On TESTGEN+, An Environment for Protocol Test Generation and Validation

by

Zhou Jingsong

B.Sc., Changsha Institute of Technology, China, 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

University of British Columbia
December 1992

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.

(Signature)

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date Feb 10 1993

DE-6 (2/88)

Abstract

This thesis addresses a significant tool ,$TESTGEN+$, for protocol test generation and validation. $TESTGEN+$ consists of two basic components:TESTGEN for protocol test generation, and TESTVAL for test validation.

TESTGEN, is a protocol TEST Generation and Selection Environment for conformance testing which has been developed at the University of British Columbia. This environment is menu driven and unique in that it is effective, general, flexible and portable. It is based on an intermediate extended transition system formalism and directly supports ASN.1 and Estelle. The test generation method adopted in the environment integrates both the control flow testing and the data flow testing with parameter variation. Furthermore, test generation and selection are integrated and guided by user-defined test suite generation constraints and parameter variation constraints. The environment will serve as a useful testbed for experimenting with test generation and selection as well as being a productive system for generating useful test suites for real-life protocols.

TESTVAL is a test suite validator, which is based on a trace analyzer, and which allows test cases to be validated against their protocol specification. Using TESTVAL, the validation of the test suite for the LAPB protocol defined in the ISO 9646 standard document has been performed and the validation result is presented in the thesis.

Contents

List of Figures

Acknowledgement

I would like to acknowledge the tremendous amount of help, support and guidance given to me by my supervisor, Dr. Son Vuong, throughout the course of my thesis work and studies at UBC.

Also, I would like to thank Dr. Samuel T. Chanson for serving as the second reader of this thesis, and also for his invaluable comment and help on this thesis.

Finally, I would like to extend my thanks to the department of Computer Science, and many graduate students in this department, for the financial support as well as the great help and encouragement during my studies at UBC.

Chapter 1

Introduction

1.1 Motivation and Objectives

Conformance testing of communication protocols is commonly used to ensure that the implemented protocol meets the design specifications and operates correctly in the communication network. The usual approach in conformance testing is based on the Finite State Machine (FSM) model, which is thus inherently limited to the testing of the control flow part of the protocol.

In the past, several methods have been put forward to handle the data flow of the protocol [$Ura-88$], [$Sra-87$], [$Wu-89$]. However these methods do not consider the dependencies between transitions and they may therefore miss hidden interactions among the service primitives exchanged in different parts of the protocol.

The Test Suite Generation (TSG) method used in TESTGEN integrate both the control flow and the data flow testing with parameter variation and can produce test cases to cover any path defined by the service primitives that an IUT is allowed to exchange with its environment. It also integrates the generation and selection of test suites by providing an environment where

various $TSG - constraints$ and constraints on the service primitive parameter can be defined by the user to control the limit and depth of the testing tree, it thus provide a great extent of flexibility.

With a growing number of standardized test suites generated, it is very important to ascertain the validity of the test suites with respect to the protocol specification. Since, TESTGEN is not suitable for test case validation , a new validator based on a trace analyzer, is developed. This test case validator is incorporated into TESTGEN to produce an enhanced version called $TESTGEN+$ which can handle both test case generation and validation.

1.2 Thesis Contributions

The main contributions of my thesis include the following:

. Contributions in the design of TESTGEN.

. Implementation of the test generation kernel part of a version of TESTGEN, including the subpath identification and the test suite specification in TTCN.

. Modifications of a trace analyzer to produce a test case validator, and validation of the LAPB test suite as defined in the ISO 9646 standard document.

1.3 Thesis Outline

In this thesis, I first discuss about the strategy in building the test generation kernel part of TESTGEN, and then the design and the implementation of that test kernel. Finally, I present

the modification of a trace analyzer to produce a test case validator and the results of validating a standardized LAPB test suite using this validator.

Chapter 2

Strategy

2.1 Extended Transition System

In this chapter, I mainly introduce the definition of an extendented transition system , and the algorithm for building the test kernel.

The old FSM formalism describes a protocol in terms of states and transitions. An *input/output* pair (i/o) is associated with each transition. This specification is suited to describe the behaviour of very simple systems. However in the real world, we need a formalism which can describe the real world protocols, and thus cause the appearance of the ETS.

The Extended Transition System (ETS) model is a quadruple $ETS = (Q, E, T, Q_{int})$ where

. Q is the set of states of the ETS,

. E is the set of events of the ETS,

. T is the set of transitions of ETS,

. Q_{int} is the initial state of ETS.

The set of states Q denotes the set product:

$Q = STATES * VAR * C * TIMER$

Q_{init} represents the initial control state S_{init}, the initial values of all variables and timers.

The set of event E denotes the set product:

$E = ISP * OSP * PDU$

ISP is the set of Input Service Primitives, OSP is the set of Output Service Primitives, PDU is the set of Protocol Data Units.

A transition in T is a relation on $Q * E * Q$ represented by a set product

$T = EPRED * AFN$

where an enabling predicate epred (a subset of EPRED) is a boolean function:

$epred : Q * E- > true, false$

(a transition can be executed only if its associated enabling predicate is true) and an action function afn (a subset of AFN) is a function:

$afn : Q * E- > Q * E$

which affects variables in VAR, timer in TIMER and parameters of OSPs and PDUs.

A transition can be executed if and only if the ISP and PDU associated with the transition (if any) are received and if the enabling predicate is true. When a transition fires, the associated action function is executed automatically. Variables and timers are set, OSP(s) and PDU(s) are assembled (their parameters are set), OSP(s) and PDU(s) are sent.

2.2 Definition

Interaction path(testing subpath):

An interaction path IP is a ETS observable path on which a sequence of interactions between the protocol and its external environment occurs, starting from the Q_{init} and ending in the same state.

I/Opath:

An I/O path is the ETS observable track $P_1, P_2....P_k$ in an IP, where

P_i has the isp I_i with parameters $Si_{1i}, ...Si_{mi}$,

P_i has the osp O_i with parameter $So_{1i}, ...So_{mi}$,

P_i has the pdu U_i with parameter $Su_{1i}, ...Su_{mi}$.

2.3 Backtracking Algorithm

Once a protocol ETS data structure has been generated, we can use the following algorithm to identify all its interaction pathes.

Loop (if there is one or more transitions have been fired in set of Q)

Step1: find an unmarked transition with the lowest transition number in the current state.

Step2: check whether the current transition can been applied or not. if the answer is TRUE, fire and mark this transition.

Step3: if there is no unmarked transition in this state, back one step, and go to step1.

step4: the new state Q is the receive state of the current transition.

step5: if the $newstate = Q_{init}$, store one interaction path, and $currentstate = Q_{init}$.

2.4 Comparison

Comparing with other strategies, our strategy has the following advantages:

Practicality:

Unlike some other strategies which can only test simple toy protocols, our test strategy can test real protocols, and thus provides a practical tool.

flexibility:

Due to the flexibility of the TSG constraint part, we can get different test cases depending on the user's different requests.

generality:

Unlike some other strategies, which limit their applications only to some certain specific protocols, our TSG tool can test different protocol specifications. So it is a generalized conformance test tool.

Chapter 3

Design and Architecture

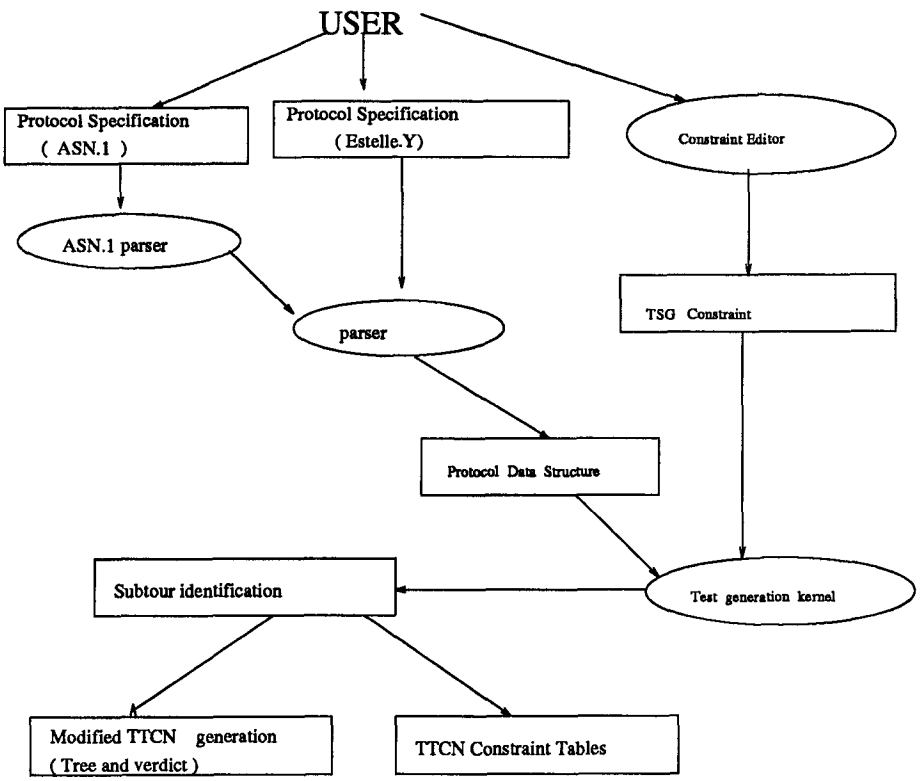

Figure 3.1: Model of Test Case Generation Design

In this chapter, we will discuss in detail the design and architecture of each part of the TEST-GEN.

