
        struct Enode* constructor;      /* Normal Enodes: constructor tree */
        char*         bit_mask;         /* Constraint Bitstring: mask string */
        long          range_hi;         /* Constraint Range: upper limit */

        /* abbreviations for fields in this union */
#       define Cons     b.constructor
#       define bitmask  b.bit_mask
#       define rangehi  b.range_hi
    } b;
} Enode;
```

Appendix C

ASN1. Representation of ASPs and PDUs

```
ORName ::= [APPLICATION 0] IMPLICIT SEQUENCE {
    StandardAttributeList,
    DomainDefineAttributeList OPTIONAL}

STANDARDATTRIBUTELIST ::= SEQUENCE {
    CountryName OPTIONAL
    AdministrationDomainName OPTIONAL,
    [0] IMPLICIT X121Address OPTIONAL,
    [1] IMPLICIT TerminalID OPTIONAL,
    [2] PrivateDomainName OPTIONAL,
    [3] IMPLICIT OrganizationName OPTIONAL,
    [4] IMPLICIT UniqueUAIdentifier OPTIONAL,
    [5] IMPLICIT PersonalName OPTIONAL,
    [6] IMPLICIT SEQUENCE OF OrganizationUnit OPTIONAL}

CountryName ::= [APPLICATION 1] CHOICE{
    NumericString,
    PrintableString}

AdministrationDomainName ::= [APPLICATION 2] CHOICE {
    NumericString,
    PrintableString}
```

```
Contents of ORName enode:

41  [Appl 0] (CONS)
39  |   [Univ 16] (SEQ)
 8  |   |   [Appl 1] (CONS)
 6  |   |   |   [Univ 19] (PRINTABLE_STR) = "Canada"
    |   |   ^---
 8  |   |   [Appl 2] (CONS)
 6  |   |   |   [Univ 19] (PRINTABLE_STR) = "BC_TEL"
    |   |   ^---
10  |   |   [Cntx 3] = "UBC-IDACOM"
 5  |   |   [Cntx 4] = "Smith"
    |   ^---
    ^---

Size of enode tree = 43 bytes
```

Figure C.1: The ASN.1 definition of ORName and its corresponding E-node tree representation

Appendix D

Data Structure for Executable Test Suite

Data structure used for representing the
TTCN based Executable Test Suite

```
/*-------------------------------------------------------------------------
 *  PROGRAM: Executable Test Suite.
 *  MODULE : Data structure of executable test tree (ts_tree.h)
 *
 *  AUTHOR : Hendra Dany
 *  DATE   : Nov 16, 1989
 *-------------------------------------------------------------------------
 */

typedef char                    PCOID;
typedef short                   PROCID;

struct Send_Node {
  PCOID         pcoid;          /* PCO identifier */
  EID           asp_pdu_id;     /* ASP or PDU identifier */
  char          *consref;       /* constraints reference id */
  PROCID        proc;           /* procedure identifier */
};

struct Receive_Node {
  PCOID         pcoid;          /* PCO identifier */
  EID           asp_pdu_id;     /* ASP or PDU identifier */
  char          *consref;       /* constraints reference id */
  PROCID        proc;           /* procedure identifier */
};
```

```
struct Attach_Node {
  char          *tree_id;           /* attached tree identifier */
  PCOID         pco[MAX_PCO_LIST];  /* list of pco ids */
  PROCID        proc;               /* procedure identifier */
};

struct Timeout_Node {
  TID           tid;            /* timer identifier */
  PROCID        proc;           /* procedure identifier */
};

struct Start_Node {
  TID           tid;            /* timer identifier */
  PCOID         proc;           /* procedure identifier */
};

struct Cancel_Node {
  TID           tid;            /* timer identifier */
};

struct Read_Node {
  TID           tid;            /* timer identifier */
  PROCID        proc;           /* procedure identifier */
};

struct Otherwise_Node {
  PCOID         pcoid;          /* PCO identifier */
  PROCID        proc;           /* procedure identifier */
};

struct Goto_Node {
  short         label;          /* event node label */
};

struct Proc_Node {
  PROCID        proc;           /* procedure identifier */
};

typedef struct Tree_Header {
  char  tree_id[TREE_ID_LEN];   /* test tree name or identifier */
  char  type;                   /* tree type */
# define ROOTTREE       'R'
# define SUBTREE        'S'
# define DEFAULTS       'D'

  char          *hdrstart;      /* pointer to Tree_Header starting addr */
  char          *nodestart;     /* pointer to test suite starting node addr */
```

