DESIGN AND IMPLEMENTATION OF A FERRY CLIP TEST SYSTEM

By

NEVILLE J. PARAKH

B.Sc., University of British Columbia, Canada, 1987

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

June 1989

© Neville J. Parakh, 1989
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 **25 July, 1989**.
Abstract

In order for Open Systems Interconnection (OSI) to work, protocol implementations must be tested as to their conformance to the specifications they purport to adhere. The International Standards Organization (ISO) has defined a set of abstract test methods for the conformance testing of computer communication protocols. The Ferry Clip concept is a test approach to realize those test methods. It is also a powerful tool that can be used for the diagnostic testing of communication protocols.

This thesis describes the design and implementation of a Ferry Clip based test system in three different environments - UNIX, MPT and OSI-PTE. A method for structuring the system into a specialized set of modules is presented. Such a structuring scheme can considerably reduce the effort required to test different protocol implementations. Implementation issues encountered in building the system under the different hardware/software environments are discussed. The versatility of the Ferry Clip approach is further illustrated by presenting a scheme for its use in Multi-layer and Multi-party testing.
Abbreviations

ASP  Abstract Service Primitive
AFC  Active Ferry Clip
CCITT  International Telegraph and Telephone Consultative Committee
CPU  Central Processing Unit
CTM  Coordinated Test Method
DTM  Distributed Test Method
E/D  Encoder/Decoder
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
INET  Internet
IPC  Interprocess Communication
ISO  International Standards Organization
ITL  Idacom Test Language
LMAP  Lower Mapping Module
LT  Lower Tester
LTM  Local Test Method
MPT  Multi-port Protocol Tester
OSI  Open Systems Interconnection
PCO  Point of Control and Observation
PDU  Protocol Data Unit
PFC  Passive Ferry Clip
PT  Protocol Tester
PTE  Protocol Testing Environment
RTM  Remote Test Method
SAP  Service Access Point
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUT</td>
<td>System Under Test</td>
</tr>
<tr>
<td>TM</td>
<td>Test Manager</td>
</tr>
<tr>
<td>TMP</td>
<td>Test Management Protocol</td>
</tr>
<tr>
<td>UT</td>
<td>Upper Tester</td>
</tr>
</tbody>
</table>
# Contents

Abstract ii  
Abbreviations iii  
Contents v  
List of Tables vii  
List of Figures viii  
Acknowledgement ix  

1 Introduction 1  
1.1 Motivation 1  
1.2 Introduction to Protocols 2  
1.3 Introduction to Protocol Testing 3  
1.4 ISO Abstract Test Methods 6  
  1.4.1 Local Test Method 6  
  1.4.2 External Test Methods 7  
1.5 The Ferry Concept 9  
1.6 The Ferry Clip Concept 12  
1.7 Outline of Thesis 13  

2 Components of a Ferry Clip based Test System 15  
2.1 Ferry Services 15  
2.2 The Ferry Control Protocol 18  
2.3 The Ferry Transfer Medium Protocol 19  
2.4 Requirements on the System Under Test 20  
2.5 Test Manager and Encoder/Decoder modules 20  
2.6 Structuring the Ferry Clips 21  
  2.6.1 The Active Ferry Clip 21  
  2.6.2 The Passive Ferry Clip 22
List of Tables

2.1 Ferry Data Services .................................................. 16
2.2 Ferry Management Services ......................................... 17
2.3 Ferry Transfer Services ............................................. 18

3.1 Mapping between FT-ASPs and X.213 ASPs ......................... 40
3.2 Active Ferry Clip Module Sizes in MPT Environment .............. 43
3.3 Passive Ferry Clip Module Sizes in MPT Environment ............ 43
3.4 Passive Ferry Clip Module Sizes in OSI-PTE Environment ....... 43

5.1 Incorrect Local Test Script Fragment for Transport Disconnect Phase .......... 60
5.2 Correct Local Test Script Fragment for Transport Disconnect Phase ........... 61
5.3 Incorrect Local Test Script Fragment for LAPB Connection/Disconnection Phase 64
List of Figures

1.1 OSI Reference Model .................................................. 3
1.2 A Protocol Entity ..................................................... 4
1.3 Conceptual Testing Architecture ..................................... 5
1.4 Local Test Method .................................................... 7
1.5 Coordinated Test Method ............................................. 8
1.6 Distributed Test Method ............................................. 9
1.7 Remote Test Method .................................................. 10
1.8 The Loopback Ferry Approach ...................................... 11
1.9 A Ferry Clip based Test System .................................... 13

2.1 Components of a Ferry Clip based Test System .................... 16

3.1 The Passive Ferry Clip's LMAP Command State FSM ............... 33
3.2 The Passive Ferry Clip's LMAP Frames/Packets FSM ............... 34
3.3 Structuring the Passive Ferry Clip in the OSI-PTE Environment .. 38

4.1 The Ferry Clip Local Test Approach .................................. 46
4.2 The Ferry Clip Distributed Test Approach ......................... 47
4.3 The Ferry Clip Remote Test Approach ................................ 49
4.4 Extended Multi-layer Testing using a Ferry Clip based Test System 51
4.5 A Local Single Multi-party Test Method ............................ 53
4.6 A Ferry Clip based Local Single Multi-party Test Method ........ 55
4.7 A Modified FY-DATA PDU for Multi-party Testing ................ 56
Acknowledgement

First and foremost, I would like to extend a very special thanks to my supervisor Dr. Samuel T. Chanson for suggesting my thesis topic, for the many fruitful discussions we had and for always being there to advice and guide me through my research.

A special thanks to my partner and close friend Bernard P. Lee for his support and determination to see our project through to completion.

Thanks to Dr. H X. Zeng for his advice and encouragement at the early stages of my research and for finding the time to be the second reader for this thesis.

Thanks to Dr. D Rayner for his advice and encouragement. Thanks to B. Smith, I. Chan, H. See, S. Chan, V. Lee and C. Anderson for always finding the time to help me. Thanks is due to the support staff in the Department of Computer Science. A special thanks to Peter Phillips and Ming Lau for all the help they gave us is setting up our system.

Thanks to the Department of Computer Science at the University of British Columbia, the Natural Sciences and Educational Research Council (NSERC) and Idacom Electronics for providing me with facilities and financial support during my research.

Last but not the least, a heartfelt thanks to my parents and my brother for their endless help, support and love without which none of this would have been possible.
Chapter 1

Introduction

This chapter presents the motivation for our research. A brief introduction to protocols, OSI and some relevant features of protocol testing are presented. The Ferry and Ferry Clip concepts are also introduced, followed by a layout of the rest of the thesis.

1.1 Motivation

The International Standards Organization (ISO) has defined a set of abstract test methods for the purposes of protocol testing. The Ferry Clip [2][3][4] concept was recently introduced as a technique for realizing these test methods. Due to its recent introduction, little work has been done to study the problems that would face the implementor of a Ferry Clip based test system. The purpose of this research was to study the feasibility of using the Ferry Clip concept in building a test system. For generality in our results we chose to build the system in three different environments and use it to test a variety of protocol implementations [23]. To study the flexibility of the Ferry Clip concept we chose to study its applicability to Multi-layer and Multi-party testing.
1.2 Introduction to Protocols

The last decade has seen major advances in the field of computer communications, to a point whereby it has been referred to as the “communication era”. It is no longer acceptable for systems manufactured by a particular vendor to simply interface and communicate amongst themselves. Computers manufactured by different vendors must communicate reliably and efficiently with one another. To achieve this goal, standards have been established “to govern the physical, electrical and procedural characteristics” [5] of communication systems. Several standards-making bodies exist, two of the most influential being the International Standards Organization (ISO) and the International Telegraph and Telephone Consultative Committee (CCITT).

At the basis of the procedural characteristics of a communication system lies the notion of a “communication protocol”. A communication protocol (henceforth referred to as protocol) is a set of rules and conventions by which two separate entities communicate with one another. The task of providing reliable and efficient communication between two entities is far too complex to be handled by a single module. Hence, the technical committee of ISO in charge of information systems (TC97), developed a model for structuring such a system. The Open Systems Interconnection (OSI) model organizes the communication system into 7 distinct protocol layers [9] (see figure 1.1).

Each layer is entrusted with performing a subset of the overall communication tasks. It provides the layer directly above it with certain services, shielding it from the details of how these services are implemented [9]. Each layer in turn can use the services of the layer immediately below it. Communication between the layers is performed through well established communication points called service access points (SAPs), via the exchange of information units.
1.3 Introduction to Protocol Testing

Ensuring different implementations of the same protocol will interwork involves verifying that each of the implementations conforms to the same standard. Section 21 of ISO TC 97 [8] defines a conformance testing methodology and framework. The "complexity of most protocols makes exhaustive testing impractical on both technical and economic grounds" [8]. Hence, ISO and CCITT's goal is to develop a set of representative test cases called test suites for each OSI protocol standard. The test suites defined for a particular protocol are rarely "complete". Hence, conformance testing "cannot guarantee conformance to a specification since it detects errors rather than their absence. What it does do is give confidence that an implementation has the required capabilities and that its behavior conforms consistently in representative instances of communication" [8].
Figure 1.2: A Protocol Entity
The protocol being tested is referred to as the implementation under test (IUT) and the system on which it resides is known as the system under test (SUT). Figure 1.3 provides a conceptual view of the testing architecture. The upper tester (UT) provides control and observation of the upper SAP of the IUT as does the lower tester (LT) over the lower SAP of the IUT. There is need for cooperation between the UT and the LT. The rules for such cooperation are called the test coordination procedures [8].

Figure 1.3: Conceptual Testing Architecture

To complicate matters further, the implementor need not provide access to both SAPs of the IUT. Furthermore, several protocol layers may be implemented together without respect to the layer boundaries. At times it may even be necessary to perform testing via a system remote to the SUT, due to the fact that large and complicated testers cannot be implemented on an SUT with limited memory.
1.4 ISO Abstract Test Methods

To bring some order to the resulting chaos, ISO has defined four abstract test methods [8]. The choice of the appropriate test method depends on characteristics of the IUT and the SUT. For example, what SAPs access is provided to and whether the tester can reside in the SUT influence the choice of the appropriate abstract test method.

Of the four test methods described, one is local and the other three external. In the local approach, the points of control and observation (PCOs) are at the service boundaries above and below the IUT. With the external approach, the LT is located remotely from the SUT and connected to it via a link or network.

1.4.1 Local Test Method

In the local test method (LTM) the PCOs are located at the service boundaries above and below the IUT. "Test events are specified in terms of the ASPs above the IUT and the ASPs and protocol data units (PDUs) below the IUT" [8] (see figure 1.4).

The LTM has several advantages over the external test methods. Since both the UT and the LT reside on the same system, coordination between them is quite simple. In fact a test system which uses the LTM may combine both the UT and the LT into a single tester to simplify matter even further [23]. A higher degree of control and observation over the IUT is possible in the LTM than that allowed by the external methods. This is due to the fact that the LTM has direct access to both SAPs of the IUT. However, a disadvantage of the LTM, which will be addressed again later in this chapter, is that the testers need to reside in the SUT. As explained earlier this may be difficult to accomplish on an SUT with limited memory. Even when the SUT has enough memory, protocol implementors may not desire a large and complex tester
1.4.2 External Test Methods

ISO has defined three external test methods - the distributed, coordinated and remote test methods.

"The coordinated test method (CTM) (see figure 1.5) requires that the test coordination procedures used to coordinate the realization of the upper and lower testers be achieved by means of test management protocols" (TMPs) [8]. Work is currently in progress to define TMPs for the Transport and Session layer protocols.

The IUT need not possess a clearly defined upper service boundary. If one is present and is accessible, the UT may use it to interface with the IUT. If the upper service boundary of the IUT is accessible via ASPs the UT can use them to observe and control the IUT. If ASPs are not provided by the IUT (e.g. X.25 packet layer in the MPT environment), the UT must use whatever mechanism the IUT provides to access and control it.
It is important to note that the TMP is designed to coordinate the activities of the LT and UT. Furthermore, the TMP is dependent on the protocol layer being tested. Coordination between the LT and UT can be greatly simplified by placing them in the same system. However, straightly speaking, the use of the CTM still requires a now redundant TMP be used between the LT and UT. The main disadvantage of the CTM is that separate TMPs must be defined for each protocol layer to be tested. Hence, a considerable amount of the test software has to be modified when testing different layers.

The distributed test method (DTM) (see figure 1.6) required test coordination procedures to synchronize the UT and remote LT. However, these test coordination procedures are not standardized as in the CTM.

Unlike the CTM and DTM, the remote test method (RTM) (see figure 1.7) does not require specific functions of the UT. Hence, the RTM can be used in systems where access to the upper SAP of the IUT is not directly available. The RTM is favored by vendors since it places the
1.5 The Ferry Concept

According to Zeng [1], the DTM and CTM have become the dominant approaches due to “their applicability and comparatively complete testing capability”. The DTM is primarily used for testing Application protocols, whereas the CTM is mainly used for the testing of the middle layers of the protocol stack, such as the Transport and Session protocols [7]. “However, experience shows that the usual method of realizing these approaches (using a portable test
In an attempt to solve these problems Zeng introduced a concept called the ferry concept. In essence, the ferry concept replaces the UT in the SUT by a relatively simple piece of software called the passive ferry (see figure 1.8). The UT is now moved to the remote machine on which the LT resides. The passive ferry in the SUT interacts with another module on the remote system, the active ferry, to transfer test data between the IUT and the tester.