From Figure 3.1, we can see the whole test process be mainly divided into three parts, parser part, user input part and the backtracking test part.

The user input is made of three parts: ASN.1 specification, Estelle.Y specification, and constraint part.

The parser part contains two parts, ASN.1 parser and protocol parser. The ASN.1 parser gets the ASN.1 specification from the user, produce ASN.1 input to the parser.

The protocol parser takes Estelle.Y specification provided by user, and the output provided by the ASN.1 parser, to produce the static data structure of the whole protocol. It becomes the input to the backtracking testing part.

The backtracking test part takes the protocol static data structure as the input to produce the subtour identification as the output. Furthermore, it will produce the Modified TTCN Generation (tree and verdict) and TTCN Constraint Tables.

3.1 User Input Part

The user should provide three kinds of inputs, the Estelle.Y Protocol specification, the ASN.1 Protocol specification, and the Constraint part.

The ASN.1 Protocol specification specifies the protocol data structure using the ASN.1 definition. Here is a simple example, which builds a subset of the X25 ASN.1 specification.

```
X25 DEFINITIONS ::=
```

```
BEGIN


NetworkAsp ::= CHOICE
               {
               NConnectRequest,
               NconnectIndication
               }


NConnectRequest ::= SEQUENCE
                        {
                        calledAddress  OCTET STRING,
                        callingAddress OCTET STRING (SIZE(0..109)),
                        receiptConfirmationSelection
                                        ENUMERATED { nouse(0), use(1) },
                        qualityOfService SEQUENCE
                                        {
                                        throughput SEQUENCE
                                                {
                                                toTargetValue INTEGER,
                                                toLeastValue  INTEGER
                                                },
```

```
                                        transitDelay SEQUENCE
                                                {
                                                targetValue INTEGER,
                                        leastValue  INTEGER
                                                }
                                        }
                        }

NConnectIndication ::= SEQUENCE
                        {
                        calledAddress   INTEGER,
                        callingAddress  OCTET STRING
                        }

x25Pdu ::= CHOICE
                {
                CallRequest,
                CallAccepted
                }

CallRequest     ::=     SEQUENCE
```

```
                    {
                    generalFormatId      NameBitString4,
                    logicalChannelId     BitInteger12
                    }

CallRequest         ::=    SEQUENCE
                           {
                           generalFormatId    NamedBitString4,
                           logicalChannelId   BitInteger12,
                           facilityLength     BitInteger4
                           }

NameBitString4 ::= BIT STRING (SIZE(4,4))

BitInteger12   ::= INTEGER (SIZE(12..12))

BitInteger4    ::= INTEGER (SIZE(4..4))

END
```

The definition of the Estelle.Y protocol specification was made by ourselves by modifying the Normal Form Estelle specification.

Here is a simple example of the Estelle.Y specification.

```
Specification  FDDIMAC;


Const


COUNT = -100 : int;

tkrecvd = true : boolean;

frrecvd = false : boolean;

MAXSIZEaaabb = 1000 : int;

STR = ''stri-.*ngggg'' : CHAR_STR;


Var


abcddddd, ccc, n1, n2: int

a2: char_str;


ISP

NconnectRequest NASP;

NconnectIndication NASP;

PHY_DATA_ind PSAP;

nisp nsap;
```

```
T_DATA_req   tsap;


OSP

DL_data_req  dsap;

DL_data_ind  dsap;

T_DATA_ind   tsap;


PDU


TPDU  sent_in DL_data_ind,

      recv_in PHY_DATA_ind;

NPDU  sent_in T_DATA_ind, recv_in T_DATA_req;

SPDU  recv_in T_DATA_req;

CallRequest   recv_in NConnectRequest;


Timer


TVX  250;

THT  380;


State
```

```
S0,S1,S2,S3,s4,s5;

Trans
      from S3
          to S1
              when NConnectrequest
                provided (( NconnectRequest.calledAddress + 1055 * n2 / ccc) = 100 )
                priority 10
                  OUTPUT TPDU, CallRequest
                    pretty
                      begin
                    reset(TVX);
                     NConnectRequest.calledAddress := NConnectRequest.qualityOfService
                     NconnectRequest.callingAddress := (10 + CallRequest.logicalChanne
                    STOP(THT);
                    end;

          to S2
            begin
              if (m1) then
```

```
    if (m3 and m4) then
      begin
        n2 := NConnectRequest.calledAddress;
        reset(TVX)
      end
    else if (m4 or m1) then
                          begin
                            n2 := n1;
                            ccc := 0;
                            n1 := 1
                          end;
  START(TVX);
end;


TO S0
WHEN SPDU
PROVIDED (m2 = m3)
ugly
BEGIN
  IF (m3 = true) THEN n1 := n2*10;
END;
```

```
        from S2
            to S0
               PROVIDED (m1 or (not m2) and m3 and STOPPED(TVX))
                   priority 20
                   BEGIN
                      start(THT);
                   END;




TRANS
        FROM s4
           to s5
               WHEN TPDU
                   provided m4
                     Begin
                         m3 := true;
                     End;
end.
```

The constraint part can either be set by default or be interactively specified by the user to

guide the selective identification process.

The user can specify the minimum and maximum uses of each state, transition, constant, input service primitive, output service primitive, protocol data unit and timer for the whole protocol testing and subpath generation. This controls the depth of the testing tree by the user himself. Also, the user can specify the constraint for the primitives; it gives a certain field of the data which can be used for this service primitive during the testing. Meanwhile, the user can set the default, and the system can automatically give the entire constraints for the testing.

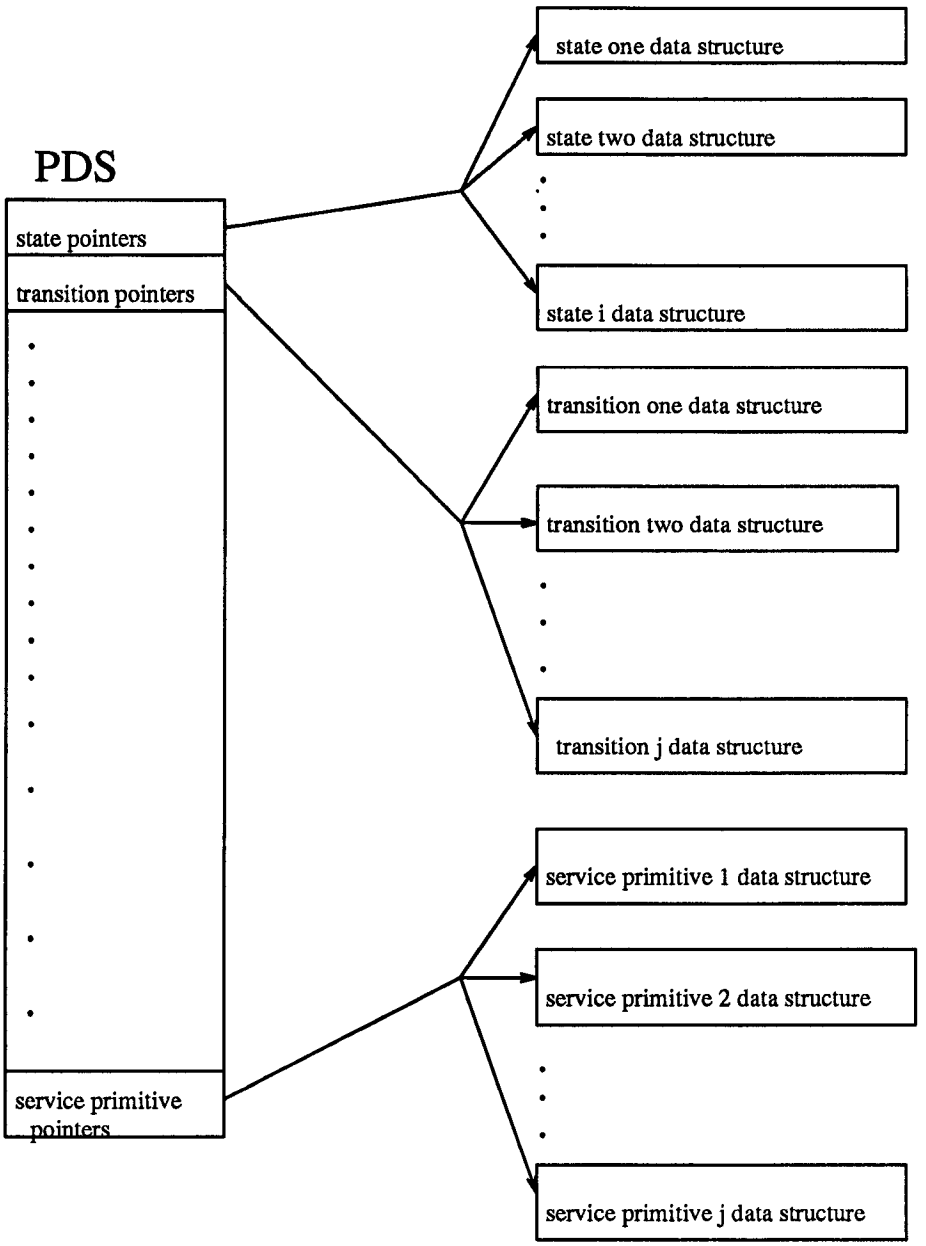

Figure 3.2: Model of Protocol Data Structure.

3.2 Parser Part

3.2.1 ASN.1 Parser

Abstract Syntax Notation One (ASN.1) is a widespread, standardized notation that provides its user with an easy to read, yet powerful definition language for describing communication protocol data structures in terms of types and values of these types. Here, we mainly use this definition to specify the data type of these data structure, and we also limit several basic data type to be used in the specification of service primitive.