```
  short         numVisit;       /* number of times this tree being traversed */
  short         numNodes;       /* number of nodes of this test tree */
  short         numLevel;       /* number of levels of this test tree */
  PCOID         pco[MAX_PCO_LIST];      /* list of pco for this tree */
  PROCID        proc;                   /* procedure identifier */
  char          defaults[TREE_ID_LEN]; /* pointer to name of default tree */
  PCOID         defpco[MAX_PCO_LIST];   /* list of pco for default tree */
  PROCID        defproc;                /* default tree procedure identifier */
} Tree_Header;

typedef struct Tree_Node {
  u_char node;                  /* Test Suite tree node tag */
# define T_SEND       0x01
# define T_RECEIVE    0x02
# define T_ATTACH     0x03
# define T_START      0x04
# define T_CANCEL     0x05
# define T_READ       0x06
# define T_TIMEOUT    0x08
# define T_GOTO       0x09
# define T_OTHERWISE  0x0A
# define T_PROC       0x0B

  struct Tree_Node *sibling;    /* pointer to sibling node */
  struct Tree_Node *next;       /* pointer to sucessor node */
  short  tag;                   /* Test Suite node tag */
  short  level;                 /* level of this Test Suite node */
  short  label;                 /* event node label */

  char   ver_type;              /* verdict tag */
# define FINAL    'V'
# define PRELIM   'P'
  char   verdict;               /* verdict or preliminary result */
# define PASS     'P'
# define FAIL     'F'
# define INCONC   'I'
# define NONE     ' '

  union {
        struct Send_Node        send;
        struct Receive_Node     recv;
        struct Attach_Node      attach;
        struct Start_Node       start;
        struct Cancel_Node      cancel;
        struct Read_Node        read;
        struct Timeout_Node     timeout;
        struct Goto_Node        Goto;
```