Synchronization between the UT and LT is simplified since they now both reside on the same system. At the same time the complexity of the UT can be enhanced since it is no longer limited by constraints on the SUT. Also, the portability of the software is greatly increased since now it is no longer necessary to rewrite the UT and LT when testing different protocol layers.
CHAPTER 1. INTRODUCTION

Figure 1.8: The Loopback Ferry Approach

Figure 1.8 provides a view of a ferry based test system. The “test connection between the LT and the UT is looped back through the IUT”. Both the test channel and the ferry channel go through the IUT” [1].

The active ferry is responsible for initiating the ferry channel at the start of a test session and bringing down the ferry channel at the end of a test session. The active ferry may also disconnect the ferry channel in the case where an error is detected during a test session.

The passive ferry does not initiate or bring down a ferry channel connection. However, in case of an error it too may disconnect the ferry channel. Its main purpose is to transfer data received by it from the remote UT to the upper SAP of the IUT. Similarly, data sent to it from the upper SAP of the IUT is sent back to the remote UT via the ferry channel.

An alternate approach suggested by Zeng is to use a separate protocol layer to provide the
CHAPTER 1. INTRODUCTION

ferry transport medium [1]. In this case the ferry channel no longer passes through the IUT. This approach is preferable since the “ferry channel will not be affected by the misbehavior of the IUT” [3].

1.6 The Ferry Clip Concept

The ferry method is limited by the “fact that observation and control of the lower service boundary of the IUT is obtained indirectly through a peer SAP of the IUT” [4]. Hence, some state transitions such as triggering incorrect events or sending data to an unconnected IUT cannot be exercised.

To overcome this problem Zeng introduced a new concept - the ferry clip concept [2][4] The ferry clip concept is an extension of the ferry concept to allow use of the LTM.

A ferry clip system consists of two major components, an active ferry clip (AFC) which resides in the test system, and a passive ferry clip (PFC) which resides in the SUT (see figure 1.9). The two ferries use the services of the ferry control protocol (FCP) (described in chapter 2) to transfer test data between the test manager (TM) in the test system and an external IUT residing in the SUT. The FCP provides a standard interface on top of some existing protocol, such as X.25, which actually transfers the test data and which shall henceforth be referred to as the ferry transfer medium protocol (FTMP). The PFC in the SUT attaches directly to both the upper and lower SAPs of the IUT, like a clip. This allows the testers in the remote system to observe and control both SAPs of the IUT. Hence, the ferry clip concept is a way in which the LTM, DTM and RTM may be realized [4] (see figures 4.1, 4.2 and 4.3).

Conformance testing can be done by the implementors during protocol development for diagnostic purposes [23]. Since diagnostic testing is performed by the vendor, all the available
SAPs may be used, including those the vendor does not wish to expose to the outside world. Hence, a ferry clip based test system (FCTS) is ideally suited for diagnostic testing and has been successfully used by us to test the X.25 packet layer protocol and the Transport Class 0 protocol [23].

1.7 Outline of Thesis

This thesis describes the design, implementation and application of a FCTS on different hardware/software environments. As of this date we have implemented the system on three different environments - Unix, MPT and OSI-PTE. Unix is a popular operating system which runs on many different mainframes and workstations; MPT [20] is a general-purpose protocol tester manufactured by Idacom Electronics; and the OSI-PTE [22] is a sophisticated test system currently being built at the University of British Columbia. This thesis will focus on the design
and experience gained in the implementation for the MPT and the OSI-PTE environments, with a brief mention of the Unix implementation.

Chapter 2 describes the ferry clip concept in more detail and provides a model for the structuring of the active and passive ferry clips in an FCTS. Chapter 3 describes the implementation of the FCTS in the three environments. Chapter 4 describes a scheme for extending the ferry clip concept to perform Multi-layer and Multi-party testing. Chapter 5 discusses potential problems and describes their solutions. Chapter 6 summarizes the thesis and provides direction for future research. Appendix A contains the state transition tables for the AFC and PFC together with diagrams for the format of a ferry data and control PDU. Appendix B contains the relevant source code for the PFC LMAP modules in the MPT and OSI-PTE environments.
Chapter 2

Components of a Ferry Clip based Test System

The previous chapter introduced the ferry clip concept. This chapter begins with a discussion of the ferry service and the FCP. The individual components of a FCTS are described in more detail followed by a model for structuring the active and passive ferry clips.

2.1 Ferry Services

The active and passive ferry clips provide services to their users and make use of the services provided by the underlying FTMP [4]. The services can be grouped into three categories.

1. Ferry Data Services

The ferry data services constitute those services provided by the active and passive ferry clips to the TM and the IUT for the transfer of test data. Two ASPs are provided (see table 2.1).

The FD-DATAreq ASP is used by the TM to send data to the AFC and by the IUT to send data to the PFC. The FD-DATAind ASP is invoked by the AFC to send data to the TM and by the PFC to send data to the IUT. The IUT is unaware of the existence of the
CHAPTER 2. COMPONENTS OF A FERRY CLIP BASED TEST SYSTEM

Figure 2.1: Components of a Ferry Clip based Test System

<table>
<thead>
<tr>
<th>ASP</th>
<th>Abbreviated as</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD-DATArequest</td>
<td>FD-DATAreq</td>
</tr>
<tr>
<td>FD-DATAindication</td>
<td>FD-DATAind</td>
</tr>
</tbody>
</table>

Table 2.1: Ferry Data Services
ferry data service. Any data sent to the upper or lower SAPs of the IUT is mapped onto a FD-DATAreq ASP by the IUT interface module (see section 2.6.2).

2. Ferry Management Service

The ferry management services are the services provided by the AFC to the TM for the management of the ferry channel and control functions. Six ASPs are provided (see table 2.2).

<table>
<thead>
<tr>
<th>ASP</th>
<th>Abbreviated as</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM-CONNECTrequest</td>
<td>FM-CONNreq</td>
</tr>
<tr>
<td>FM-CONNECTconfirm</td>
<td>FM-CONNcnf</td>
</tr>
<tr>
<td>FM-DISCONNECTrequest</td>
<td>FM-DISCreq</td>
</tr>
<tr>
<td>FM-DISCONNECTindication</td>
<td>FM-DISCind</td>
</tr>
<tr>
<td>FM-CONTROLrequest</td>
<td>FM-CNTLreq</td>
</tr>
<tr>
<td>FM-CONTROLconfirm</td>
<td>FM-CNTLcnf</td>
</tr>
</tbody>
</table>

Table 2.2: Ferry Management Services

The FM-CONN and FM-DISC ASPs are used to set up and bring down a ferry channel connection. The FM-CNTL ASPs allow the TM to change the state of the active and passive ferry clips. This can be used to perform functions such as loopback testing of the ferry channel and flow control between the two ferries (described in chapter 3).

3. Ferry Transfer Service

The ferry transfer service constitutes those services provided by the active and passive lower mapping modules (see sections 2.6.1 and 2.6.2) to their respective finite state machine modules (see sections 2.6.1 and 2.6.2). Nine ASPs are provided (see table 2.3).

The FT-ASP are mapped onto the ASPs of the underlying FTMP by the active and passive ferry clip's lower mapping modules. The FT-CONN and FT-DISC ASPs are used
CHAPTER 2. COMPONENTS OF A FERRY CLIP BASED TEST SYSTEM

<table>
<thead>
<tr>
<th>ASP</th>
<th>Abbreviated as</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT-CONNECTRequest</td>
<td>FT-CONNreq</td>
</tr>
<tr>
<td>FT-CONNECTIndication</td>
<td>FT-CONNind</td>
</tr>
<tr>
<td>FT-CONNECTResponse</td>
<td>FT-CONNrsp</td>
</tr>
<tr>
<td>FT-CONNECTconfirm</td>
<td>FT-CONNcfm</td>
</tr>
<tr>
<td>FT-DISCONNECTRequest</td>
<td>FT-DISCreq</td>
</tr>
<tr>
<td>FT-DISCONNECTIndication</td>
<td>FT-DISCind</td>
</tr>
<tr>
<td>FT-DATArequest</td>
<td>FT-DATAreq</td>
</tr>
<tr>
<td>FT-DATAindication</td>
<td>FT-DATAind</td>
</tr>
<tr>
<td>FT-ERRORindication</td>
<td>FT-ERRORind</td>
</tr>
</tbody>
</table>

Table 2.3: Ferry Transfer Services

The FT-DATA ASPs are used to send data between the two ferry clips. The FT-ERROR ASP is generated by the lower mapping module when it detects the occurrence of an error.

2.2 The Ferry Control Protocol

The purpose of the FCP is to provide a transparent, data transfer service between the two ferries, independent of the FTMP. The FCP is defined in terms of the ferry data, management and transfer services. Appendix A provides two tables which define the state transition rules for the active and passive ferry clips. Also provided in appendix A are the formats for the ferry data (FY-DATA) and ferry control (FY-CNTL) PDUs.

The FCP is functionally equivalent to the ISO Transport Class 0 protocol [16]. It has facilities for setting up and disconnecting a ferry channel connection. In the data transfer phase it has facilities for fragmenting and reassembling data packets. However, it has no provision for recovering from errors and thus requires a reliable FTMP.

It should be noted that replacing the FCP with the ISO Transport Class 0 protocol, or for
that matter any ISO protocol has several disadvantages.

The FCP is a very simple and compact protocol designed to be independent of the FTMP. Eliminating the FCP and building the ferry clips directly on the FTMP results in a rewrite of much of the ferry clips when the FTMP, which is not designed with the ferry clips in mind, is changed. Using a FCP has the advantage that only the AFC LMAP and PFC LMAP need to be rewritten. Furthermore, certain ferry based functions such as loop-back testing and identifying the IUT layer and SAP to which test data is to be sent need to be implemented even if the FCP is eliminated. These functions are provided by the FCP and hence little saving is made in eliminating the FCP.

The passive ferry clip has only two states. There are several advantages to keeping the PFC portion of the FCP simple. A simpler PFC control protocol results in a smaller PFC module. This is important since the PFC must reside in the SUT, including SUTs with stringent memory limitations. Since part of the PFC, namely the IUT interface module, needs to be rewritten for each new protocol to be tested, keeping the PFC simple reduces the amount of work that needs to be performed to test different protocols.

2.3 The Ferry Transfer Medium Protocol

In order to keep the FCP simple and easy to implement, certain requirements were placed on the FTMP. Since the FCP cannot handle lost, mangled or out of sequence packets, it is necessary that the FTMP guarantees end-to-end error free delivery of data and does not resequence data packets (i.e., Transport functionality according to the OSI/RM terminology\(^1\)). The protocol used as the FTMP may be connection-oriented or connectionless, and either stream (e.g., Unix

\(^1\)However, in the case that the tester and the SUT are directly connected, Data Link or Network layer functionality is sufficient.
2.4 Requirements on the System Under Test

In order to use the ferry clip approach the SUT should satisfy the following requirements. It must provide a communication medium for the transfer of information to and from the test system. Furthermore, the IUT must provide an interface through which the ferry clip can be attached. Ideally both the upper and lower SAPs of the IUT are exposed thereby allowing maximum degree of control and observation; this is, for example, often possible in diagnostic testing. However, the ferry clip concept can be used even if only one of the interfaces is exposed, as demonstrated by our testing of the X.25 packet layer in the MPT environment [23].

2.5 Test Manager and Encoder/Decoder modules

The TM is that component of the FCTS that oversees the operation of the system. It reads and executes the test script and logs all incoming and outgoing data exchanges for future analysis. Furthermore, it is the responsibility of the TM to continue or abort the execution of a test script if an abnormal condition is detected.

The TM communicates with the AFC through the E/D module (see figure 2). The TM uses the services of the AFC by means of the FM-ASPs. The test data is encoded by the E/D module so as to make it easy for the IUT Interface to convert the test data into a form that the IUT accepts. Similarly, data received by the AFC is sent to the TM through the E/D module so that it can be converted into a format the TM understands.

With the ferry clip approach, events for both SAPs of an IUT can be specified in the same test script. Furthermore, since both the UT and LT can be merged together within the TM
on the test system, the synchronization problems between the UT and LT do not occur in a FCTS.

The ferry clip concept could be extended to incorporate multiple ferry clips inside the SUT. Interaction and synchronization between different IUTs could be specified in a single test script. This could prove to be a useful feature for testing a protocol stack, allowing observation of the protocol exchanges at various layer boundaries.

2.6 Structuring the Ferry Clips

This section presents a scheme for structuring the active and passive ferry clips in a FCTS (see figure 2). Structuring the system in this manner considerably reduces the amount of work necessary to test different protocol implementations. It also reduces the effort required to use a different FTMP.

2.6.1 The Active Ferry Clip

The functions performed by the AFC are independent of the particular IUT being tested. Hence, changing the IUT has no effect on the AFC. However, since the AFC uses the services of an underlying FTMP, part of its code is FTMP specific. Nevertheless, a considerable portion of the AFC's functions such as fragmenting, reassembly and buffering of test data are operations that are independent of the FTMP.