3.2.2 Protocol Parser

Protocol parser takes the Estelle.Y specification, the ASN.1 parser output to produce the static data structure of the protocol.

As we can see from figure 3.2, The PDS is a complex structure which has the entire pointers for each of the protocol static data structures, and these data structures contain the whole information which each test part needs.

Inside the input service primitive, output service primitive and protocol data unit, there is a pointer to the ASN.1 data structure which contains the detailed service primitive information. When we do the protocol testing, we get a pointer pointing to the PDS after we call the parser function, then we get the whole information that we need in the backtracking testing.

3.3 Backtracking Testing Part

3.3.1 Dynamic Data Structure

In order to get the data value from every data structure which we defined in the static data part, to produce the subtour identification and TTCN test generation, we should define a dynamic data structure.

Each test case consists of some tested states and transitions; inside the state, we should write down the actual value of every variable, timer, and service primitive.

Inside the dynamic data structure, we define a backtracking timer structure; it records the state (stopped or running) and value of the timer in each test state.

Inside the dynamic data structure, we define a backtracking variable structure, it records the actual value for each variable in every testing state.

Also, we define the backtracking service primitive structure in the dynamic data structure, and we use it to record the actual value for each service primitive in every testing state.

Then, we can define the backtracking state structure; it contains the transition number which starts from here, and the three data structures we described above. Therefore, the backtracking state structure can get the actual values of timer, variable, and service primitive in each backtracking state.

In order to check the constraint part which the user specified, we should define a data structure to record the times for each state, transition, isp, osp, constant, pdu and other data structures which we have tested.

For each subpath, we need to get the information about how many backtracking states it has,

and the entire information about that state. So, we define a backtracking subpath structure; it will write down the entire information we described above. In order to store the subpaths we get from the backtracking test, we define a subpath head data structure, which storing the total subpaths and we can reach any of the subpath through this data structure.

3.3.2 Backtracking Testing

The backtracking testing program will take the output from the ASN.1 parser and the input from the user, create and store the dynamic backtracking state data structure as the testing goes on. When a subpath has been reached, the program will also store this subpath and link it to the previous one. After we finish the backtracking testing, we will create a subpath head data structure and make a pointer to the beginning of the subpath and a pointer to the end of the subpath; also, it will record the total numbers of subpaths.

3.4 Subtour Identification

Subtour identification is a list of the subpaths, which are generated from the backtracking testing program. Inside the subpath, we can see the total states being tested in this subtour. Also during a certain state, there is information about which input service primitive and input protocol data unit we get, which output service primitive and output protocol data unit we send.

In addition, we provide detailed information about each backtracking state for all subtours we generate.

For each state, we provide how many transitions which can be applied from there, the type and

the actual value for each variable in this backtracking state.

Also, we provide the frequency for using each variable, timer, constant in that time and the frequency in assigning variable, starting timer, stopping timer and resetting timer.

For each transition, we provide the name of the associated input service primitive, input protocol data unit, output service primitive, output protocol data unit, and the values of the service primitives for the above three data structures.

With these information, the user will fully understand the testing results.

3.5 Abstract TTCN Test Suite

In constructing a generic or abstract test suite, a test notation is used to describe abstract test cases. The test notation can be an informal notation or a formal description technique(FDT). TTCN is an informal notation with clearly defined, but not formally defined, semantics.

In the abstract testing methodology, a test suite is looked upon as a hierarchy ranging from the complete test suite, through test groups, test cases and test steps, down to test events. TTCN provides a naming structure to reflect the position of test cases in this hierarchy. It also provides the means of structuring test cases as a hierarchy of test steps culminating in test events. In TTCN, the basic test events are sending and receiving Abstract Service Primitive(ASPs), Protocol Data Units(PDUs) and timer events.

Two form of the notation are provided: a human readable tabular form, called $TTCN.GR$, for use in OSI conformance test suite standards; and a machine-processable form, called $TTCN.MP$, for use in representing TTCN in a canonical form within computer systems and as the syntax

to be used when transferring TTCN test cases between different computer systems. The two form are semantically equivalent.

Normally, an abstract test suite written in TTCN shall have the following four sections in the following order:

a) Suite Overview:

which is the information needed for the general presentation and understanding of the test suite, such as test references and a description of its overall purpose:

b) Declaration Part:

which is the set of components that comprise the test suite is described. This section shall contain the definition of any abbreviations to be used later in the test suite:

c) Constraint Part:

which is when the set set of values for the ASPs, PDUs, and their parameters used in the Dynamic Part. the constraint shall be specified using:

1. TTCN tables: or
2. the ASN.1 Modular Method; or
3. both TTCN tables and the ASN.1 Modular Method.

d) Dynamic Part:

which comprises three sections that contain tables specifying test behaviour expressed mainly in terms of the occurrence of ASPs at PCOs.

After getting the subtour identification and the subtour data structure, we can use them to construct the Abstract TTCN test suite.

Our abstract TTCN test suite will contain two parts, the modified TTCN dynamic part and the TTCN constraint table part.

3.5.1 Modified TTCN Dynamic Part

In this modified TTCN dynamic part, we build the behaviour description to describe the behaviour of the lower tester and/or upper tester in terms of test events using the tree notation. We also place the verdict or result information in the verdicts column which is associated with TTCN statements in a behaviour tree.

In the TTCN notation, each TTCN statement shall be shown on a separate statement line. Sequences of TTCN statements are represented one statement line after the other, each new TTCN statement being indented once from left to right.

Test TTCN statements at the same level of identification and belonging to the same predecessor node represent the possible alternative TTCN statements which may occur at that time. Alternative TTCN statements shall be given in the order in which the tester shall repeatedly attempt them until one occurs. Each behaviour description shall contain at least one behaviour tree.

3.5.2 TTCN constraint table part

It is necessary to specify in detail the values of ASP parameter and PDU fields. The encodings shall be described using either the Tabular Method or the ASN.1 Modular Method. Here we

use the first one.

In the TTCN tabular form, a constraint is defined by specifying a value and optional length for each PDU field. Each field entry in the field name column shall have been declared in the relevant ASP or PDU type declaration. When defining constraints on an ASP or PDU, values assigned to each field shall be of the type specified in the ASP or PDU declaration.

Chapter 4

Implementation

In this chapter, we discuss the implementation of the backtracking test kernel part and its environment.

The backtracking test generation part is implemented in the C and UNIX Sun 4 workstation environment.

4.1 The Environment

There are some advantages to achieving the implementation of the backtracking test kernel part.

First, The available ASN.1 parser and the protocol parser make it possible to get the entire static data structure (including the ASN.1 part), which has all the information for the backtracking testing part.

Also, the UNIX environment provides a better programming debugging, a huge C-program library, and almost all needed primitive functions. It is quite convenient to develop an entire program in this environment.

4.2 Backtracking Testing Kernel

The backtracking testing kernel part consists of nine modules; they are static data structure checking module, dynamic data structure module, check constraint module, expression and effect function module, initial module, backing module, subpath identification module, TTCN dynamic module, and TTCN constraint module.

4.2.1 Static Data Structure Checking Module

In order to make sure that each field of the static data structure completely correct, we need to make a check function to check them first before we use this information for our backtracking testing.

The checking function takes care of the following static data structure parts.

general information PDS part:

This part is the index part which points to the entire static data structure we build for the testing; also, it has the total numbers of states, transitions, variables and other parts of static data structures. We check whether the total numbers of states, transitions and the numbers of other data structure are among the maximum numbers we define for these data structures.

state part:

In the state part of the static data structure, we should check whether the key number of the state is among the state numbers which we define in the static data structure; the

nb_of_tr field is less than the total numbers of transitions we define in the static data structure, and the *tr_key*[*i*] field is among the *nb_of_tr* field we define for this state.

transition part:

In the transition part of the static data structure, we should check whether the key number of the state is among the transition numbers which we define in the static data structure; the *from_st* and *to_st* fileds are among the state numbers which we define in the static data structure, and the other fields are also among the information which we define in the static data structures.

constant part and variable part:

In the constant part and variable part of the static data structure, we should check whether the key number of both data structures are among the numbers we define for the constants and the variables; also we should check the type of both data structures among the three data types we defined before.

input, output service primitive part:

In the input and output service primitive part of the static data structure, we should check whether the key number of both data structures are among the numbers we define for them; also we should check the *nb_of_pdus* fields is less than the total number of pdus we define in this static data structure.

expression part:

In the expression part of the static data structure, we should check whether the key field is among the total expression numbers of the PPDS part, the *left_kind* field belongs to the KIND type we defined before, and the operator field belongs to the OPERATOR type we defined before.

effect function part:

In the effect function part of the static data structure, we should check whether the key number is among the total PDS effect function numbers, and the *stmt_kind* field belongs to the SKIND type which we defined before.

4.2.2 Dynamic Data Structure Module

As the backtracking goes on, this part creates and copies the dynamic data structure in order to record the whole protocol testing results. This module includes the following functions.

creat backtracking node submodule:

The creating backtracking node part has *new_bvar*(), *new_btimer*(), *new_bspam*(), *new_btrans*(), *new_bstate*(), *new_belt*() functions; these functions create a new backtracking node.

copy backtracking node submodule:

The copy backtracking node part has *copy_bvar*(), *copy_btimer*(), *copy_bspam*(), *copy_btrans*(), *copy_bstate*(), *copy_belt*() functions; those functions copy the current backtracking testing node to the current subpath and move to the next testing node after that.

subpath store and index creat submodule:

This submodule includes the functions to link the testing nodes, store the available subpathes, link the current subpath to the previous one, and finally create the index for the whole dynamic subpaths.