```
        struct Otherwise_Node   other;
        struct Proc_Node        proc;
        /* abbreviation for fields in this union */
#       define Tsend             a.send
#       define Trecv             a.recv
#       define Tattach           a.attach
#       define Tstart            a.start
#       define Tcancel           a.cancel
#       define Tread             a.read
#       define Tout              a.timeout
#       define Tgoto             a.Goto
#       define Tother            a.other
#       define Tproc             a.proc
  } a;
} Tree_Node;
```

Appendix E

Derivation of Executable Test Suite

Test Identifier: 316.4.1.1

Summary: This test checks that the IUT can behave as a relay and a recipient.
The tester LT_1 sends a UMPDU for two recipients, one on the IUT and one which requires the IUT to relay the UMPDU to it.

DYNAMIC BEHAVIOUR				
DEFAULTS: LIB_General				
BEHAVIOUR DESCRIPTION	LABEL	CONSTRAINTS REFERENCE	COMMENTS	RESULTS
+LIB_Tester_open_TWA[ ]				
LT_1 ! R_XFERreq		TRNreq_10	1	
+LIB_Tester_ack[ ]				
LT_2 ? R_XFERind		TRNind_20	2	
UT1 ? MT_DELind		DELind_7	3	Pass
LT_2 ? R_XFERind		TRNind_21	4	
UT1 ? MT_DELind		DELind_7	3	Pass
UT1 ? MT_DELind		DELind_7	3	
LT_2 ? R_XFERind		TRNind_20	2	Pass
LT_2 ? R_XFERind		TRNind_21	4	Pass
UT1 ? MT_DELind		DELind_7	3	
+LIB_Tester_ack[ ]				
LT_2 ? R_XFERind		TRNind_20	2	Pass
LT_2 ? R_XFERind		TRNind_21	4	Pass

Comments:
1. UMPDU_1_20 for 2 recipients (UA & tester)
2. UMPDU_0_16 with 1 recipient
3. message for UAE
4. UMPDU_0_17 with 2 recipients

Figure E.1: An example of X.403 test case specified in TTCN

Contents of test case 'P1_316.4.1.1'

Test Id : P1_316.4.1.1[ ]
Tree Type : ROOT TREE
Num of nodes : 14 Num of levels : 5
DEAFULTS : LIB_General[ ]

+LIB_Tester_open_TWA[ ]			(Tag: 1, Level: 0)
LT_1 ! R_XFER_REQ_EID	TRNreq_10		(Tag: 2, Level: 1)
+LIB_Tester_ack[ ]			(Tag: 3, Level: 2)
LT_2 ? R_XFER_IND_EID	TRNind_20		(Tag: 4, Level: 3)
UT_1 ? MT_DELIVER_IND	DELind_7	[PASS]	(Tag: 5, Level: 4)
LT_2 ? R_XFER_IND_EID	TRNind_21		(Tag: 6, Level: 3)
UT_1 ? MT_DELIVER_IND	DELind_7	[PASS]	(Tag: 7, Level: 4)
UT_1 ? MT_DELIVER_IND	DELind_7		(Tag: 8, Level: 3)
LT_2 ? R_XFER_IND_EID	TRNind_20	[PASS]	(Tag: 9, Level: 4)
LT_2 ? R_XFER_IND_EID	TRNind_21	[PASS]	(Tag: 10, Level: 4)
UT_1 ? MT_DELIVER_IND	DELind_7		(Tag: 11, Level: 2)
+LIB_Tester_ack[ ]			(Tag: 12, Level: 3)
LT_2 ? R_XFER_IND_EID	TRNind_20	[PASS]	(Tag: 13, Level: 4)
LT_2 ? R_XFER_IND_EID	TRNind_21	[PASS]	(Tag: 14, Level: 4)

Figure E.2: An executable test tree representation of a TTCN test case

```
Thu Jun 21 19:56:18 1990

Execution of test case 'P1_316.4.1.1'
   Test Id      : P1_316.4.1.1[ ]
   Tree Type    : ROOT TREE
   Num of nodes: 14                Num of levels: 5
   DEFAULTS     : LIB_General[ ]
--------------------------------------------------------------------------------
19:56:18 +LIB_Tester_open_TWA[ ]  (Tag:1, Level:0)  <SUCCESS>
19:56:18 LT_1 ! R_OPEN_REQ_EID OPENreq_1  (Tag:1, Level:0)  <SUCCESS>
19:56:21 LT_1 ? R_OPEN_CFM_EID OPENcfm_1  (Tag:2, Level:1)  <SUCCESS>
19:56:21 LT_1 ! R_XFER_REQ_EID TRNreq_10  (Tag:8, Level:7)  <SUCCESS>
19:56:27 +LIB_Tester_ack[ ]  (Tag:3, Level:2)  <SUCCESS>
19:56:27 LT_2 ? R_OPEN_IND_EID OPENind_1  (Tag:1, Level:0)
*** PCO mismatch: test case pcoid=LT_2, observed pcoid=UT_1  <FAILURE>
*** Observed Event: UT_1 ? MT_DELIVER_IND_EID
19:56:27 UT_1 ? MT_DELIVER_IND_EID DELind_7  (Tag:11, Level:2)  <SUCCESS>
19:56:33 +LIB_Tester_ack[ ]   (Tag:12, Level:3)  <SUCCESS>
19:56:33 LT_2 ? R_OPEN_IND_EID OPENind_1  (Tag:1, Level:0)  <SUCCESS>
19:56:33 LT_2 ! R_OPEN_RSP_EID OPENrsp_1  (Tag:2, Level:1)  <SUCCESS>
19:56:38 LT_2 ? R_XFER_IND_EID TRNind_20 [PASS]  (Tag:14, Level:4)  <SUCCESS>

<<<*** Verdict for Test Case P1_316.4.1.1: [PASS] ***>>>
```

Figure E.3: A sample conformance log produced by the ferry clip based test system