To simplify the task of modifying the AFC when a different FTMP is used, it was decided to structure the AFC into two modules (see figure 2.1) as follows.

1. The Active Ferry Clip Finite State Machine Module (AFC FSM)

All functions of the AFC that are independent of the FTMP are incorporated into this module. The main function of this module is to implement the AFC's protocol state
machine (see appendix A) - hence its name. Fragmenting and reassembly of test data packets as well as buffering of test data to be sent to the PFC are also performed by this module.

The AFC FSM provides services to the TM module through a single interface via the FM-ASPs. It also uses the services of the Lower Mapping Module (LMAP) through another interface by means of the FT-ASPs.

2. The Active Ferry Clip Lower Mapping Module (AFC LMAP)

This module contains all the code that is dependent on the particular FTMP. Specifically, it maps the AFC’s ASPs (namely the FT-ASPs) into the ASPs or commands specific to the FTMP being used. The complexity of this mapping and hence the corresponding size of the AFC LMAP module depends on both the FTMP as well as the interface it provides.

By localizing the code specific to the FTMP in this module, it is possible to change the FTMP by simply rewriting this module. No change to any other part of the AFC is required.

A library of LMAP modules corresponding to different FTMPs can be set up. Thus, the problem of configuring an AFC to use a particular FTMP supported by the SUT is reduced to simply selecting an appropriate AFC LMAP module from the library.

2.6.2 The Passive Ferry Clip

The chief goal in designing a PFC is to keep it small and compact so that it may be possible to implement it in a SUT with memory limitations. The PFC’s code depends on both the IUT as well as the FTMP being used. Once again, our aim was to structure the PFC so
as to facilitate the easy replacement of both the FTMP and the IUT. Hence, it was decided to structure the PFC into three modules (see figure 2) as follows.

1. Passive Ferry Clip Finite State Machine Module (PFC FSM)

   This module contains all the functions of the PFC that are independent of the IUT and FTMP. It implements the PFC protocol state machine (see appendix A). Fragmentation, reassembly and the buffering of test data packets are also performed by this module.

   The PFC FSM provides services to the **IUT Interface Module** (described below) through a single interface via the FM-ASPs. It in turn uses the services of the PFC LMAP through another interface by means of the FT-ASPs.

2. Passive Ferry Clip Lower Mapping Module (PFC LMAP)

   This module contains the code for all the PFC functions that are specific to a particular FTMP, but independent of the IUT. It maps the PFC's ASPs (FT-ASPs) into the ASPs or commands specific to the FTMP being used. Hence, it is the only module in the PFC that needs to be modified when the FTMP is replaced by a new one.

3. IUT Interface Module

   This module contains all the code specific to the IUT. The PFC interfaces with the IUT through its upper and/or lower SAPs. Some data conversion is usually necessary to convert the data received by the PFC in a ferry PDU (see appendix A) into the format used by the IUT. This format is often implementation dependent and therefore unique to each IUT; hence this module needs to be rewritten or modified for each new IUT.

   The IUT interface module is subdivided into two parts - the Upper and Lower IUT Interface modules, which perform the conversions required by the upper and lower SAPs
of the IUT respectively.
Chapter 3

Implementation of the Ferry Clip based Test System

The previous chapter described a scheme for structuring the active and passive ferry clips in a FCTS. This chapter describes the use of this scheme to implement the AFC and PFC in the MPT, OSI-PTE and Unix environments. Each environment is described in detail, followed by a description of the implementation of the AFC and PFC in the environment. The discussion will focus on the MPT and OSI-PTE environments with a brief description of the Unix implementation. A detailed description of the implementation of the TM and E/D can be found in [23].

3.1 The Idacom MPT Implementation

The MPT368.2 [20] is a portable protocol tester (PT) manufactured by Idacom Electronics. It runs a proprietary operating system with a built in Forth interpreter and contains three Motorola 68000 CPUs, two of which are available for implementing the FCTS. The third CPU does not have access to the RS-232 interface. However, it does have access to the modem and printer ports and can be used to control these devices. Even though memory is partitioned
between the three CPUs, a CPU can access another CPU's memory partition. Communication between the CPUs is via inter-CPU messages. The operating system only allows one process per CPU and is event driven. Events are triggered by an incoming frame, a keyboard entry, a timer expiration or an inter-CPU message.

3.1.1 Overview of the Implementation

Our goal was to build a FCTS that would allow us to test protocols on a PT. Two PTs were used for this purpose - one for the test system and the other for the SUT. The two PTs were connected via an RS-232C serial cable. The TM, E/D and AFC were implemented as a single MPT process on one of the two available CPUs of the test system. The PFC and IUT were each implemented on a different CPU in the SUT. The PFC was separated from the IUT so as to minimize any interference it may cause on the operation of the IUT. The FTMP used was the X.25 packet layer protocol.

3.1.2 The Active Ferry Clip

The AFC was structured into two modules - the AFC FSM and AFC LMAP (see figure 2.1).

The Active Ferry Clip Finite State Machine Module

Fragmentation and reassembly of data packets is necessary since the FTMP places a limit on the maximum size of a packet it will accept. The AFC accepts data packets of arbitrary size (upto some maximum limit) from the TM and fragments it so that each fragment can fit into an FTMP packet. The packet fragment size chosen must be the lesser of the active and passive ferry clip's FTMP maximum packet size. The packet fragments are reassembled at the PFC. The PFC performs similar fragmentation of data packets received from the IUT before
transmitting them to the AFC.

The underlying FTMP could refuse to accept data from the AFC FSM once the FTMP’s local buffers fill up. Since the AFC FSM does not buffer the data sent to it by the TM, some mechanism is needed to inform the TM to retransmit when the FTMP cannot accept a packet because its buffers are full. The scheme used is described below.

When the TM sends a packet to the AFC FSM for transmission to the PFC, the AFC FSM extracts data from the packet and packs it into a FY-DATA PDU (see appendix A). It then sends the FY-DATA PDU to the AFC LMAP via a FT-DATAreq ASP for transmission to the PFC. If the FTMP accepts the packet for transmission to the PFC, the AFC LMAP returns a “packet accepted” message to the AFC FSM. The AFC FSM then extracts more data from the packet sent to it by the TM and repeats the process of sending it to the AFC LMAP until the whole data packet has been sent. Once the entire data packet is sent, the AFC FSM returns a “Ferry send completed” message to the TM. The TM can now attempt to send another packet if it so desires. If the FTMP does not accept the FY-DATA PDU due to the fact that its local buffers are full, the AFC LMAP returns a “packet not accepted” message to the AFC FSM. The AFC FSM in turn returns a “Ferry send blocked” message to the TM to inform it that the packet has not yet been sent. The process of calling the TM results in the underlying FTMP being run to clear out the FY-DATA PDUs thereby clearing the FTMP’s local buffers. The TM calls the AFC FSM repeatedly with a FM-DATAreq ASP until the AFC FSM returns a “Ferry send completed” message. In this manner the TM is kept in sync with the underlying FTMP.

The TM, E/D, AFC and FTMP were all implemented as a single MPT process, since they all resided on the same CPU in the test system. Hence, it was essential that the FTMP be
invoked periodically so that it could clear any waiting packets. To keep the structure of the
system uniform it was decided that the TM would invoke the AFC periodically with a special
service primitive, FM-FLUSH, so that the AFC could in turn invoke the FTMP to send out
any waiting data packets. Hence, the TM would not have to invoke the FTMP directly. If the
TM invoked the FTMP directly, part of its code would in turn be dependent on the FTMP,
thereby destroying the modularity of the system.

The AFC control protocol was implemented by building the AFC FSM as a simple finite
state machine with three states - Disconnected, Connecting and Connected (see Appendix A).
Entry to the AFC FSM was achieved through two interface points - the TM module (via the
E/D module) and the AFC LMAP. Depending on the current state of the AFC FSM and the
service primitive it received from the interface point, an action specified by the AFC control
protocol was performed.

The Active Ferry Clip Lower Mapping Module

The FTMP used in this implementation was the X.25 packet layer protocol. In the MPT
version of X.25, the user must specify the frames and packets to be generated. For example,
a "SABM" command would send across a data link layer "set asynchronous balanced mode"
frame, and a "CALL" command would send a packet layer "call request" packet to the peer
entity.

It is LMAP's responsibility to map the FT-ASP's (see table 3) into the MPT X.25 com-
mands and vice-versa. For example, an FT-CONNreq requires the AFC LMAP send a "set
asynchronous balanced mode" frame and wait for an "unnumbered acknowledgement" frame,
after which it must send a "call request" packet and wait for a "call confirm" packet before
returning a FT-CONNcnf to the AFC FSM module. This is complicated further by the fact that other frames and packets could be received by the AFC LMAP module during the process of setting up the ferry channel connection. These must be handled by the LMAP module.

With this sort of interface, the mapping between the Ferry ASPs and the FTMP commands is non-trivial. To best realize this mapping it was decided to implement the LMAP modules as a set finite state machine. An example of the FSM used to implement the PFC LMAP in the MPT environment can be found in figures 3.1 and 3.2.

An FTMP Independent Flow Control Scheme

The FTMP buffers test data packets until the AFC FSM or PFC FSM is ready to accept them. The PFC FSM also maintains a set of buffers to temporarily hold test data packets received from the AFC until the IUT accepts them. It is often difficult for the PFC FSM to use the same buffers that the FTMP uses as modification to the FTMP can be very complex and is sometimes not possible due to the unavailability of source code.

In the general case where a different set of buffers is used by the FTMP and PFC FSM, the flow control scheme used by the FTMP is insufficient in guaranteeing that the PFC FSM’s buffers will not overflow. To guarantee that the AFC will not swamp the PFC and overflow its FSM’s buffers a flow control scheme independent of that used by the FTMP is required. This second level of flow control operates between the AFC FSM and PFC FSM. Since this increases the functionality of the AFC and PFC, thereby increasing the size of their code, the flow control scheme should be as simple and compact as possible.

The scheme used was as follows. The “reserved bits” in the control field of a FY-CNTL PDU [4] (see appendix A) were used to implement two additional control functions, namely "flow
control on” (FT-FLOWon) and “flow control off” (FT-FLOWoff). When the PFC discovers its local buffers have filled past a certain “high-water mark” it sends a FT-FLOWon control message to the AFC. The AFC then ceases to send any further data to the PFC. However, it continues to accept data packets sent to it by the PFC, thereby allowing the PFC to clear its buffers. When the PFC’s local buffers clear below a “low-water mark”, it sends a FT-FLOWoff control message to the the AFC, informing it to resume sending data packets.

The “high-water mark” must be chosen carefully. This is because there is a time lag between the PFC discovering that the “high-water mark” has been reached and the FT-FLOWon control message reaching the AFC. During this time lag, data may still be sent to the PFC which must be ready to accept them. Hence, the “high-water mark” must be chosen so that there will still be enough space in the PFC’s local buffers to accept data until the FT-FLOWon control message gets to the AFC. The additional buffers required can usually be estimated quite easily as follows:

\[ N = \left\lfloor B \times \frac{E}{P} \right\rfloor \]

Generally, there is no problem in estimating “\(N\)”. Only in the case of long haul networks where the end-to-end delay has a high degree of variance does “\(E\)” become tricky to estimate. Overestimating “\(E\)” simply results in some buffer space being unused. However, underestimating it could result in the PFC’s local buffers overflowing.
CHAPTER 3. IMPLEMENTATION OF THE FERRY CLIP BASED TEST SYSTEM

It should be noted that in most cases simply allocating a large amount of buffers in the PFC FSM is adequate in insuring that the buffers will not be exhausted and hence the flow control scheme described need not be implemented. However, if the SUT has stringent memory limitations, allocating a large buffer pool in the PFC FSM may not be practical. In such a circumstance the simple flow control scheme described may be used. We note that the flow control mechanism constitutes only about 10% of the PFC code and about 5% of the AFC code.

3.1.3 The Passive Ferry Clip

The PFC was implemented on one of the two available CPUs in the SUT. The other available CPU was used to run the IUT. The structuring was done in this manner so that the PFC would not interfere with the operation of the IUT. The PFC was structured into three modules - PFC FSM, PFC LMAP and the IUT Interface module. The IUT Interface module resides on the same CPU as the IUT. Communication between the PFC FSM and the IUT Interface modules was achieved by means of inter-CPU messages.

The Passive Ferry Clip Finite State Machine Module

Three sets of buffers were maintained in the PFC FSM. The first set was to store data packets received from the AFC before they were processed. The other two sets of buffers were used to store packets to be sent to the AFC - one set for data packets and the other for control packets. The reason for separating the “data” and “control” buffers was so that higher priority could be given to packets in the control buffer. This would ensure that a FT-FLOWon control message could be sent promptly to the AFC without being delayed by waiting data packets.

The PFC control protocol was implemented by building the PFC FSM as a simple finite
CHAPTER 3. IMPLEMENTATION OF THE FERRY CLIP BASED TEST SYSTEM

state machine with two state - Disconnected and Connected (see Appendix A). Entry to the PFC FSM was achieved through two interface points - the IUT Interface module and the PFC LMAP. Depending on the current state of the PFC FSM and the service primitive it received from the interface point, an action specified by the PFC control protocol was performed.