4.2.3 Check Constraint Module

When the user specifies the constraints for each part of the testing protocol, it limits the depth of the testing tree. Those constraints come to the static data structure part as the testing goes on. The check constraint module checks the constraints which the user specifies, to make sure that they do not pass the constraint limits.

The check constraint module function set includes the following functions: *check_state*(), *check_tran*(), *check_const*(), *check_isp*(), *check_osp*(), *check_pdu*(), *check_var*(), *check_assvar*(), *check_tstart*(), *check_tstop*(), *check_treset*(), *check_crest*(), *check_cstart*(), *check_cstop*(), *check_ctimer*(). They do the constraint checking for each part of the testing protocol.

4.2.4 Expression and Effect Function Module

Usually, there is some condition for the firing of a transition; it can simply be the constant expression, variable expression, primitive expression. It can also be the timer expression and complex expression.

Similarly, there is some effect action after the firing of a transition, it simply can be the assignment statement, timer statement. It can also be the if statement, and compound statement.

The expression and effect function module set include the following functions:

trans_expr_check(), *ifst_effn*(), *ast_effn*(), *tst_effn*, *cst_effn*(), *eff_trans*().

The *trans_expr_check*() function handles the condition predicate for each of the transitions, and the rest functions handle the effect action whenever a transition is fired.

4.2.5 Initiate Module

As the testing starts, there should be some initial values for the IUT, such as the initial state, initial values for variables, initial status for timer.

In order to do the protocol backtracking testing, we also need to copy parts of the static data structure values into the dynamic data structure, such as the entire initial input service primitives, output service primitives, protocol data units. All of these guarantee the consistency of the whole dynamic data structure with the static data structure.

4.2.6 Backtracking Module

The backtracking module is the kernel of the TESTGEN; it implements the backtracking algorithm we described before, finds the whole possible subpaths, and produces the entire testing results.

The backtracking module consists of a backtracking control part, a transition and state selection part, a backing part, a subpath storing part.

control part:

The control part controls the running of the entire backtracking testing process; the testing is finished if the Boolean expression in the control part can satisfy the following two requirements:

1. The current testing state is the initial state.

2. All the transitions in all testing states which can be fired have already been applied.

If the control part detects that the current testing process meets the above two requirements, it automatically stops the testing, and starts the subpath output modules. Otherwise, it continues running the backtracking module.

transition and state selection part:

The transition and state selection part handle the selection of a new testing state and a new applying transition in the current testing state. The new testing state is the destination state of the current transition. To select a new testing state, we need to know the current applying transition from the old testing state, and we also need to check whether the new testing state passes the constraint limits which the user specifies.

The new applying transition is the next applying transition in the current testing state. To select a new applying transition, we need to know the current testing state; also, we need to check whether the new applying transition passes the constraint limit which the user specifies.

After the transition is fired , we need to mark the firing of that transition in the current testing state. The function set of this module includes $Ntr\_find()$ function; it finds the next applying transition for the current testing state, checks the constraint limit for this transition, and decides whether this transition can be fired or not according to the return value from the expression checking function of this transition. Also, it includes some other

functions such as the *mark_trans*() function and so on.

backing part:

As the testing is in progress, we need to go back one step under one of the two circumstances. The new testing state we reach is the initial state, or the current testing state has no transition which can be applied.

When the testing process is in one of the two above situations, the current testing state goes back to the previous one, and all the transitions in the current testing state are unmarked.

subpath store part:

After building a subpath, we need to store it. Thus, the backtracking module calls the subpath store function to store the current subpath and link it to the previous one.

4.2.7 Subpath Identification Module

The subpath identification module outputs all the subpaths which generated from the backtracking testing kernel in a certain format.

The subpath identification module includes two functions *sub_print*() and *sel_print*().

The *sub_print*() outputs all the subpaths, generated from the testing kernel. Here is a simple subpath output example.

```
Specification id = TPO
```

```
yyparse() = 0

The subpathes number is 1

The subpathes are

 idle - TCREQ  /[CR]-> wfcc -[CC]/ TCCON-> data - TDATR  /-> data -/[DT]-> data\

 -[DT]/-> data -/-> data - TDREQ  / NDREQ->idle

 ***************************************


The subpathes number is 2

The subpathes are

 idle - TCREQ  /[CR]-> wfcc -[CC]/ TCCON-> data - TDATR  /-> data -/[DT]-> data\

 -[DT]/-> data -/-> data - NDIND  / TDIND->idle

 ***************************************


The subpathes number is 3

The subpathes are

 idle - TCREQ  /[CR]-> wfcc -[CC]/ TCCON-> data - TDATR  /-> data -/[DT]-> data\

 -[DT]/-> data -/-> data - NRIND  / TDIND->idle

 ***************************************


The subpathes number is 4

The subpathes are
```

```
 idle - TCREQ  /[CR]-> wfcc -[DR]/ NDREQ, TDIND->idle
 ************************************


The subpathes number is 5
The subpathes are
 idle - TCREQ  / TDIND->idle
 ************************************


The subpathes number is 6
The subpathes are
 idle -[CR]/ TCIND-> wftr - TCRES  /[CC]-> data - TDATR  /-> data -/[DT]-> data\
 -[DT]/-> data -/-> data - TDREQ  / NDREQ->idle
 ************************************


The subpathes number is 7
The subpathes are
 idle -[CR]/ TCIND-> wftr - TCRES  /[CC]-> data - TDATR  /-> data -/[DT]-> data\
 -[DT]/-> data -/-> data - NDIND  / TDIND->idle
 ************************************


The subpathes number is 8
```

```
The subpathes are
 idle -[CR]/ TCIND-> wftr - TCRES  /[CC]-> data - TDATR  /-> data -/[DT]-> data\
 -[DT]/-> data -/-> data - NRIND  / TDIND->idle
 ****************************************
```

The *sel_print*() prints the detailed information about some certain testing state in some subpathes. It prints until the current time, how many times that state, the whole variables, constants, and timers have been used in the backtracking testing. Also the values for the whole variables, timers, the names and the values of the all service primitives which have been used in this testing state. The user can select the testing state and the subpath number which he would like to get more detailed information about the testing results.

Here, we have a simple example of one testing state in one subpath of the tp0 testing result.

```
The subpath number is 2
The subpath are
 idle - TCREQ  /[CR]-> wfcc -[CC]/ TCCON-> data - TDATR  /-> data -/[DT]-> data -[DT]/·
ata -/-> data - NDIND  / TDIND->idle
 ****************************************

*****************************************************
```

```
This is the detailed information for the state 0

*****************************************************

The used of the state in the backtracking is 0


//////////////////  DETAILED   INFORMATION ///////////////



//////*********************************************///////

This is the information about the transition 0

//////*********************************************///////


****** The isp name of this transition is TCREQ *****

 TCREQ.qtsReq = TRUE

 TCREQ.fromAddr = 312

 TCREQ.toAddr = 316


**********output pdu one name of this transition is CR *******

 CR.sourceRef = 665

 CR.option =

 CR.callingAddr = 314

 CR.calledAddr = 225
```

```
 CR.maxTpdu = 1024


This is the information about the transition 1

//////*****************************************///////


****** The isp name of this transition is TCREQ *****

 TCREQ.qtsReq =

 TCREQ.fromAddr = 321

 TCREQ.toAddr = 324


********* osp 1 name of this transition is TDIND ********

 TDIND.tsDiscReason =

 TDIND.tsUserReason =


************** The constant information are ****************

//////////////*********************////////////////////////

The constant number is 0

The used for this constant is 1


////////////************************////////////////////////

************   The timer information are  *******************
```

```
//////////////*************************////////////////////////
The timer  key  number  is  0

The started times for this timer is 0

The stopped times for this timer is 0

The reseted times for this timer is 0

The check of the reseted for this timer is 0

The check of the started for this timer is 0

The check of the stopped for this timer is 0

The check of the timed out for this timer is 0
```

4.2.8 TTCN Dynamic Module

The TTCN dynamic part contains the main body of the test suite, it comprises three sections that contain tables specifying test behaviour expressed mainly in terms of the occurrence of ASPs at PCOs. it provides the information in a certain format, Behaviour Description, Label, Constraint Reference, Verdict, Comments.

Our modified TTCN dynamic output simplifiesn the format, and provide some important parts of them. Such as behaviour tree part, constraint reference part and the verdict part.

The TTCN dynamic module takes the subpathes from the testing kernel and produces the modified TTCN dynamic table. Here, we have a simple example, the two test cases of the FDDI.

```
-----------------------------------------------------------------------------

                        Test    Case    Dynamic     Behaviour

-----------------------------------------------------------------------------

    Identifier    Case 1

-----------------------------------------------------------------------------

 Behaviour  Description                Constraint Ref          Verdict

-----------------------------------------------------------------------------

 ! SmMaCtrlReq                      SmMaCtrlReq_I0

  ! PhUnitDataIndication                  PhUnitDataIndication_I0

   ? PhUnitDataRequest                  PhUnitDataRequest_O1

                                                                        pass
```

```
-----------------------------------------------------------------------------

                        Test    Case    Dynamic     Behaviour

-----------------------------------------------------------------------------

    Identifier    Case 2

-----------------------------------------------------------------------------

 Behaviour  Description                Constraint Ref          Verdict

-----------------------------------------------------------------------------

! SmMaCtrlReq                      SmMaCtrlReq_I2

  ! PhUnitDataIndication                  PhUnitDataIndication_I2

   ? PhUnitDataRequest                  PhUnitDataRequest_03

                                                                        pass
```

4.2.9 TTCN Constraint Module

It is necessary to specify, in detail, the values of ASP parameters and PDU fields. The TTCN Constraint Part specifies values for the ASPs, PDUs, and their parameters used in the Dynamic Part. And we use the TTCN constraint tables to specify these information.

The TTCN tabular form a constraint is defined by specifying a value for each PDU field. This information shall be provided in some certain format. Also, each field entry in the field name column shall have been declared in the relevant ASP or PDU declaration, values assigned to each field shall be of the type specified in the PDU (or ASP) declaration.

The TTCN constraint module takes the generated subpathes and the TTCN dynamic part as the input, and produce corresponding TTCN constraint tables.

Here, we have a simple example, part of the FDDI TTCN constraint table.

```
-----------------------------------------------------------------------------

                          ASP Constraint

-----------------------------------------------------------------------------


     ASP Name:  SmMaCtrlReq            Constraint Name: SmMaCtrlReq_I1
```

Field Name	Value	Comments
SmMaCtrlReq.ctrlAction	0	

ASP Constraint

ASP Name: PhUnitDataIndication Constraint Name: PhUnitDataIndication.

Field Name	Value	Comments
PhUnitDataIndication.phIndication		0

ASP Constraint

ASP Name: PhUnitDataRequest Constraint Name: PhUnitDataRequest_01

Field Name	Field Name	Value	Comments

```
-----------------------------------------------------------------------------

        PhUnitDataRequest.phRequest                    0


-----------------------------------------------------------------------------
```

Chapter 5

Test Case Validator and LAPB Validation

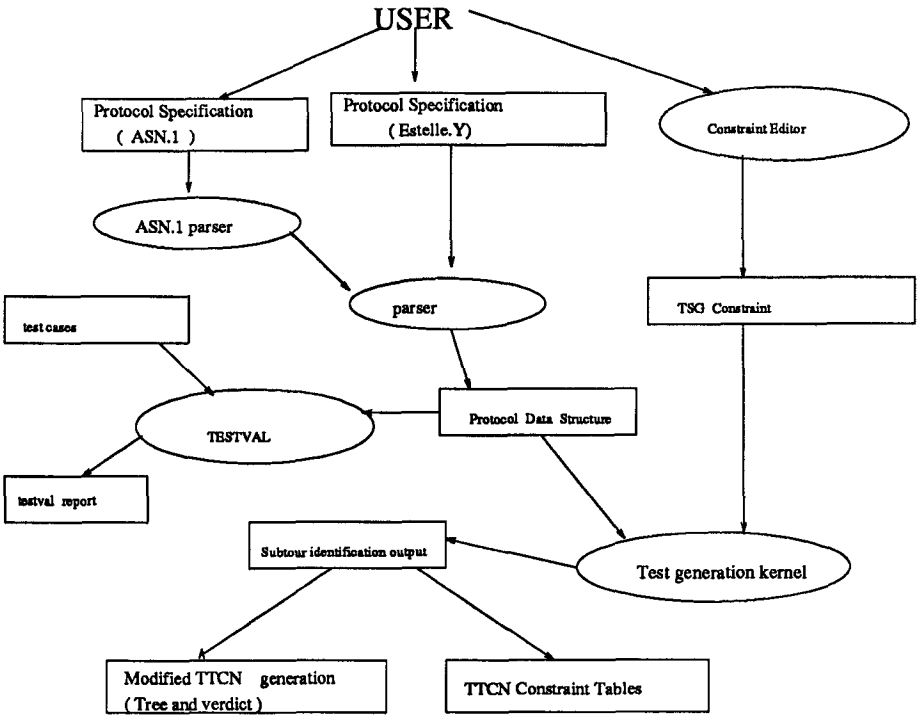

Figure 5.1: Components of TESTGEN+

In this chapter, I discuss the test case validator and the validation of the standardized LAPB test suite.

Test case generation and validation are two important topics in protocol testing research. Compared to test case generation, much less work has been done in the area of test case validation. As part of a Ph.D research conducted at the University of British Columbia, a trace analysis tool has been developed based on the protocol data structure (PDS) produced by the parser TESTGEN. This tool can handle any protocol specifications that can be mapped into an EFSM, and can be easily modified to serve as a test case validator, called TESTVAL.

TESTVAL can be incorporated into TESTGEN to produce an overal tool called *TESTGEN+* for protocol test generation and validation, as shown in Figure 5.1

5.1 Trace Analysis and Validator

5.1.1 Trace Analysis

Trace analysis is a very important part of protocol conformance testing. Compared to test case generation, trace analysis has the following main differences.

1. In test case generation, the initial and final states of each path are assumed to be known. Trace analysis is different; we need to determine the initial and final states from the observed input and output message sequences. Even for finite state machines which deal with control flow only, determining the initial state is not always possible. The condition is even more complicated for extended finite state machines which deal with both control and data flows.

2. In test case generation, every possible path of a formal specification has to be considered. Trace analysis, on the other hand, needs only to consider a finite number of paths that comply with the observed sequence of input and output messages.

3. The paths which have to be considered in trace analysis are finite in number and are a subset of those for test case generation. The trace contains information to determine the actual number of loop iterations, specific values of parameters, and the outcome of conditional statements. The absence of this information is largely responsible for the inefficiency in test case generation.

The trace analysis which was developed in the University of British Columbia is based on the extended finite state machine (EFSM), and the general steps are as following:
Given an Estelle specification and its implementation M.

1. Transform the Estelle specification into the single module Estelle specification.

2. Map from Estelle specification E to finite state machine F.

3. Perform trace analysis for conformance testing:

   a: Select from F the set of paths satisfying the input and output message in the case of trace analysis, and all the paths from the given initial state to the given final state with constraint for test case generation.

   b: Detect and delete the infeasible paths using symbolic evaluation if necessary.

   c: Assign verdict.

The implementation of the trace analysis consists of three parts:

Preprocessing Phase:

In this phase, some transitions of formal specification are preprocessed to be changed into finite state machine form.

Main Phase:

In order to make the implementation simple, fast and feasible without diminishing practicality, we assume that the trace analysis is started at the initial state. We analyze traces with respect to PDS of Estelle formal protocol specification. Data structure paths contain information such as data structures on a number of transitions and *to_state* of each transition with respect to the same *from_state*. Also, the current pointer of trace processing by trace analyzer is kept in order to process the right trace from the file.

Final Phase:

At *from_state* and *to_state*, these states contain an environment on all variables of a specification and all candidate transitions at that state. If there is a candiate transition, values of variables and service primitive from trace are substituted by a symbolic representation of PDS. If the predicate and the output primitives of a transition are satisfied with a trace, we can say that the trace conforms to the formal specification with respect to the trace.

Traces not conforming to the formal specifications are from the unsatisfaction of predicates, and the no matching of inputs and or outputs.

5.1.2 Validator

The trace analysis can also be used as a validator to validate the test cases. In order to handle various kinds of unexpected test cases, I made modifications to several parts of the trace analyzer to make it more general and practical for the users.

To perform a validation, we need two input files: the protocol Estelle specification and the actual test cases.

Here is an example for the test case input file, which is a simple test case of LAPB.

```
Case 1:

DataIndicat DISC A 1 - - - / DataRequest DM A 1 - - - / -

DataIndicat DISC A 1 - - - / DataRequest DM A 1 - - - / -

DataIndicat RR A 1 - 0 - / DataRequest DM A 1 - - - / -
```

Each test case consists of "*input*1(*or input*2)/*output*1/*output*2" where input1 and output1 are of lower interaction point of IUT and input2 and output2 are of upper interaction point. The input1 and output1 consist of "*interactionprimitive*", "*PDUtype*", "*address*", "*P/Fbit*", "*sendsequencenumber*", "*receivesequencenumber*" and "*datafields*".

5.2 LAPB

5.2.1 Introduction

X.25 is a set making up the international standard network access protocols for layer 1, 2 and 3. X.25 defined the interface between the host, called DTE(DATA Terminal Equipment) by CCITT, and the carrier's equipment, called a DCE(DATA Circuitterminating Equipment) by CCITT.

X.25 layer 1, the X.21 interface standard deals with the electrical, mechanical, procedural, and functional interface between DTE and DCE. It generally defines the physical interface between the DTE and DCE.

Layer 2, data link layer protocol, performs the link management and data transfer between the DTE and DCE, and ensures reliable communication between the DTE and the DCE.

Layer 3 manages connections between a pair of DTEs. Two forms of connection are provided, virtual calls and permanent virtual circuits.

The data link layer protocol used with X.25 is a version of the HDLC protocol called LAPB. The function of LAPB protocol is to provide the packet layer with an $error - free$ packet transport facility over the physical link between DTE and its local DCE.

As discussed, the physical interface between the DTE and the local DCE is defined in recommendation X.21. The datalink layer provides the packet layer with a reliable packet transport facility across the physical link between the DTE and the DCE. In the context of the OSI reference Model, the packet layer is the same as the network layer. The transport layer thus uses the services provided by the packet to enable it to exchange Transport PDUs with a remote

transport layer.

The frame structure, error and flow control procedures used by the link layer are based on the HDLC protocol. HDLC is a *link – level* protocol that has been defined by the ISO for use on both *point – to – point* and multipoint data links. It supports *full – duplex*, *transport – mode* operation and is now extensively used in both *terminal – based* networks and computer networks. It uses the ABM(asynchronous Balanced Mode) of operation, which is also referred to as LAPB in the CCITT X.25 standard document.