The Passive Ferry Clip Lower Mapping Module

As explained above, the PFC LMAP was implemented as a FSM. The FSM to realize the PFC LMAP is considerably smaller and less complicated than its AFC LMAP counterpart. The FSM for the PFC LMAP can be found in figures 3.1 and 3.2. The FSM is divided into two parts for the purpose of readability. The first FSM (figure 3.1) represents the actions taken by the LMAP module when it receives FT-ASPs from the PFC FSM. The second FSM (figure 3.2) represents the actions taken on receipt of a frame or packet from the FTMP.

The IUT Interface Module

In the MPT environment, the IUT interface was placed together with the IUT on a single CPU. The IUT Interface module was divided into three parts: the PFC interface, an input handler and an output handler.

Simple message passing IPC is used to communicate between the PFC FSM and the IUT interface. The PFC FSM can send two types of messages to the IUT interface, namely initialize-IPC and incoming-packet.

When the PFC first attaches itself to the IUT, it sends it an initialize-IPC message. On receiving this message the PFC interface initializes the IPC structures required for subsequent communication. When the PFC has a packet for the IUT, it sends the buffer address of the
CHAPTER 3. IMPLEMENTATION OF THE FERRY CLIP BASED TEST SYSTEM

Notation:

\[ +A \lor +B / -C \] means on receipt of A or B transmit C

\[ \text{KEY:} \]

A: 
-FT-CONNrsp / -FT-DISCind
+FT-DATAreq / -FT-ERRORind
+FT-DISCreq / 
+* / -FT-ERRORind

B: 
+FT-CONNrsp / -FT-DISCind
+FT-DATAreq / -FT-ERRORind
+FT-DISCreq / 
+* / -FT-ERRORind

C: 
+FT-DATAreq / -DATApkt 
+* / -FT-ERRORind

D: 
+FT-DISCreq / -DISC

Passive Ferry Clip LMAP (Command State) FSM

Figure 3.1: The Passive Ferry Clip’s LMAP Command State FSM
CHAPTER 3. IMPLEMENTATION OF THE FERRY CLIP BASED TEST SYSTEM

Figure 3.2: The Passive Ferry Clip’s LMAP Frames/Packets FSM
packet along with the destination SAP identifier to the IUT Interface module via an incoming packet message. The PFC interface simply puts the packet into the IUT’s buffer, and frees the buffer allocated by the PFC by replying to the incoming message.

Whenever the IUT has output, the output handler is invoked. It simply allocates a common buffer and messages the PFC to handle the packet.

The entire IUT interface was implemented in about 250 lines of Forth code. Further details on the implementation of the IUT Interface module may be found in [23].

3.1.4 The IUT Tested

The X.25 packet layer protocol was used as the IUT in the MPT environment. Only the lower SAP of the IUT was accessible; yet the FCTS proved to be an effective tool in testing the IUT. Details of the test language, procedures used and results obtained may be found in [23].

3.2 The OSI-PTE Implementation

The Open Systems Interconnection - Protocol Testing Environment (OSI-PTE) [21] [22] is a sophisticated system recently developed at the University of British Columbia for the implementation and testing of communication protocols. In essence, it is the realization of the OSI reference model [9] within a single operating system process for the purpose of efficiency. Besides providing an operating environment which is close to the OSI reference model, the OSI-PTE also allows the incorporation of a TM into the system. The system supports all the test methods recommended by ISO, as well as passive monitoring, logging and even provides analysis capabilities.

Each protocol is structured as a single or a group of protocol entities. Each protocol entity provides the entity directly above it with certain services; it in turn uses the services of the
entity directly below it. In addition to protocol entities the OSI-PTE also uses "test entities". Test entities are user programs which utilize the protocol entities to control and observe the IUT. Test entities may be specified to span several protocol layers and access several protocol entities with ASPs through SAPs located at several different service boundaries simultaneously.

Communication between the entities is by means of an event-posting scheme, whereby a protocol entity posts an event to the protocol entity directly above or below it in the protocol hierarchy. An event is defined in terms of the event-id, event parameters and the service data unit. The event-id is an encoding of the service primitive for that protocol. The event parameters are the service parameters for the service primitive encoded in the event-id. The service data unit is the data that is being sent with the service primitive. As an example, consider the ISO Transport protocol connect request service primitive.

\[
\text{T-CONNECTreq ( src, dst, option, qos, tsdu )}
\]

where:

- \text{src: Calling Transport address.}
- \text{dst: Called Transport address.}
- \text{option: Transport connect options.}
- \text{qos: Transport quality of service parameter.}
- \text{tsdu: Transport service data unit.}

In this example the event-id is T-CONNECTreq, the event parameters are "src", "dst", "option" and "qos" and the service data unit is "tsdu".

The OSI-PTE, like the MPT's operating system, is event driven. At the heart of the system is a central dispatcher which oversees the operation of the system. The two important event types are the "message event" and a "timer expiry event". Each protocol entity can receive a
"message event" posted to either its upper or lower SAP. The message event would contain the event-id, event parameters and the service data unit for that event. A protocol entity may also receive notification on the expiry of a timer that it started. This notification is sent to it from the dispatcher.

The current version of the OSI-PTE was built on a Sun 3/50, running Sun/OS 3.2 (BSD Unix 4.2). However, since the OSI-PTE is operating system independent, it should be possible to port our OSI-PTE portion of the FCTS to most other systems. The "C" programming language was used to write both the OSI-PTE and the FCTS.

3.2.1 Overview of the Implementation

Our goal was to use the existing TM and AFC built-in the MPT environment to test protocols in the OSI-PTE environment. This represents the typical case where the SUT and test system differ in architecture. Hence, problems faced in this implementation would be typical of the types of problems that would face the designers of a FCTS.

To achieve this goal we needed to build a PFC in the OSI-PTE environment. It was decided to use the X.25 packet layer protocol as the FTMP, since this was the protocol used as the FTMP in the MPT environment. The ISO Transport Class 0 protocol [16] was chosen as the IUT.

3.2.2 The Passive Ferry Clip

The first problem faced was how to structure the PFC in the OSI-PTE environment. A normal ISO protocol has two well defined SAPs by which it interfaces to the protocol directly above and below it in the protocol hierarchy. The OSI-PTE was designed to implement protocols of this nature. The PFC is different in that it requires three access points, one to the
protocol being used as the FTMP and the other two to the two SAPs of the IUT. Hence, some restructuring was necessary to build a PFC in this environment. Using the scheme for structuring the PFC discussed in chapter 2, it was decided to structure the PFC as depicted in figure 3.3.
The PFC FSM was implemented as a single protocol entity. The IUT Interface was subdivided into the Lower and Upper IUT Interface modules, each implemented as a separate protocol entity. The system is now modular and fits well into the general structure of the OSI-PTE.

The Passive Ferry Clip Finite State Machine Module

Functionally, the PFC FSM is very similar to that implemented in the MPT environment. Received packet fragments are unpacked from the ferry PDU format and sent via the event posting scheme to the Lower or Upper IUT Interface. Similarly, packets received from the Upper or Lower IUT Interface entities are fragmented and sent to the AFC.

The Passive Ferry Clip Lower Mapping Module

The FTMP used was the X.25 packet layer protocol which provides the user with an X.213 interface [11] [12]. The user may use this protocol by invoking it with the X.213 service primitives. With this interface the mapping between the ferry FT-ASPs and the X.213 ASPs is one-to-one (see table 3.1). Hence the LMAP module is trivial. The code for the LMAP module can be found in appendix B.

The IUT Interface Module

The IUT Interface module was subdivided into two entities - the Upper and Lower IUT Interfaces. The Upper IUT Interface converted data received from the AFC in an FY-DATA PDU into a form that could be accepted by the upper SAP of the IUT. It also converted data sent by the IUT to its upper SAP into a format that could be sent in a FY-DATA PDU and that
X.213 ASP & FT-ASP

<table>
<thead>
<tr>
<th>X.213 ASP</th>
<th>FT-ASP</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-CONNECTindication</td>
<td>FT-CONNind</td>
</tr>
<tr>
<td>N-CONNECTresponse</td>
<td>FT-CONNrsp</td>
</tr>
<tr>
<td>N-DISCONNECTrequest</td>
<td>FT-DISCreq</td>
</tr>
<tr>
<td>N-DISCONNECTindication</td>
<td>FT-DISCind</td>
</tr>
<tr>
<td>N-DATArequest</td>
<td>FT-DATAreq</td>
</tr>
<tr>
<td>N-DATAindication</td>
<td>FT-DATAind</td>
</tr>
<tr>
<td>All other ASPs</td>
<td>FT-ERRORind</td>
</tr>
</tbody>
</table>

Table 3.1: Mapping between FT-ASPs and X.213 ASPs

the E/D in the test system could understand. The Lower IUT Interface performed a similar set of mappings for the lower SAP of the IUT.

The IUT expects data in the form of a message event. Hence, it is necessary to convert the bit stream received in a FY-DATA PDU into an event-id, event parameters and a service data unit. It was decided that the E/D module on the MPT make three FD-DATAreq calls to the AFC FSM for each IUT event. The first call would contain the event-id, the second a linear encoding of the event parameters and the third call would contain the service data unit. Each call results in separate a FY-DATA PDUs being sent to the PFC. The reason for separating the ASP into three calls is for generality and reasons dictated by the OSI-PTE environment. Not all IUT events require event parameters and service data units. To handle this the first byte of the second FD-DATAreq was used to indicate the absence or presence of the event parameters. If this byte was set to “1” the IUT Interface could expect the event parameters to follow. Similarly, the second byte was used to indicate the absence or presence of a service data unit. This scheme is sufficiently general so that it can be used for any protocol in the OSI-PTE environment.
3.2.3 The IUT Tested

The IUT tested by this implementation of the FCTS was the ISO Transport Class 0 protocol. Both SAPs of the protocol were accessible and so a thorough diagnostic testing could be performed. In fact, the FCTS proved to be quite an effective tool in detecting problems with the IUT and aiding in their solution. A detailed description of the diagnostic testing of this IUT can be found in [23].

3.3 The Unix Implementation

Unix is a powerful and widely used operating system. The Unix environment used to develop the FCTS consisted of a pair of SUN 3/50 workstations connected via an ethernet local area network. Each workstation ran a version of Unix BSD 4.2 (Sun/OS 3.2).

The FCTS was implemented as a set of three processes. The TM, E/D and AFC were structured into a single process running on one workstation. The PFC and IUT were implemented as two separate processes running on the other workstation.

Communication between the ferry clips was achieved using Unix stream sockets in the internetwork domain (TCP/IP) [6]. This particular type of socket was chosen since it best fit the FTMP requirements (see chapter 2). The protocol tested using this FCTS implementation was the X.25 packet layer protocol [23].

The main challenge faced in this implementation was devicing a scheme to prevent a deadlock occurring between the AFC and PFC processes. When a process writes to a socket, the "write operation" will block if the socket's internal buffers are full. The socket's buffers are cleared when the process at the other end reads data off the socket. If both the AFC and PFC processes send data to an FTMP socket whose buffers are full, both will block and wait for the other to
clear the socket's buffers. This problem may be solved by making the write operation performed by the AFC non-blocking. Hence, the AFC could always clear a blocked PFC's buffers, thereby preventing a deadlock from occurring.

3.4 Comparison of the Lower Mapping Modules

The size of the LMAP module depends not only on the protocol used as the FTMP, but also on the type of interface the protocol provides. In our implementations, the X.25 packet layer protocol was used as the FTMP in both the MPT and OSI-PTE implementations. Appendix B contains the sources for the PFC LMAP in both these environments. The MPT version of the PFC LMAP constitutes a large portion of the overall size of the PFC. It is considerably more complicated than its OSI-PTE counterpart and required more time to write and test. In contrast the OSI-PTE version of the PFC LMAP constitutes only a small portion of the overall size of the PFC and was trivial to write and test. This difference was due to the different user interfaces provided by the two different protocol implementations.

The LMAP module should not need to be changed if a different implementation of the same protocol is used as the FTMP on the other system. In our case, the same AFC LMAP was used to interface with the PFC LMAP in both the MPT and OSI-PTE environments.