5.2.2 LAPB Estelle Specification

The LAPB was specified according to the *Packet – Switch* Protocol documentation. The Finite State Machine(FSM) structuring the LAPB protocol is described in the documentation.

Link Set Up

After DTE and DCE entered into the Initial State, The DCE sends the *"DM"* primitive to the DTE. It changes the state from the INITIAL to *SEND_DM* and starts the timer1. When timer1 is out, the DCE retransmits the *"DM"*. After N2 times of timeout, DCE sends a "*SABM*" to the DTE. It enters *SABM_SENT* state and starts timer1. DTE gets the *"DM"* N2 times but ignores them. Eventually, DTE gets *"SABM"* under INITIAL state; it sends a *"UA"* to response the DCE and changes its state from INITIAL to ABM. DCE will receive that *"UA"* response and change its state from *SEND_SABM* to ABM. At this moment, both DCE and DTE get in ABM state and be ready to transmit the information frames.

Information Frame Sending

When the LAPB entity module gets a *Data_Req* primitive from its user module, the LAPB entity will transmit the Iframe by setting its parameter variable ns equal to its window variable vs, and parameter variable nr equal to its window variable vr. After sending the Iframe, the vs is creased by 1 with mod k. This Iframe should be buffered in retxbuffer before it is acknowledged by its peer LAPB entity.

Information Frame Receiving

When an Iframe arrives, it is accepted if its parameter variable ns is equal to the window's vr. The window variable vr is the number of the Iframe the current LAPB waiting for. By accepting the Iframe, the vr is increased by 1 with mod k, and timer2 is started. The acknowledgement for this Iframe should not be sent at this moment; it will be piggybacked in the next outsending Iframe or be sent when timer2 is out.

Receiving Acknowledgement

There may be two kinds of acknowledgements: Iframe with $pf = 1$, or in retransmission a buffer should be released. The corresponding action in this specification is that retxprt(retransmission buffer pointer) increases by 1 with mod k and window variable nubuffered(number of buffered Iframe in retransmission buffer) decreased by 1.

Receiving Reject

The *REJ_cmd* can inform the receiving LAPB of the number of Iframe which the peer LAPB in remote station is waiting for.

Link Disconnect Condition

When the LAPB receives the *dis_req* from the upper layer, it sends *dis_cmd* to remote station and sets discmyself as true. When discmyself is true, the current LAPB entity module stops sending anything out, but can not refuse receiving coming messages until receiving *disc_resp*, and stops running LAPB protocol too. This disconnection procedure guarantees that no message is lost during the disconnection procedure.

5.2.3 LAPB Specification

PDU Encoding and Deconding

Every LAPB PDU contains the following items in the form of interaction parameters:

.address: the address of the station the message comes from, DCE or DTE in this program.

.control: type of the message; it can be INF for information frame, SUP for supervisory frame, and UNB for unnumbered frame.

.ns: for information frame, it tells the receiver that sequence number of the coming information frame. This ns is assigned to be window variable vs.

.nr: for information frame; it tells the receiver the acknowledgement from sender if pf is equal to 1. This ns assigned to be window variable vr. It also includes the acknowledgement when the *RR_cmd* is sent while pf is set.

.pf: as Iframe; when it is set, it indicates that the acknowledgement is piggybacked. As Uframe when pf is set, it indicates that the sender site is polling for an Iframe.

.udata: transmitted user data.

ASN.1 Specification

The ASN.1 specification part of the LAPB specification specifies the all filed data type of each ASP and PDU, and it will be inputted into the ASN.1 parser.

5.3 LAPB Validation and Result Analysis

Using this validator, I validated the whole LAPB test cases defined in the ISO 9646 standard document. In general, more than eighty percent of test cases passed our validation; the remainder could not pass due to the incompleteness of the LAPB specification used in the validation. Further details of the validation and the analysis are given in this section.

Validation

The whole LAPB test suite consists of around 300 test cases which are classified into seven phases.

DL1:

The DL1 is the Disconnected Phase; it is the state when IUT receives DISC and sends the UA or DM frame. There are 36 valuable test cases here. After the validation, 31 test cases passed which means that more than ninety percent of test cases passed the validation in DL1.

DL2:

The DL2 is the Link Disconnection phase; it is the state when IUT sends DISC frame. There are 31 valuable test cases here; 29 test cases passed the validation, which means that more than ninety three percent of test cases passed the validation in DL2.

DL3:

The DL3 is the Link Set up phase; it is the state when IUT sends SABM frame from Disconnected Phase. There are 29 valuable test cases here. After the validation, 28 test cases passed. This means that more than ninety five percent of test cases passed the validation in DL3.

DL4:

The DL4 is the Information Transfer phase; it is the state when IUT receives the SABM and sends the UA frame, or IUT sends the SABM and it receives the UA frame. There are 43 valuable test cases here, and 33 test cases passed the validation after the test. This means that around seventy percent of test cases passed the validation in DL4.

DL5:

The DL5 is the Frame reject condition phase; it is the state when IUT sends the FRMR frame from the Information Transfer phase. There are 40 valuable test cases here; 28 test cases passed the validation after the test. Thus seventy percent of test cases passed the validation in DL5.

DL6:

The DL6 is the IUT busy condition phase; it is the state when IUT sends the RNR frame from the Information Transfer phase. There are 32 valuable test cases here, and 20 test cases passed the validation after the test; sixty percent of test cases passed the validation in DL6.

DL7:

The DL7 is the Sent Reject condition phase; it is the state when IUT sends the REJ frame from the Information Transfer phase. There are 33 valuable test cases here, and 27 test cases passed this validation; it means that around eighty percent of test cases passed the validation in DL7.

From the seven phase testing results, we see that more than eighty percent of test cases here passed the validation. Clearly, this result is quite productive.

Analysis

From the above results, we are surprised to see that the total percentage rate with which the test cases passed this validation vary in different phases. This difference is mainly because of

the different limitations of this protocol specification among the seven phases. In the first three phases, we have a very complete LAPB specification. Most of the test cases which stand in the ISO 9646 standard conform to the specification we used here, a few of them could not pass. However, in the later phases, due to the incompleteness of the LAPB specifications, some test cases have been ignored, and there is a relatively lower passing rate.

After carefully analyzing these test cases, we find that some of the test cases couldn't be validated by this validator. There are several reasons:

– limitations of the specification of the LAPB protocol:

Due to the limitations of this LAPB specification, there are some test cases could not be included; for example, in the Disconnected phase, the $DL1\_102$ is supposed to verify that IUT sends a DM with F=0 in response to DISC P=0. The specifications here only have this situation with F=1, and P=1. The validator could not find any corresponding cases in the current LAPB version; this will result in the failure of this test case validation.

```
DL1_102

DataIndicat DISC A 0 - - - / DataRequest DM A 0 - - - / -
```

– limitations of the validator:

Due to the limitations of the validator, there are still some test cases which could not be validated. Let us look at some examples here.

In the DL4, there is a test case $DL4\_116$; it verifies that the IUT in the information transfer phase can manage its send window correctly. For the purposes of this test, the IUT shall transmit I frame that has a $N(S)$ value within the send window. Acknowledgements from the tester will rotate the send window for the DTE. The IUT window rotation shall be observed over the entire valid range of sequence numbers. The IUT shall stop the window rotation function when outstanding acknowledgements are not sent from the tester.

The test cases are obviously out of the input format we used here; thus, they can not be validated by this validator.

The redundant test case:

After carefully going through all the unsuccessful test cases, we find a few redundant test cases which may be omitted from a standard test suite.

Let us look at two test cases in the DL4, and at the information transfer phase in the LAPB. The point is to verify that in this phase, the IUT shall transmit an FRMR frame in response to a command frame with undefined or unimplemented control field. The C/R bit can be set to "0" or "1" and the W bit shall be set to "1" in the FRMR information field. In the ISO 9646 standard test cases, this test case was put into two separated test cases, which include the C/R both 0 and 1.

It definitely increases the number of unnecessary test cases, and produces some redundant test cases.

5.4 Conclusion

This chapter presents a test case validator, which is based on the finite state machine model and Estelle.Y specification.

After using it to validate the ISO 9646 LAPB test cases, a most satisfactory result has been produced.

Here, we can see that this validator is not only an efficient, general, flexible and semi-automatic tool for various kinds of protocol conformance test cases, but is also a very remarkable apporach to the protocol verification and validation, both theoretically and practically.

However, with the limitation of this validator, we couldn't test all possible test cases here; clearly, some improvement is still needed. For example, in one of the LAPB test cases, we could not validate the test case which has wrong information in the "checksum" field simply because we hadn't provided any input format to do so. In short, some improvement is still needed in future work.

Chapter 6

Conclusion and Future work

6.1 Summary

In this thesis, we discussed about the implementation of the TESTGEN and the test suite validation for some real protocol LAPB , TESTVAL, a test case validator based on an existing trace analyzer.

TESTGEN is intended to serve as a effective, general, flexible, portable, and user friendly environment for protocol generation.

However, TESTGEN is found to be unsuitable for test case validation. We have thus developed a test case validator called TESTVAL which is based on existing protocol trace analyzer and applied it to the standard LAPB test suite.

In general, we find TESTGEN+, including TESTGEN and TESTVAL, a general, useful and effective tool for protocol test generation and validation.