3.5 Size of Modules

The MPT implementation of the AFC and PFC was coded in Forth. The OSI-PTE implementation of the PFC was coded in C. The following tables summarize the size of the various FCTS modules in terms of number of lines of code excluding comments.
CHAPTER 3. IMPLEMENTATION OF THE FERRY CLIP BASED TEST SYSTEM

<table>
<thead>
<tr>
<th>Module</th>
<th>Lines of Forth code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFC FSM</td>
<td>580</td>
</tr>
<tr>
<td>AFC LMAP</td>
<td>540</td>
</tr>
<tr>
<td>Support Routines</td>
<td>420</td>
</tr>
</tbody>
</table>

Table 3.2: Active Ferry Clip Module Sizes in MPT Environment

<table>
<thead>
<tr>
<th>Module</th>
<th>Lines of Forth code</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFC FSM</td>
<td>600</td>
</tr>
<tr>
<td>PFC LMAP</td>
<td>300</td>
</tr>
<tr>
<td>IUT Interface Module</td>
<td>250</td>
</tr>
<tr>
<td>Support Routines</td>
<td>500</td>
</tr>
</tbody>
</table>

Table 3.3: Passive Ferry Clip Module Sizes in MPT Environment

<table>
<thead>
<tr>
<th>Module</th>
<th>Lines of C code</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFC FSM</td>
<td>540</td>
</tr>
<tr>
<td>PFC LMAP</td>
<td>190</td>
</tr>
<tr>
<td>IUT Interface Module</td>
<td>1250</td>
</tr>
</tbody>
</table>

Table 3.4: Passive Ferry Clip Module Sizes in OSI-PTE Environment
3.6 Loop-back Testing and Testing of a Null IUT

The FCP has a facility called "loop-back testing" to aid in the testing of the PFC FSM and the test channel. The AFC puts the PFC into loop-back mode by sending it a FY-CNTL PDU with the L-bit set (see appendix A). While in loop-back mode the PFC returns any data it receives from the AFC to the AFC after toggling the upper/lower (U/L) bit in the FY-DATA PDU. The data is not sent to the IUT. Also, while the PFC is in loop-back mode any data received by it from the IUT is discarded. In this manner the AFC can check the FY-DATA PDU it sends against what it receives to ensure that the PFC is functioning properly and that data is not being altered by the FTMP. The PFC is put back into normal mode when it receives a FY-CNTL PDU from the AFC with the L-bit turned off.

Loop-back testing is a useful feature that allows testing of a new PFC FSM or FTMP. However, since no data is sent to the IUT it cannot test if the IUT Interface module is functioning correctly. It is very important to test the IUT Interface module before actually using the FCTS. To do this we used a "Null IUT". The Null IUT is a trivial protocol which simply returns data it receives without altering it in any way. Data received by the Null IUT on its lower SAP is returned via its upper SAP and vice-versa. Hence, the AFC can check if the IUT Interface module is functioning correctly. On bringing up a new system, we first performed loop-back testing to verify that the PFC FSM was functioning properly. Once this was done, we added a Null IUT to the system to test the IUT Interface module. If data was altered in any way, we would know immediately that the problem was caused by a faulty IUT Interface module.
Chapter 4

Ferry Clip Applications

The previous chapter described the implementation of a FCTS. This chapter illustrates the versatility of the ferry clip concept by describing some of its applications. The manner in which the ISO Abstract Test Methods (see chapter 1) can be realized using a FCTS is described. Also described are extensions to the ferry clip concept to perform Multi-layer and Multi-party testing.

4.1 Realization of the ISO Abstract Test Methods

A FCTS can be used to realize the Local, Distributed and Remote abstract test methods. There are several advantages in using a FCTS to do this. The LT and UT no longer need to reside in the SUT. Since both the LT and UT now reside in the test system, synchronization between them is simplified.

4.1.1 Realization of the Local Test Method

The LTM is the most powerful of all the test methods. Since it has access to both the upper and lower service boundaries of the IUT it is possible to use it to exercise those state transitions of the IUT that would be difficult or impossible to reach using the other test methods. The
main disadvantages of the conventional LTM, which often makes its use impractical, are that both the LT and UT need to reside in the SUT and the need to have direct access to both the upper and lower service boundaries of the IUT.

Figure 4.1 shows how the ferry clip approach can be used to realize the LTM. The PFC is the only part of the FCTS that must reside in the SUT; the LT and UT now both reside in the test system. The PFC is small and simple to write. Due to its small size it can even be implemented in systems with stringent memory limitations. Hence, the ferry clip approach makes the LTM feasible. Furthermore commercial protocol testers which offer many facilities
already built in, such as data logging and analysis capabilities can be used as the external tester.

We have used the ferry clip based LTM quite successfully to test the X.25 packet layer protocol and the ISO Transport Class 0 protocol. For further details see [23].

4.1.2 Realization of the Distributed Test Method

In the DTM the lower service interface of the IUT is accessed indirectly via the protocol layers below the IUT. Using a FCTS to realize the DTM, the upper service interface of the IUT can be accessed via the ferry clips as shown in figure 4.2.

Figure 4.2: The Ferry Clip Distributed Test Approach
The main problem with using the conventional DTM is that synchronization procedures are required between the LT in the test system and the remote UT in the SUT. Using a FCTS to realize the DTM simplifies this problem since both the LT and UT now reside on the same system. In fact the LT and UT may now be combined into a single tester [23] thereby eliminating the synchronization problem altogether.

### 4.1.3 Realization of the Remote Test Method

In the RTM no access is provided to the upper service interface of the IUT. In the conventional RTM, the lower service interface of the IUT is accessed indirectly via the protocol layers directly below the IUT. When a FCTS is used to realize the RTM, test data is sent directly to the lower service interface of the IUT via the two ferry clips (see figure 4.3).

Since the lower service interface of the IUT is accessed directly, certain test cases (e.g. sending the IUT incorrect data via its lower service interface) which cannot be handled by the conventional RTM can now be performed. Hence, the FCTS version of the RTM is considerably more powerful than its conventional counterpart.

It has been suggested that the CTM can be used as an alternative to the ferry clip approach. The disadvantage of doing so is that the CTM requires a specific TMPs be written for each layer to be tested. Hence, a considerable amount of tester software needs to be modified for testing different layers.

### 4.2 Multi-layer Testing using a Ferry Clip based Test System

Multi-layer testing involves testing “a set of any number of adjacent layers of the SUT” [8] in combination with access only to the lower and/or upper service boundaries of the protocol stack. Little additional work needs to be done to use a FCTS to test a Multi-layer IUT.
CHAPTER 4. FERRY CLIP APPLICATIONS

Figure 4.3: The Ferry Clip Remote Test Approach
Replacing the term "IUT" in figures 4.1, 4.2 and 4.3 with the term "Multi-layer IUT" depicts how a FCTS can be used to perform Multi-layer testing. It is important to note that only the upper service interface of the highest layer and the lower service interface of the lowest layer in the Multi-layer IUT are accessed by these methods.

At times it is convenient to observe and control the data exchanges at some or all the service boundaries within a Multi-layer IUT. It is important to note that this form of testing a Multi-layer IUT is no longer ISO Multi-layer testing due to the fact that service boundaries other than the upper and lower ones are being accessed. It is possible to adapt a FCTS to do this. We refer to this form of testing a Multi-layer IUT as "Extended Multi-layer testing".

To observe the data exchanges at the boundary between two layers requires the data that is normally sent between two adjacent layers is also trapped by the PFC and sent to the remote TM. Figure 4.4 illustrates how a FCTS may be used to perform Extended Multi-layer testing. Only the PFC needs to be modified. The existing FCP already has a provision (the LI-bits in an FY-DATA PDU) to send data to and receive data from several protocol layers in the SUT simultaneously (see appendix A). However, if the PFC clips on to the upper and lower service interfaces of the individual layers within a Multi-layer IUT, the links between the individual layers will now be broken. This means that data sent by a (N)-entity to its upper service interface, normally destined for (N+1)-entity, is now sent to the PFC instead. Similarly, data sent by a (N)-entity to its lower service interface, normally destined for (N-1)-entity, is also rerouted to the PFC. To rectify this the PFC control protocol needs to be modified slightly as follows.

$U_N$: Data sent to upper service interface of layer N and received by the PFC FSM.
$L_N$: Data sent to lower service interface of layer N and received by the PFC FSM.
$h$: Highest layer of Multi-layer IUT.
Figure 4.4: Extended Multi-layer Testing using a Ferry Clip based Test System
CHAPTER 4.  FERRY CLIP APPLICATIONS

$l$: Lowest layer of Multi-layer IUT.

case(i): $(U_N \text{ AND } N = h) \text{ OR } (L_N \text{ AND } N = l)$

Response:
Forward data to AFC.

case(ii): $(U_N \text{ AND } N \neq h)$

Response:
(a) Forward data to AFC, and
(b) Send data to lower service interface of layer $(N + 1)$.

case(iii): $(L_N \text{ AND } N \neq l)$

Response:
(a) Forward data to AFC, and
(b) Send data to upper service interface of layer $(N - 1)$.

The PFC FSM can easily be modified to incorporate this change in the PFC control protocol. In this fashion data sent between two adjacent layers of a Multi-layer IUT can also be trapped and sent to the TM on the remote system. The TM can also send data to any of the service interfaces of the Multi-layer IUT. Hence, for example it could send incorrect data to a particular service interface and observe the responses produces at the other service interfaces of the Multi-layer IUT.
4.3 Multi-party Testing using a Ferry Clip based Test System

Multi-party test methods "involve one or more LTs cooperating with one or more UTs" [15] via a number of connections through a IUT.

![Diagram of Multi-party Testing](image)

Figure 4.5: A Local Single Multi-party Test Method

Figure 4.5 illustrates how the LTM can be adapted to perform Multi-party testing of a single or multi-layer IUT. Traditionally one of the LTs is designated as the "master" LT. It initiates a test case and collects observations from the other LTs and UTs so that it may analyze the
observations and assign a verdict. "There is a one-to-one correspondence between each LT and UT, making pairs of testers surrounding the IUT" [15]. Some form of synchronization is required between the LTs. Also required is an ability for the LTs to communicate synchronously and asynchronously amongst themselves. Furthermore, coordination procedures are also required between the LTs and UTs. This coordination is more difficult to achieve in the conventional distributed Multi-party test method than with the conventional local Multi-party method due to the fact that in the distributed Multi-party method the UTs and LTs reside on different systems.

There are numerous instances where Multi-party testing of an IUT is necessary. For example, to test the ability of a MHS entity [13] to create a second copy of a message so that it may be relayed to two different domains requires the use of a Multi-party test method with three LTs. One of the LTs is the originator of the message and the other two are the recipients.

Most protocols that support multiple connections do so by using a field in the data sent or received by them to specify the connection identifier. In this case the existing FCTS can be used without modification to do Multi-party testing. However, in the case that the protocol provides different physical connections (e.g. Unix socket connections) for sending and receiving data, a small modification is required to use the FCTS to perform Multi-party testing as illustrated in figure 4.6. Each individual connection is handled by an upper and lower IUT Interface module pair. Test data sent to the PFC FSM now requires some indication of what connection (IUT Interface) the data is to be sent to. Similarly, data received by the PFC FSM from a particular connection of the IUT requires that an indication of the data connection be sent to the AFC along with the test data. To handle this we propose a minor change in the format of a FY-DATA PDU.
CHAPTER 4. FERRY CLIP APPLICATIONS

Figure 4.6: A Ferry Clip based Local Single Multi-party Test Method
CHAPTER 4. FERRY CLIP APPLICATIONS

<table>
<thead>
<tr>
<th>Header Field</th>
<th>Connection Field</th>
<th>Length Field</th>
<th>Data Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>1 Byte</td>
<td>2 Bytes</td>
<td></td>
</tr>
</tbody>
</table>

Connection Id = 8 Bits = 256 possible connections

Figure 4.7: A Modified FY-DATA PDU for Multi-party Testing

Figure 4.7 illustrates the new proposed format of the FY-DATA PDU. The only change is the addition of a "connection byte" which specifies the connection the test data is being sent to or received from.

In reality a separate IUT Interface pair is not required for each connection through the IUT. A single upper and lower IUT Interface would suffice. The IUT interface would now be responsible for the mapping between a connection-id (indicated by the connection byte) and an actual connection through the IUT and vice-versa.

There are several advantages to using a FCTS to realize a Multi-party test method. The LTs and UTs can all be moved to the test system, thereby simplifying the coordination procedures between the LTs and their corresponding UTs. Furthermore since all the testers now reside on the same system, it may be possible to implement them as a single tester, thereby eliminating
the synchronization problem altogether. The space requirements of placing several LTs and UTs in the SUT would often render the Local Multi-party test method infeasible, were it not implemented using a FCTS. Even the Distributed Multi-party test method requires several UTs to be placed in the SUT. This too may be infeasible if not realized using a FCTS. Finally, the tester software (LT and UT) in a FCTS completely resides on the test system unlike the conventional test method which requires part of the tester software to reside in the SUT. Hence, in a FCTS the tester software need only be written once for the test system instead of having to be rewritten for each new SUT. This is probably the greatest advantage of using a FCTS to implement a Multi-party test method, considering the potential size and complexity of the tester software required to perform Multi-party testing.
Chapter 5

Some Interesting Ferry Clip based Issues

The previous chapter described some applications of a FCTS. This chapter deals with some interesting issues that arise from using a FCTS. These issues can be broadly divided into two classes - conceptual and implementation based. The conceptual issues discussed are the "ordering" and "time-lag" problems. Chapter 3 described some implementation based problems encountered while building a FCTS. This chapter addresses two other implementation based issues, namely the need to use ASN.1 [14] and the requirement that the IUT have exposed interfaces. Some other miscellaneous issues are also discussed.