6.2 Future work

Despite the usefulness of TESTGEN+, There are still some areas which need further work to improve the tool.

. In TESTGEN, the number of data types supported for the ASN.1 data part representation, is limited to only a few types, thus limiting its applicability to real-life protocols. How to extend TESTGEN to support more data types is one of the major improvements in the future TESTGEN version.

. In practice, TESTGEN is for test case generation, and TESTVAL is for test case validation. It is useful to combine these two tools together to handle the entire test generation and validation. This is an ongoing piece of development work in the Protocol Engineering Group at UBC.

. Also, we need to improve the output format of the TTCN dynamic part and TTCN constraint part, as well as producing more complete and beautiful output format in general.

. Still, we need to improve the parser part and the test kernel part to adopt more and more real protocols for conformance testing.

. For TESTVAL, we also need to improve it further and to make it more general for validing test cases of any protocols.

Appendix A

LAPB Estelle Specification

```
Specification lapb;

CONST
       MAXCLOCK = 1000: int;
       MAXSEQ   = 7: int;
       DATASIZE = 1024: int;
       N2MAX    = 6: int;
       k        = 8: int;
       ZERO     = 0: int;
       ONE      = 1: int;
       NO       = false: boolean;
       YES      = true : boolean;

VAR
       datano,retxptr : int;
       busyremote : boolean;
       busymyself : boolean;
       discmyself : boolean;
       vs,vr : int;
       timer1on,timer2on : boolean;
       n2,nubuffered : int;
       data : int;

ISP
```

```
        UP        nsap;


OSP
      DOWN        nsap;


PDU

        SABM      recv_in  UP;
        DISC      recv_in  UP;
        UA        recv_in  UP;
        DM        recv_in  UP;
        I         recv_in  UP;
        FRMR      recv_in  UP;
        RRcmd     recv_in  UP;
        RRresp    recv_in  UP;
        RNRcmd    recv_in  UP;
        REJcmd    recv_in  UP;
        RNRresp   recv_in  UP;
        REJresp   recv_in  UP;
        BADcmd    recv_in  UP;
        BADresp   recv_in  UP;

Timer

      Timer1   2;
      Timer2   1;


STATE
      INITIAL, DMSend, SABMSend, ABM,SABMWait, UAWait;

TRANS
     FROM   INITIAL
       TO   DMSend
         WHEN       UP
         OUTPUT DM
```

```
    BEGIN
     timer1on := YES;
     n2 := n2 + ONE;
    END;


  TO   DMSend
    OUTPUT DM
    BEGIN
     n2 := n2 + ONE;
     END;


FROM   DMSend

   TO   DMSend
     WHEN        DISC
     OUTPUT  DM
     BEGIN
      timer1on := YES;
      n2 := n2 + ONE;
     END;

   TO   SABMSend
     WHEN        DM
     OUTPUT  SABM
     BEGIN
      timer1on := NO;
      n2 := ZERO;
     END;


   TO   SABMSend
     WHEN       FRMR
     OUTPUT  SABM
     BEGIN
      timer1on := NO;
      n2 := ZERO;
```

```
    END;


  TO   DMSend
    PROVIDED   ((n2 < N2MAX) and (timer1on = true) and Timeout(Timer1))
    OUTPUT  DM
    BEGIN
     n2 := n2 + ONE;
     END;

  TO   DMSend
    PROVIDED    ((n2 >= N2MAX) and (timer1on = true))
    OUTPUT  SABM
    BEGIN
     timer1on := NO;
     n2 := ZERO;
    END;


  TO   ABM
   WHEN        SABM
    OUTPUT UA
    BEGIN
     vs := ZERO;
     vr := ZERO;
     timer1on := NO;
     timer2on := NO;
     n2 := ZERO;
     nubuffered := ZERO;
    END;


FROM   SABMSend
  TO   SABMSend
    WHEN        SABM
    OUTPUT  UA
    BEGIN
      timer1on := YES;
```

```
    n2 := n2 + ONE;
  END;

TO   DMSend
  WHEN      DISC
  OUTPUT  DM
  BEGIN
   timer1on := NO;
   n2 := ZERO;
  END;

TO   ABM
  WHEN      UA
  BEGIN
   vs := ZERO;
   vr := ZERO;
   timer1on := NO;
   timer2on := NO;
   n2 := ZERO;
   nubuffered := ZERO;
  END;


TO   DMSend
  WHEN        I
  OUTPUT   DM
  BEGIN
    timer1on := NO;
    n2 := ZERO;
  END;


TO   DMSend
  WHEN         RRcmd
  OUTPUT   DM
  BEGIN
   timer1on := NO;
   n2 := ZERO;
```

```
  END;

TO   DMSend
  WHEN         RNRcmd
  OUTPUT   DM
  BEGIN
   timer1on := NO;
   n2 := ZERO;
  END;

TO   DMSend
  WHEN         REJcmd
  OUTPUT   DM
  BEGIN
    timer1on := NO;
    n2 := ZERO;
  END;

TO   DMSend
  WHEN         RRresp
  OUTPUT   DM
  BEGIN
   timer1on := NO;
   n2 := ZERO;
  END;

TO   DMSend
  WHEN         RNRresp
  OUTPUT   DM
  BEGIN
   timer1on := NO;
   n2 := ZERO;
  END;

TO   DMSend
  WHEN         REJresp
  OUTPUT   DM
  BEGIN
```

```
      timer1on := NO;
      n2 := ZERO;
    END;

  TO   DMSend
    WHEN       BADcmd
    OUTPUT   DM
    BEGIN
      timer1on := NO;
      n2 := ZERO;
    END;

  TO   DMSend
    WHEN       BADresp
    OUTPUT   DM
    BEGIN
      timer1on := NO;
      n2 := ZERO;
    END;

  TO   SABMSend
    PROVIDED    (timer1on = true )
    OUTPUT   SABM
    BEGIN
      timer1on := YES;
      n2 := n2 + ONE;
    END;


FROM   ABM

  TO   DMSend
    WHEN     DISC
    OUTPUT  UA
    BEGIN
      timer1on := NO;
      n2 := ZERO;
    END;
```

```
TO   DMSend
  WHEN     UA
  OUTPUT  DM
  BEGIN
   timer1on := NO;
   n2 := ZERO;
  END;

TO   ABM
  WHEN      RRcmd
  PROVIDED    ((busymyself = false) and (discmyself = false))
  OUTPUT   I
  BEGIN
   retxptr := (retxptr + 1) mod k;
   nubuffered := nubuffered + 1;
   vs := (vs + 1) mod k;
  END;

TO   ABM
  WHEN      RRcmd
  PROVIDED    ( busymyself = true )
   BEGIN
    nubuffered := nubuffered - 1;
    retxptr := (retxptr + 1) mod k;
   END;

TO   ABM
   WHEN     REJcmd
   OUTPUT  I;

TO   SABMSend
   WHEN       DM
   OUTPUT  SABM;

TO   UAWait
   WHEN       FRMR
   OUTPUT  SABM;
```

```
TO   SABMWait
   WHEN        BADcmd
   OUTPUT  FRMR;

TO   SABMWait
   WHEN        BADresp
   OUTPUT   DM;

TO   ABM
   WHEN       I
   PROVIDED    (busymyself = false)
   OUTPUT   I
   BEGIN
    vr := (vr + 1) mod k;
    timer2on := true;
    nubuffered := nubuffered - 1;
    vs := (vs + 1) mod k;
    nubuffered := nubuffered + 1;
    timer1on := YES;
   END;

TO   ABM
   WHEN       UP
   PROVIDED   ((busyremote = false) and (timer2on = false) and (nubuffered < M
   OUTPUT   I
   BEGIN
    vs := (vs + 1) mod k;
    nubuffered := nubuffered + 1;
    timer1on := YES;
   END;

TO   ABM
   PROVIDED    ((busymyself = false) and (timer2on = true) and Timeout(Timer2)
   OUTPUT   RRcmd
   BEGIN
    timer2on := NO;
   END;
```

```
    TO   ABM
      PROVIDED     ((busyremote = false) and (timer1on = true ) and (nubuffered < l
      OUTPUT   I
      BEGIN
       if (nubuffered <= k)
       then
        timer1on := YES
       else
        timer2on := NO;
      END;


    TO   ABM
       WHEN       I
       PROVIDED     (busymyself = false)
       OUTPUT   REJcmd;

    TO   ABM
       WHEN       I
       PROVIDED     (busymyself = true);

    TO   ABM
       WHEN       UP
       PROVIDED    (UP.hostbusy = true)
       OUTPUT   RNRresp
       BEGIN
        busymyself := YES;
       END;

FROM    SABMWait
  TO    ABM
    WHEN          SABM
    OUTPUT   UA
    BEGIN
     vs := ZERO;
     vr := ZERO;
     timer1on := NO;
```

```
   timer2on := NO;
   n2 := ZERO;
   nubuffered := ZERO;
  END;

TO    DMSend
  WHEN         DISC
  OUTPUT   UA
  BEGIN
   timer1on := NO;
   n2 := ZERO;
  END;

TO    DMSend
  WHEN         UA
  OUTPUT     DM
  BEGIN
   timer1on := NO;
   n2 := ZERO;
  END;

TO    DMSend
  WHEN        UA
  OUTPUT     SABM
  BEGIN
   timer1on := NO;
   n2 := ZERO;
  END;

TO    SABMWait
  WHEN         I
  OUTPUT   FRMR
  BEGIN
   timer1on := YES;
   n2 := n2 + ONE;
  END;

TO    DMSend
```

```
  WHEN       FRMR
  OUTPUT   DM
  BEGIN
   timer1on := NO;
   n2 := ZERO;
  END;

TO    SABMWait
  WHEN         RRcmd
  OUTPUT   FRMR;

TO    SABMWait
  WHEN         RNRcmd
  OUTPUT   FRMR;

TO    SABMWait
  WHEN         REJcmd
  OUTPUT     FRMR;

TO    SABMWait
  WHEN         BADcmd
  OUTPUT     FRMR
  BEGIN
   timer1on := YES;
   n2 := n2 + ONE;
  END;

TO   SABMWait
  PROVIDED     ((timer1on = true) and (n2 < N2MAX) and Timeout(Timer1))
  OUTPUT   FRMR
  BEGIN
   timer1on := YES;
   n2 := n2 + ONE;
  END;

TO   UAWait
  PROVIDED     ((timer1on = true) and (n2 >= N2MAX))
  OUTPUT   SABM
```

```
    BEGIN
     timer1on := NO;
     n2 := ZERO;
    END;

FROM   UAWait
  TO   UAWait
    WHEN      SABM
    OUTPUT   UA
    BEGIN
     timer1on := YES;
     n2 := n2 + ONE;
    END;

  TO   DMSend
    WHEN      DISC
    OUTPUT  DM
    BEGIN
     timer1on := NO;
     n2 := ZERO;
    END;

  TO   ABM
    WHEN       UA
    OUTPUT  DM
    BEGIN
     timer1on := NO;
     n2 := ZERO;
    END;

  TO   SABMSend
    WHEN        DM
    OUTPUT   SABM
    BEGIN
     timer1on := NO;
     n2 := ZERO;
    END;
```

```
        TO   UAWait
          PROVIDED     ((timer1on = true) and (n2 < N2MAX) and Timeout(Timer1))
          OUTPUT   SABM
          BEGIN
           timer1on := YES;
           n2 := n2 + ONE;
          END;

        TO   DMSend
          PROVIDED    ((timer1on = true) and (n2 >= N2MAX))
          OUTPUT   DM
          BEGIN
           timer1on := NO;
           n2 := ZERO;
          END;

END.
```