5.1 The Ordering Problem

When a protocol entity needs to communicate with the entity directly above it in the protocol stack, it invokes an ASP at its upper SAP. This ASP is received by the layer directly above the sending entity via its lower SAP. Similarly, when a protocol entity needs the services of the entity directly below it, it invokes an ASP at its lower SAP which in turn is received by the entity below it via its upper SAP.
In the case of an event driven system, when an entity invokes an ASP at its upper or lower SAP, an event is automatically generated by the system for the recipient of that ASP (see OSI-PTE in chapter 3). Normally, protocol entities are combined into a single operating system process (e.g. the OSI-PTE environment). At most each protocol entity may be implemented as a single process. In both of these cases, events can never be generated simultaneously at both SAPs of a protocol entity. No ambiguity exists here since even if an ASP is invoked almost simultaneously at each SAP of the entity, the system will generate events for the respective recipient entities in the order in which they were generated by the sending entity. When a FCTS is used to realize the LTM, the PFC will be the recipient of the ASPs from both SAPs of the IUT. In an event driven system, the PFC is guaranteed to receive the ASPs in the order in which they were sent by the IUT. Since neither the PFC nor the AFC reorder the ASPs, the tester in turn receives the ASPs in the order in which the IUT generated them.

In a non-event based system (e.g. the Unix environment), the PFC often needs to poll both SAPs of the IUT to check if an ASP has been invoked at a particular SAP. A potential problem could occur with polling the SAPs. If the IUT generates an ASP almost simultaneously for both its upper and lower SAPs, the order of polling the SAPs could result in the second ASP invoked by the IUT being detected first by the PFC. This phenomenon is referred to as the "ordering problem". A number of protocols were studied to see if ASPs were ever invoked simultaneously at both SAPs of a protocol and if so whether it could cause a problem.

The protocols studied were X.25 (data link and packet layer protocols)[10], the ISO Transport protocols (TP0 to TP4) [16] and FTAM [17]. It was observed that in the few cases that a protocol generated an ASP (or PDU) almost simultaneously at both its upper and lower SAPs, the protocol standard did not specify the order in which the ASPs (or PDUs) were to be sent.
by the protocol. Hence, no harm was done if the PFC received the second ASP (or PDU) sent by the IUT before the first one.

An example of such behavior can be seen in the disconnect phase of the ISO Transport protocol. When the Transport protocol receives a “DR” (disconnect request) PDU from its lower SAP it responds by sending a “DC” (disconnect confirm) PDU to its lower SAP and a T_DISCind ASP to its upper SAP. The ISO standard does not specify which event should be performed first and hence no assumption can be made about the relative order of these events in a test script.

<table>
<thead>
<tr>
<th>Event</th>
<th>Verdict</th>
</tr>
</thead>
<tbody>
<tr>
<td>L! DR</td>
<td></td>
</tr>
<tr>
<td>L? DC</td>
<td></td>
</tr>
<tr>
<td>U? T_DISCind</td>
<td>PASS</td>
</tr>
<tr>
<td>Default</td>
<td>FAIL</td>
</tr>
</tbody>
</table>

Table 5.1: Incorrect Local Test Script Fragment for Transport Disconnect Phase

The problem with this test script fragment is that it assumes that a “DC” PDU will be received at the lower SAP before the “T_DISCind” ASP is received at the upper SAP. Since the protocol standard specifies no such ordering, different implementations of this protocol may generate these events in different order. Hence, with the above test script it is possible that a “conforming” IUT is given a FAIL verdict.

A correct test script fragment to test the disconnect phase of the Transport Protocol would
CHAPTER 5. SOME INTERESTING FERRY CLIP BASED ISSUES

<table>
<thead>
<tr>
<th>Event</th>
<th>Verdict</th>
</tr>
</thead>
<tbody>
<tr>
<td>L! DR</td>
<td></td>
</tr>
<tr>
<td>U? T_DISCind</td>
<td></td>
</tr>
<tr>
<td>L? DC</td>
<td>PASS</td>
</tr>
<tr>
<td>Default</td>
<td>FAIL</td>
</tr>
<tr>
<td>L? DC</td>
<td></td>
</tr>
<tr>
<td>U? T_DISCind</td>
<td>PASS</td>
</tr>
<tr>
<td>Default</td>
<td>FAIL</td>
</tr>
<tr>
<td>Default</td>
<td>FAIL</td>
</tr>
</tbody>
</table>

Table 5.2: Correct Local Test Script Fragment for Transport Disconnect Phase

be as provided in table 5.1. In this test script no order is implied of the ASPs (or PDUs) to be received at the upper and lower SAPs.

A protocol that generates ASPs (or PDUs) almost simultaneously at both its upper and lower SAPs may require one of the ASPs (or PDUs) be received first by the recipient entity. To achieve this, the protocol must send the first ASP (or PDU) and then wait for an acknowledgement from the recipient entity. On receipt of this acknowledgement, the protocol may then send the second ASP (or PDU). Our study of a number of ISO protocols showed that this mechanism was always employed whenever the order of packets to be received is significant. In this case, since the ASPs (or PDUs) are no longer sent simultaneously, the ordering problem does not arise. In the case when order of receipt of ASPs (or PDUs) does not matter, the ISO document does not specify which ASP (or PDU) is to be sent first. Hence, no problem results if the order of polling the SAPs results in the second ASP (or PDU) sent by the protocol being received first by the PFC.
CHAPTER 5. SOME INTERESTING FERRY CLIP BASED ISSUES

5.2 The Time-lag Problem

In a FCTS, the tester and IUT reside on different computer systems. Hence, there is a time lag between the IUT generating an ASP (or PDU) at one of its SAPs and the corresponding ASP (or PDU) being received by the tester. Correspondingly, there is a time lag between the tester sending test data and the IUT receiving it.

The tester assuming the IUT to be in a certain state could send it some test data and in doing so expect a certain reaction from the IUT based on the test data it has sent. In the meantime the IUT not yet having received this test data from the tester may also send the tester some data, based on the state it thinks the tester is in. It too may expect a response from the tester based on the data it has just sent. Each system now possesses incorrect state information about the other system. Our concern was whether this apparent inconsistency of states could cause the FCTS to give an incorrect verdict. This problem is referred to as the "time-lag problem".

It is important to note that the time-lag problem is not unique to test systems based on the ferry clip concept. It can occur in conventional test systems too. The probability of it occurring in the ferry clip version of the LTM is higher due to the fact that both the LT and UT are on a physically different system from the IUT, thereby stretching the time-lag even further. Furthermore, the problem is even more likely to occur when the two systems are connected by a long-haul network, where there may be long time delays associated with data traveling between the two systems.

No time-lag based problems were detected in our implementations. Hence, once again X.25, the Transport protocols and FTAM were studied to see if the time-lag problem was possible and whether it could cause a problem.
CHAPTER 5. SOME INTERESTING FERRY CLIP BASED ISSUES

From the protocols studied it was observed that a protocol entity normally responds only when data is sent to it. Hence, in normal operation the time-lag problem will not occur since the protocol being tested will only send data after it has received data from the tester. As long as the test script is correctly written and the IUT conforms, the tester will know exactly what it should receive from the IUT. Any unexpected data received by the tester from the IUT would result in a "FAIL verdict".

Some unusual behavior is possible when a protocol that uses timers has its timeout value set too small. The solution is to increase the value of the timeout variable in the protocol. Care must be taken in writing a test script to handle such a situation as illustrated by the following example.

Consider a FCTS being used to test the LAPB protocol's link connection and disconnection capability. Furthermore, consider the IUT to be set up as a DCE. The IUT initiates the connection by sending a SABM frame with the poll-bit set and expecting a UA frame with the final-bit set. At the same time it starts timer T1. If the timeout value of timer T1 is too short and expires before the UA frame arrives from the tester, the IUT will resend a SABM frame with the poll bit set and restart timer T1.

The tester will receive two SABM frames instead of just the one it expects. Table 5.2 contains an incorrect test script fragment to test LAPB link setup and disconnection. When the tester receives the second SABM frame instead of the DISC frame it expects it concludes the IUT is non-conforming and passes a "FAIL verdict". Hence, as can be seen from this example, if the test script writer is not careful to account for the possibility of a timeout, a "fail verdict" could be passed for a conforming IUT.

No situation was observed where the time-lag problem would cause incorrect operation of
CHAPTER 5. SOME INTERESTING FERRY CLIP BASED ISSUES

<table>
<thead>
<tr>
<th>Event</th>
<th>Verdict</th>
</tr>
</thead>
<tbody>
<tr>
<td>U! Set_Up_Link</td>
<td></td>
</tr>
<tr>
<td>L? SABM(Poll bit set)</td>
<td></td>
</tr>
<tr>
<td>L! UA(Final bit set)</td>
<td></td>
</tr>
<tr>
<td>U! Disconnect_Link</td>
<td></td>
</tr>
<tr>
<td>L? DISC</td>
<td></td>
</tr>
<tr>
<td>L! UA</td>
<td>PASS</td>
</tr>
<tr>
<td>Default</td>
<td>FAIL</td>
</tr>
<tr>
<td>Default</td>
<td>FAIL</td>
</tr>
</tbody>
</table>

Table 5.3: Incorrect Local Test Script Fragment for LAPB Connection/Disconnection Phase

the FCTS. Hence, from our case study, we conclude that the time-lag issue is not generally a problem.

5.3 The Need for using ASN.1

In all three environments in which our FCTS was built, the test system and the SUT had
the same CPU architecture, namely all were Motorola 68000 based processors. Hence, the basic
data types had the same representation on both the test system and the SUT. A more general
situation would be one in which the architecture of the test system and the SUT differ, resulting
in incompatible basic data types between the two machines. For example on one machine (e.g.
IBM-PC) the representation for the basic data type “integer” may be 16 bits and on another
(e.g. Sun 3/50) it may be 32 bits.

In such a heterogeneous environment, it is the E/D module’s job to convert the test data
to be sent to the SUT into the data representation that is accepted by the SUT. Similarly,
the E/D module is also responsible for mapping the data received from the SUT into the data.
CHAPTER 5. SOME INTERESTING FERRY CLIP BASED ISSUES

representation accepted by the test system.

Bochmann [19] suggested using a standard data representation technique, such as ASN.1. Using ASN.1 has the advantage that a large portion of the E/D module that was previously involved with mapping between incompatible data representations can be written once in ASN.1 with little or no change required to it when testing different implementations of the same protocol on different SUTs.

However, this assumes that an ASN.1 compiler is available on both the test system as well as the SUT. In practice this may not be the case. We were unable to use ASN.1 in our system due to the fact that although an ASN.1 compiler was available for the SUT, none was available for our test system.

5.4 The Need for Exposed Interfaces

One of the main objections from opponents of the ferry clip approach is that the use of the ferry clip method requires the IUT expose one or both of its service interfaces. Recent trends have shown that vendors do not wish to expose the service interfaces of their products. Furthermore, several layers are usually implemented together as a single product, with no exposed interfaces in between the layers.

However, the need for exposed interfaces is not unique to the ferry clip method. The conventional LTM requires direct access to both service boundaries of an IUT. The conventional DTM requires direct access to the upper service boundary of an IUT. With the coordinated test method, some interface is required to connect the UT with the IUT. Hence, all in all, the ferry clip method is no better nor worse than the conventional test methods in its requirement for exposed interfaces.
CHAPTER 5. SOME INTERESTING FERRY CLIP BASED ISSUES

The choice of the appropriate test method depends to a large extent on which service boundaries of the IUT are accessible. If both service boundaries are exposed, the ferry clip version of the LTM is the preferred approach. If the lower service boundary is exposed, the ferry clip version of the RTM can be used. Similarly, if only the upper service boundary of the IUT is exposed the ferry clip version of the DTM may be used. In the case where several protocol layers are implemented together with no exposed interfaces between the intermediate layers, the Multi-layer version of the ferry clip approach can be used. For higher layer IUTs requiring Multi-party testing (e.g. MHS) the Multi-party variant of the ferry clip method can be used. In everyone of these approaches the ferry clip realization of the conventional test methods have several advantages over their conventional counterparts.

5.5 The Distributed Test Method Logging Problem

Recently a serious difficulty was discovered with the requirement to produce a conformance log for the conventional DTM. “It was realized that with the [conventional] DTM it was not possible to avoid placing logging requirements on the SUT. Furthermore, in the [conventional] DTM, the upper boundary may be a human interface, leaving the responsibility for recording the conformance log unclear” [18].

When a FCTS is used to realize the DTM, the UT is placed in the test system. Since both LT and UT reside on the same system, the logging problem does not arise. Logging data exchanges is a simple matter in a FCTS, requiring the TM log any data received from or sent to the AFC FSM via the E/D module. For further details of how logging is performed in a FCTS refer to [23].
Chapter 6

Conclusions

This chapter summarizes the work done and draws some conclusions about the ferry clip test approach. Areas of future research are also mentioned.

6.1 Summary of Work Done

The ferry clip approach to protocol testing was recently introduced as an attempt to overcome some problems associated with the conventional test methods. The ferry clip method allows both the LT and UT to reside on the test system, thereby reducing the amount of software that must be implemented in the SUT as well as enhancing the synchronization between the LT and UT.

A model is provided for structuring a FCTS so as to reduce the amount of work necessary to modify the test software to test different protocol implementations and/or use a different underlying protocol as the FTMP. The model is at a sufficiently high level so that it can be implemented in almost any environment.

Three implementations of the FCTS were built in three different environments. The thesis has focused on the implementations in the MPT and OSI-PTE environments. The model for structuring the FCTS was used consistently in all the environments. Interesting design
CHAPTER 6. CONCLUSIONS

and implementation issues were encountered. The need for a FTMP independent flow control scheme was pointed out and implemented. The nature of the OSI-PTE environment required refining our basic model to split the IUT Interface module into a Lower and Upper IUT Interface. Using the FCTS it was possible to effectively test the X.25 packet layer protocol in the MPT environment and the ISO Transport Class 0 protocol in the OSI-PTE environment. The FCTS proved to be an extremely effective tool for diagnostic testing and was used to correct certain problems in our test implementation of the Transport protocol.

A FCTS can be used to realize the abstract LTM, DTM and RTM. Furthermore, it can also be adapted to perform Multi-layer testing. A scheme to perform Extended Multi-layer testing was described. Some high level protocols such as MHS require Multi-party test approaches. A scheme for extending the FCTS to perform Multi-party testing was also presented. The versatility of the ferry clip approach was illustrated by how easily it can be adapted to realize all these different test approaches.

Some interesting issues associated with the use of the ferry clip concept were also discussed. We believe that the “ordering” and “time-lag” problems will not cause incorrect operation of the FCTS if care is taken when writing local test scripts for the FCTS. ASN.1 can aid in reducing the encoding/decoding effort when the test system and SUT have a different architecture. However, its use requires that an ASN.1 compiler be available on both the test system and the SUT. It was also pointed out that the logging problem that affects the conventional DTM will not occur when a FCTS is used to realize the DTM.
6.2 Future Research

Our implementations have shown that the FCTS is an extremely powerful and effective tool to perform diagnostic testing. Furthermore, the structuring scheme suggested proved to be extremely useful in all the environments in which the FCTS was built. Like any project, however, some interesting areas of research remain and are discussed below.

• ISO has recently put emphasis on Multi-party test approaches. Currently, however, little work has been done in the area. The ferry clip approach provides a useful means for performing Multi-party testing and has several advantages over the conventional approach. A FCTS implemented to perform Multi-party testing would not only prove interesting study but also provide some useful and much needed input to ISO. A lot more work needs to be done in designing the TM (LT and UT) to perform Multi-party testing.

• The need for using ASN.1 in a heterogeneous environment was pointed out. The effort required to use ASN.1 is an issue not addressed by this thesis since an ASN.1 compiler was not available on our test system. A comparison of the current approach of encoding/decoding and that using ASN.1 would make an interesting and useful study.

• Even though a scheme was provided for implementing a Extended Multi-party test system using a FCTS, no actual implementation was performed. The use of such a system to test a multi-layer IUT would provide an interesting area of study.
Bibliography


Appendix A

Ferry Control Protocol and Ferry PDUs

Passive Ferry Clip State Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>FT-CONN ind</td>
<td>P1: FT-CONN rsp</td>
<td>connected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P2: FT-DISC req</td>
<td>idle</td>
</tr>
<tr>
<td></td>
<td>FT-DISC ind</td>
<td>none</td>
<td>idle</td>
</tr>
<tr>
<td></td>
<td>FT-ERROR ind</td>
<td>FT-DISC req</td>
<td>idle</td>
</tr>
<tr>
<td></td>
<td>FD-DATA req</td>
<td>none</td>
<td>idle</td>
</tr>
<tr>
<td>connected</td>
<td>FT-DISC ind</td>
<td>none</td>
<td>idle</td>
</tr>
<tr>
<td></td>
<td>FT-ERROR ind</td>
<td>FT-DISC req</td>
<td>idle</td>
</tr>
<tr>
<td></td>
<td>FD-DATA ind</td>
<td>P3: FT-DATA req (loop back)</td>
<td>connected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P4: FD-DATA ind</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>P5: control actions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FD-DATA req</td>
<td>P6: FT-DATA req(FY-DATA)</td>
<td>connected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P3: none</td>
<td></td>
</tr>
</tbody>
</table>

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 control actions and generate FY-CNTL (using FT-DATA req) back to active ferry.
- P6 – the passive ferry clip is not in loop-back mode.
### Active Ferry Clip State Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>FM-CONN req</td>
<td>FT-CONN req</td>
<td>connecting</td>
</tr>
<tr>
<td>idle</td>
<td>FM-DISC req</td>
<td>none</td>
<td>idle</td>
</tr>
<tr>
<td>idle</td>
<td>FM-CNTL req</td>
<td>FT-CONN cnf</td>
<td>connected</td>
</tr>
<tr>
<td>idle</td>
<td>FD-DATA req</td>
<td>FT-DISC req</td>
<td>idle</td>
</tr>
<tr>
<td>idle</td>
<td>FT-DISC ind</td>
<td>FT-DISC cnf</td>
<td>idle</td>
</tr>
<tr>
<td>idle</td>
<td>FT-ERROR ind</td>
<td>FM-DISC ind</td>
<td>idle</td>
</tr>
<tr>
<td>connecting</td>
<td>FM-DISC req</td>
<td>FT-DISC req</td>
<td>idle</td>
</tr>
<tr>
<td>connecting</td>
<td>FT-CONN cnf</td>
<td>FT-CONN cnf</td>
<td>connected</td>
</tr>
<tr>
<td>connected</td>
<td>FT-DISC ind</td>
<td>FT-DISC ind</td>
<td>idle</td>
</tr>
<tr>
<td>connected</td>
<td>FM-DISC req</td>
<td>FT-DISC req</td>
<td>idle</td>
</tr>
<tr>
<td>connected</td>
<td>FD-DATA req</td>
<td>FT-DATA req(FY-CNTL)</td>
<td>connected</td>
</tr>
<tr>
<td>connected</td>
<td>FT-DATA req</td>
<td>FT-DATA req(FY-DATA)</td>
<td>connected</td>
</tr>
<tr>
<td>connected</td>
<td>FT-DATA ind</td>
<td>P1: FD-DATA ind</td>
<td>connected</td>
</tr>
<tr>
<td>connected</td>
<td>FT-DISC ind</td>
<td>FM-DISC ind</td>
<td>idle</td>
</tr>
<tr>
<td>connected</td>
<td>FT-ERROR ind</td>
<td>EXCEPTION</td>
<td>idle</td>
</tr>
</tbody>
</table>

**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, 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.
**FY-DATA PDU Format**

<table>
<thead>
<tr>
<th>T</th>
<th>LI</th>
<th>U/L</th>
<th>*</th>
<th>M</th>
<th>length</th>
<th>test data</th>
</tr>
</thead>
</table>

- **T** - FY-PDU Type.
  - T = 0 for FY-DATA PDUs.
- **LI** - Layer Identifier.
  - LI = 1, ..., 7 to correspond to OSI layers 1 through 7.
- **U/L** - Upper/Lower Interface Bit.
  - 0 => test data to/bom lower service interface of IUT.
  - 1 => test data to/from upper service interface of IUT.
- **M** - More.
  - 0 => test data is not segmented or is last of a series of segments.
  - 1 => at least one segment follows this one.
- **length** - number of bytes of test data within the FY-DATA PDU.

**FY-CNTL PDU Format**

<table>
<thead>
<tr>
<th>T</th>
<th>L</th>
<th>*</th>
<th>*</th>
</tr>
</thead>
</table>

- **T** - FY-PDU Type.
  - T = 1 for FY-CNTL PDUs.
- **L** - Loop-Back Bit.
  - 0 => set passive ferry into normal (non loop-back) mode.
  - 1 => set passive ferry into loop-back mode.
- ***** - Reserved.
Appendix B

Passive Ferry Clip Lower Mapping Modules

"Forth" Source Code for Lower Mapping Module in MPT Environment

( ----------------------------------------------- )
( File Title : F_CSTATES.F )
( Document Id : Passive Ferry Clip Component )
( Project Id : Ferry Clip )
( Date : August 16, 1988 )
( System : PT 3.2 / MPT 1.2 )
( Version : 1.0 )
( Author : Neville J. Parakh )
( ----------------------------------------------- )

( This file contains routines that implement the states for the )
( receipt of COMMANDS from the "Passive Ferry FSM" )

( Routine called when an INVALID COMMAND is received )
: INVALID_COMMAND ( STATE -- )
   CR ." CSTATE" ." : INVALID COMMAND RECEIVED" CR
;

( Send the DATA PACKET to the ACTIVE FERRY )
: SEND_BP2 ( -- SUCCESS | FAILURE )
   BUFFER-POOL2 REMOVE_FROM_QUEUE
   SUCCESS =
   IF

75
DUP 1+ DUP C0 256 * SWAP 1+ C0 +  ( FY-DATA PDU LENGTH )
3 >>= # 3+  ( NUMBER OF BYTES )
LOCAL-BUFFER C!  ( STORE LENGTH IN FIRST
                   BYTE OF STRING )
LOCAL-BUFFER 1+  ( DESTINATION FOR CMOVE )
LOCAL-BUFFER C0  ( LENGTH FOR CMOVE )
CMOVE
LOCAL-BUFFER SENDD  ( SEND DATA FRAGMENT TO
                        ACTIVE FERRY )
SUCCESS  ( DATA GOT SENT )
ELSE
FAILURE  ( NO DATA TO SEND )
THEN
;

( Send the CONTROL PACKET to the ACTIVE_FERRY )
: SEND_BP3 ( -- SUCCESS | FAILURE )
BUFFER-POOL3 REMOVE_FROM_QUEUE
SUCCESS =
IF
  3 LOCAL-BUFFER C!  ( STORE LENGTH IN FIRST
                       BYTE OF STRING )
LOCAL-BUFFER 1+  ( DESTINATION FOR CMOVE )
3  ( LENGTH FOR CMOVE )
CMOVE
LOCAL-BUFFER SENDD  ( SEND DATA FRAGMENT TO
                        ACTIVE FERRY )
SUCCESS  ( CONTROL PACKET WAS SENT )
ELSE
FAILURE  ( NO CONTROL PACKET TO SEND )
THEN
;

( C State 0 - Disconnected state, waiting for X.25 to send a )
( CALL-CONNECTED packet )
: CSTATE0 ( COMMAND -- )
: DOCASE
  CASE FT-CONNresp { ( RETURN FT-DISCind )
                      FT-DISCind ADD_TO_LMAP_RETURN
                    }
  CASE FT-DATAreq { ( RETURN FT-ERRORind )
                    }
  CASE FT-ERRORind { ( RETURN FT-ERRORind )
                    }
  CASE FT-DISCind { ( RETURN FT-DISPind )
                     FT-DISPind ADD_TO_LMAP_RETURN
                   }
  OTHERWISE { ( RETURN FT-DISPind )
              FT-DISPind ADD_TO_LMAP_RETURN
            }
END
APPENDIX B. PASSIVE FERRY CLIP LOWER MAPPING MODULES

FT-ERRORind ADD_TO_LMAP_RETURN
}
CASE FT-DISCreq { } ( IGNORE SINCE ALREADY DISCONNECTED )
CASE DUP { ( INVALID COMMAND )
0 INVALID_COMMAND }
ENDCASE
;

(C State 1 - Transmitted DISC, waiting for a response )
: CSTATE1 ( COMMAND -- )
DOCASE
CASE FT-CONNresp { FT-DISCind ADD_TO_LMAP_RETURN ( RETURN FT-DISCind ) }
CASE FT-DATAreq { FT-ERRORind ADD_TO_LMAP_RETURN ( RETURN FT-ERRORind ) }
CASE FT-DISCreq { } ( IGNORE SINCE ALREADY DISCONNECTED )
CASE DUP { 1 INVALID_COMMAND } ( INVALID COMMAND RECEIVED )
ENDCASE
;

(C State 2 - Connected state, waiting for data from "Active Ferry")
: CSTATE2 ( COMMAND -- )
DOCASE
CASE FT-CONNresp { } ( IGNORE FT-CONNresp )
CASE FT-DISCreq { DISC ( TRANSMIT DISC FRAME )
1 LSTATE ! ( MOVE TO STATE 1 ) }
CASE FT-DATAreq { WINDOW?
1 =
IF ( YES IT CAN )
SEND_BP3
SUCCESS =
IF
FT-DATAsent DATA-RESULT !
ELSE
SEND_BP2
SUCCESS =
  IF
   FT-DATAsent DATA-RESULT !
  ELSE
   " PASSIVE FERRY: OUTPUT BUFFERS EMPTY"
   QQ_EMPTY
   FT-DATAnotsent DATA-RESULT !
  THEN
  THEN
  ELSE ( NO DATA CANNOT BE SENT )
   FT-DATAnotsent DATA-RESULT !
  THEN
ENDCASE

CASE DUP { 2 INVALID_COMMAND } ( INVALID COMMAND RECEIVED )
( This file contains routines that implement the states for the )
( receipt of FRAMES and PACKETS by the "Lower Mapping Module" )

(Routine that is called if a request is made to remove an )
(element from an EMPTY OUTPUT QUEUE )

: OQ_EMPTY ( STRING -- )
   CR COUNT TYPE CR

(Routine that is called if the LMAP-RETURN queue is full )

: LMAP_RETURN_FULL ( STRING -- )
   CR COUNT TYPE CR

(Routine to add a message to the LMAP-RETURN queue )

: ADD_TO_LMAP_RETURN ( VALUE -- )
   LMAP-RETURN ADD_TO_QUEUE
   SUCCESS =
   IF
     ( SPACE AVAILABLE )
     \* ( STORE MESSAGE IN QUEUE )
   ELSE
     ( NO SPACE AVAILABLE )
     DROP
     ( DROP VALUE FROM STACK )
   " LMAP RETURN QUEUE FULL" LMAP_RETURN_FULL
   THEN

;
APPENDIX B. PASSIVE FERRY CLIP LOWER MAPPING MODULES

: FPSTATE0 ( -- )
  FRAME-TYPE @
  R*I = ( I-FRAME ? )
  IF
  PACKET-TYPE @
  DOCASE
    CASE R*CALLREQ { DOER_INITIALIZE_FERRY
      O PAINT CLEAR_TEXT
      " PASSIVE FERRY CONNECTED" BTYPE WCR
      FT-CONNind ADD_TO_LMAP_RETURN ( RETURN
      FT-CONNind )
    2 LSTATE ! ( MOVE TO STATE 2 )
    }
  CASE DUP { } ( IGNORE ALL OTHER PACKETS )
  ENDCASE
  THEN
  ( IGNORE ALL FRAMES )

( FP State 1 - Transmitted DISC - waiting for a response )

: FPSTATE1 ( -- )
  FRAME-TYPE @
  R*I = ( I-FRAME ? )
  IF
  PACKET-TYPE @
  DOCASE
    CASE R*RESTARTREQ { FT-DISCind ADD TO LMAP RETURN ( RETURN
    FT-DISCind )
    0 LSTATE ! ( MOVE TO STATE 0 )
    }
  CASE DUP { } ( IGNORE ANY OTHER PACKETS )
  ENDCASE
  ELSE
  FRAME-TYPE @
  DOCASE
    CASE R*DISC ORCASE
    R*DM ORCASE
    R*UA
      { O LSTATE ! ( MOVE TO STATE 0 )
      }
      CASE DUP { } ( IGNORE ANY OTHER FRAMES )
APPENDIX B. PASSIVE FERRY CLIP LOWER MAPPING MODULES

ENDCASE
THEN
;

(FP State 2 - Connected state, waiting for data from "Active Ferry")
: FPSTATE2 ( -- )
FRAME-TYPE @
R*I = ( I-FRAME ? )
IF
PACKET-TYPE @
DOCASE
CASE 0 { } ( IGNORE NULL PACKET )
CASE R*RRP { } ( IGNORE RR PACKET )
CASE R*RNRP { } ( IGNORE RNR PACKET )
CASE R*DATAP { FT-DATAind ADD_TO_LMAP_RETURN ( RETURN
FT-DATAind )
DATA-POINTER @ ( X.25 DATA BUFFER )
INPUT-BUFFER ( INPUT BUFFER )
DATA-LENGTH @ ( LENGTH OF FY-PDU )
CMOVE ( MOVE DATA FROM X.25 BUFFER INTO INPUT BUFFER )
}
CASE R*RESTARTREQ { FT-DISCind ADD_TO_LMAP_RETURN ( RETURN
FT-DISCind )
0 LSTATE ! ( MOVE TO STATE 0 )
}
CASE R*CLEARREQ { FT-DISCind ADD_TO_LMAP_RETURN ( RETURN
FT-DISCind )
0 LSTATE ! ( MOVE TO STATE 0 )
}
CASE DUP { FT-ERRORind ADD_TO_LMAP_RETURN ( RETURN
FT-ERRORind )
" PASSIVE FERRY GETS UNDEFINED EVENT = " RTYPE PACKET-TYPE @ W. WCR
}

ENDCASE
ELSE
FRAME-TYPE @
DOCASE
CASE R*DISC ORCASE
R*DM { FT-DISCind ADD_TO_LMAP_RETURN ( RETURN
APPENDIX B. PASSIVE FERRY CLIP LOWER MAPPING MODULES

FT-DISCind )

0 LSTATE !

" PASSIVE FERRY GOT DISC FROM ACTIVE" YTYPE WCR

CASE DUP { }

ENDCASE

THEN


"C" Source Code for Lower Mapping Module in OSI-PTE Environment

/******************************************************************************
/* Program: Passive Ferry Clip. */
/* Module: Lower Mapping Module routines to use X.25 PLP */
/* Author: Neville J. Parakh */
/* Date: January 12, 1989 */
/* Description: */
/* This module contains two main routines: */
/* (1) LMAPUp: */
/* This is the routine called as the result of an incoming event from the FTMP. It is written as a state machine. Furthermore, it is dependent on the FTP being used. All references to the FTP being used are localized in this module. */
/* (2) LMAPDown: */
/* This is the routine called to send a FY- packet to the Active Ferry via the FTMP on the Passive Ferry side. It is called as a procedure from routines in the Passive Ferry FSM Module. */
/******************************************************************************/

#include <stdio.h>
#include "n_header.h"
#include "passive/error.h"
#include "passive/param.h"
#include "passive/events.h"
#include "passive/fsm/fr_nva.h"
#include "passive/fsm/fr_ccb.h"
#include "passive/fsm/fr_states.h"
#include "passive/fsm/fr_asp.h"
#include "passive/fsm/decode.h"

/* Variables to hold the Network address for the Passive and Active FTMPs - assigned in LMAPUp when we get a N_CONN_REQ_EID */
static N_ADDR passive_ftmp_addr;
static N_ADDR active_ftmp_addr;
static N_QOS qos;

void LMAPUp(frn,cid,eid,epa)
struct fr_nva *frn;
CID *cid;
EID eid;
EPA *epa;
{
    int fr_decode();
    void PassiveFSM();
    void ERROR();
    struct fr_ccb *ptr_ccb;
    struct N_CONN_IND_EPA *ptr_conn_ind;
    struct N_DATA_IND_EPA *ptr_data_ind;
    struct N_EXPD_IND_EPA *ptr_expd_ind;
    struct N_DISC_IND_EPA *ptr_disc_ind;
    FerryPDU token;
    extern N_ADDR passive_ftmp_addr;
    extern N_ADDR active_ftmp_addr;
    extern N_QOS qos;

    switch(eid)
    {
    case N_CONN_IND_EID:
        ptr_conn_ind = (struct N_CONN_IND_EPA *)epa;
        passive_ftmp_addr = ptr_conn_ind->dst_naddr;
        active_ftmp_addr = ptr_conn_ind->src_naddr;
        qos = ptr_conn_ind->nqos;
        if (ptr_conn_ind->nsdu != RXBUF_NULL)
            FreeRxBuf(ptr_conn_ind->nsdu);
        /* Call Passive Ferry FSM */
        PassiveFSM(FT_CONNind,frn->ccb,(FerryPDU *)NULL,
                   TX_NULL,RXBUF_NULL);
        break;

    case N_DATA_IND_EID:
        ptr_data_ind = (struct N_DATA_IND_EPA *)epa;
        bzero(&token,sizeof(token));
        fr_decode(&token,(b8)0,ptr_data_ind->nsdu,(b8)0);
APPENDIX B. PASSIVE FERRY CLIP LOWER MAPPING MODULES

/* Decode the header byte */
if (token.t_bit == (b8)0)
{
    /* Decode FY-PDU header byte */
    token.li_bits = (token.header & 0x70) >> 4;
    token.ul_bit = (token.header & 0x08) >> 3;
    token.reserved1 = (token.header & 0x06) >> 1;
    token.m_bit = token.header & 0x01;
    TruncateRxHead(ptr_data_ind->nsdu, token.data);
    PassiveFSM(FT_DATAind, frn->ccb, &token, TX_NULL, ptr_data_ind->nsdu);
}
else
{
    /* Decode the FY-CNTL PDU */
    token.l_bit = (token.header & 0x40) >> 6;
    token.reserved2 = token.header & 0x3f;
    TruncateRxHead(ptr_data_ind->nsdu, token.data);
    PassiveFSM(FT_DATAind, frn->ccb, &token, TX_NULL, ptr_data_ind->nsdu);
}
break;

case N_DACK_IND_EID:
    PassiveFSM(FT_ERRORind, frn->ccb, (FerryPDU *)NULL, TX_NULL, RXBUF_NULL);
    break;

case N_EXPD_IND_EID:
    ptr_expd_ind = (struct N_EXPD_IND_EPA *)epa;
    if (ptr_expd_ind->nsdu != RXBUF_NULL)
        FreeRxBuf(ptr_conn_ind->nsdu);
    PassiveFSM(FT_ERRORind, frn->ccb, (FerryPDU *)NULL, TX_NULL, RXBUF_NULL);
    break;

case N_RSET_IND_EID:
    ptr_ccb = frn->ccb;
    PostEvent(ptr_ccb->nid2, &(ptr_ccb->cid2), N_SERVICE_DN, (EID)N_RSET_RSP_EID, (EPA *)NULL);
    if (frn->state == CONNECTED)
        PassiveFSM(FT_ERRORind, ptr_ccb, (FerryPDU *)NULL, TX_NULL, RXBUF_NULL);
APPENDIX B. PASSIVE FERRY CLIP LOWER MAPPING MODULES

break;

case N_RSET_CFM_EID:
    ptr_ccb = frn->ccb;
    if (frn->state == CONNECTED)
        PassiveFSM(FT.ERRORind,ptr_ccb,(FerryPDU *)NULL,
                   TX_NULL,RXBUF_NULL);
    break;

case N_DISC_IND_EID:
    ptr_disc_ind = (struct N_DISC_IND_EPA *)epa;
    /* if (ptr_disc_ind->nsdu != RXBUF_NULL)
    FreeRxBuf(ptr_disc_ind->nsdu); */
    PassiveFSM(FT_DISCind,frn->ccb,(FerryPDU *)NULL,
               TX_NULL,RXBUF_NULL);
    break;

default:
    ERROR(INVALID_ASP1,(void *)NULL);
    PassiveFSM(FT_ERRORind,frn->ccb,(FerryPDU *)NULL,
               TX_NULL,RXBUF_NULL);

}
}

void LMAPDown(frc,fr.asp.ptr.txbuf)
struct fr_ccb *frc;
int fr.asp;
TXBUF *ptr_txbuf;
{
    void ERROR();
    void PassiveFSM();
    struct N_CONN_RSP_EPA n_conn_rsp_epa;
    struct N_DISC_REQ_EPA n_disc_req_epa;
    struct N_DATA_REQ_EPA n_data_req_epa;
    extern N_QOS qos;

    switch(fr.asp)
    {
        case FT_CONNrsp:
            n_conn_rsp_epa.rsp_naddr = passive_ftmp_addr;
n_conn_rsp_epa.modified = FALSE;
n_conn_rsp_epa.modification_reason = (OCTET)0;
n_conn_rsp_epa.rcpt = FALSE;
n_conn_rsp_epa.expd = FALSE;
n_conn_rsp_epa.nqos = qos;
n_conn_rsp_epa.extra = (N_EXTRA)0;
n_conn_rsp_epa.nsdu = TX_NULL;
PostEvent(frc->nid2,&(frc->cid2),N_SERVICE_DN,
           (EID)N_CONN_RSP_EID,&n_conn_rsp_epa);
break;

case FT_DISCreq:
    n_disc_req_epa.rsp_naddr = passive_ftmp_addr;
    n_disc_req_epa.modified = FALSE;
    n_disc_req_epa.modification_reason = (OCTET)0;
    n_disc_req_epa.nreason = N_DISCONNECT_UNDEFINED;
    n_disc_req_epa.cause = (OCTET)0;
    n_disc_req_epa.diagnostic = (OCTET)0;
    n_disc_req_epa.nsdu = TX_NULL;
    PostEvent(frc->nid2,&(frc->cid2),N_SERVICE_DN,
               (EID)N_DISC_REQ_EID,&n_disc_req_epa);
break;

case FT_DATAreq:
    n_data_req_epa.nsdu = ptr.txbuf;
    n_data_req_epa.creq = FALSE;
    n_data_req_epa.qbit = FALSE;
    n_data_req_epa.mbit = FALSE;
    PostEvent(frc->nid2,&(frc->cid2),N_SERVICE_DN,
               (EID)N_DATA_REQ_EID,&n_data_req_epa);
break;

case FDLCONNind:
    PostEvent(frc->nid3,&(frc->cid3),LOWER_IUT_SERVICE_UP,
               (EID)LOWER_IUT_CONN_IND,(EPA *)NULL);
break;

case FDL_DISCind:
    PostEvent(frc->nid3,&(frc->cid3),LOWER_IUT_SERVICE_UP,
               (EID)LOWER_IUT_DISC_IND,(EPA *)NULL);
break;
case FDU_CONNind:
    PostEvent(frc->nid4, &(frc->cid4), UPPER_IUT_SERVICE_DN,
              (EID)UPPER_IUT_CONN_IND, (EPA *)NULL);
    break;

case FDU_DISCind:
    PostEvent(frc->nid4, &(frc->cid4), UPPER_IUT_SERVICE_DN,
              (EID)UPPER_IUT_DISC_IND, (EPA *)NULL);
    break;

    /* Invalid ASP's */
    /* case FT_CONNind: */
    /* case FT_DISCind: */
    /* case FT_DATAind: */
    /* case FT_ERRORind: */
    /* case FD_DATAreq: */
    /* case FD_DATAind: */
    default:
        ERROR(INVALID_ASP2, (void *)NULL);
        PassiveFSM(FT_ERRORind, frc, (FerryPDU *)NULL,
                    TX_NULL, RXBUF_NULL);
    }