Appendix B

PART OF LAPB SUBTOUR IDENTIFICATION OUTPUT

```
The subpathes number is 1
The subpathes are
 DMSend -[DISC]/[DM]->DMSend
 ************************************

The subpathes number is 2
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[DISC]/[DM]->DMSend
 ************************************

The subpathes number is 3
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 SABMWait -[DISC]/[UA]->DMSend
 ************************************

The subpathes number is 4
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 SABMWait -[UA]/[DM]->DMSend
```

```
 ************************************

The subpathes number is 5
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 SABMWait -[UA]/[SABM]->DMSend
 ************************************

The subpathes number is 6
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 SABMWait -[I]/[FRMR]-> SABMWait -[FRMR]/[DM]->DMSend
 ************************************

The subpathes number is 7
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DISC]/[DM]->DMSend
 ************************************

The subpathes number is 8
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
 ABM -[SABM]/[UA]-> ABM -[DISC]/[UA]->DMSend
 ************************************

The subpathes number is 9
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
 ABM -[SABM]/[UA]-> ABM -[UA]/[DM]->DMSend
 ************************************
```

```
The subpathes number is 10
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
 ABM -[SABM]/[UA]-> ABM -[DM]/[SABM]-> SABMSend -[I]/[DM]->DMSend
 ***************************************

The subpathes number is 11
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
 ABM -[SABM]/[UA]-> ABM -[DM]/[SABM]-> SABMSend -[RRcmd]/[DM]->DMSend
 ***************************************

The subpathes number is 12
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
 ABM -[SABM]/[UA]-> ABM -[DM]/[SABM]-> SABMSend -[RNRcmd]/[DM]->DMSend
 ***************************************

The subpathes number is 13
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
 ABM -[SABM]/[UA]-> ABM -[DM]/[SABM]-> SABMSend -[REJcmd]/[DM]->DMSend
 ***************************************

The subpathes number is 14
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
```

```
 ABM -[SABM]/[UA]-> ABM -[DM]/[SABM]-> SABMSend -[RRresp]/[DM]->DMSend
 ************************************

The subpathes number is 15
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
 ABM -[SABM]/[UA]-> ABM -[DM]/[SABM]-> SABMSend -[RNRresp]/[DM]->DMSend
 ************************************

The subpathes number is 16
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
 ABM -[SABM]/[UA]-> ABM -[DM]/[SABM]-> SABMSend -[REJresp]/[DM]->DMSend
 ************************************

The subpathes number is 17
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
 ABM -[SABM]/[UA]-> ABM -[DM]/[SABM]-> SABMSend -[BADcmd]/[DM]->DMSend
 ************************************

The subpathes number is 18
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]-> ABM -[RNRresp]/->
 ABM -[SABM]/[UA]-> ABM -[DM]/[SABM]-> SABMSend -[BADresp]/[DM]->DMSend
 ************************************

The subpathes number is 19
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
```

```
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
DMSend
 ***************************************

The subpathes number is 20
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[BADcmd]/[FRMR]-> SABMWait -[DISC]/[UA]->DMSend
 ***************************************

The subpathes number is 21
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[BADcmd]/[FRMR]-> SABMWait -[UA]/[DM]->DMSend
 ***************************************

The subpathes number is 22
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[BADcmd]/[FRMR]-> SABMWait -[UA]/[SABM]->DMSend
 ***************************************

The subpathes number is 23
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
```

```
 ABM -[BADcmd]/[FRMR]-> SABMWait -[I]/[FRMR]-> SABMWait -[FRMR]/[DM]->DMSend
 ************************************

The subpathes number is 24
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[DM]/[SABM]-> SABMSend -[I]/[DM]->DMSend
 ************************************

The subpathes number is 25
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[DM]/[SABM]-> SABMSend -[RRcmd]/[DM]->DMSend
 ************************************

The subpathes number is 26
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[DM]/[SABM]-> SABMSend -[RNRcmd]/[DM]->DMSend
 ************************************

The subpathes number is 27
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[DM]/[SABM]-> SABMSend -[REJcmd]/[DM]->DMSend
 ************************************
```

```
The subpathes number is 28
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[DM]/[SABM]-> SABMSend -[RRresp]/[DM]->DMSend
 ***************************************

The subpathes number is 29
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[DM]/[SABM]-> SABMSend -[RNRresp]/[DM]->DMSend
 ***************************************

The subpathes number is 30
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[DM]/[SABM]-> SABMSend -[REJresp]/[DM]->DMSend
 ***************************************

The subpathes number is 31
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[DM]/[SABM]-> SABMSend -[BADcmd]/[DM]->DMSend
 ***************************************

The subpathes number is 32
```

```
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[DM]/[SABM]-> SABMSend -[BADresp]/[DM]->DMSend
 ***************************************

The subpathes number is 33
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[REJcmd]/[I]-> ABM -[RNRresp]/-> ABM -[RRresp]/->
 ABM -[DISC]/[UA]->DMSend
 ***************************************

The subpathes number is 34
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[REJcmd]/[I]-> ABM -[RNRresp]/-> ABM -[RRresp]/->
 ABM -[UA]/[DM]->DMSend
 ***************************************

The subpathes number is 35
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
DMSend
 ***************************************
```

```
The subpathes number is 36
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
DMSend

 ************************************

The subpathes number is 37
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADcmd]/[FRMR]->
 ABM -[FRMR]/[SABM]-> UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADcmd]/[FRMR]->
 ABM -[FRMR]/[SABM]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 SABMSend -[UA]/-> ABM -[I]/-> ABM - UP  /[I]->
 SABMWait -[UA]/[DM]->DMSend
 ************************************

The subpathes number is 38
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-:
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADcmd]/[FRMR]->
 ABM -[FRMR]/[SABM]-> UAWait -[SABM]/[UA]-> UAWait -[UA]/[DM]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADcmd]/[FRMR]->
 ABM -[FRMR]/[SABM]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 SABMSend -[UA]/-> ABM -[I]/-> ABM - UP  /[I]->
 SABMWait -[UA]/[SABM]->DMSend
 ************************************
```

```
The subpathes number is 39
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[DM]/[SABM]-> SABMSend -[SABM]/[UA]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADresp]/[DM]->
 ABM -[I]/-> ABM - UP  /[I]-> ABM -[BADcmd]/[FRMR]->
 ABM -[FRMR]/[SABM]-> UAWait -[SABM]/[UA]-> UAWait -[SABM]/[UA]->
DMSend
 ***************************************

The subpathes number is 40
The subpathes are
 DMSend -[DM]/[SABM]-> SABMSend -[SABM]/[UA]-> SABMSend -[UA]/-> ABM - UP  /[RNRresp]-
 ABM - UP  /[I]-> ABM -[BADresp]/[DM]-> SABMWait -[SABM]/[UA]->
 UAWait -[SABM]/[UA]-> UAWait -[SABM]/[UA]-> UAWait -[DISC]/[DM]->DMSend
 ***************************************
```

Appendix C

LAPB test cases and validation results for DL1

```
DL1_101:
DataIndicat DISC A 1 - - - / DataRequest DM A 1 - - - / -
pass

DL1_102:
DataIndicat DISC A 0 - - - / DataRequest DM A 0 - - - / -
failure

DL1_103:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
pass

DL1_104:
DataIndicat SABM A 0 - - - / DataRequest UA A 0 - - - / -
DataIndicat SABM A 0 - - - / DataRequest DM A 0 - - - / -
failure

DL1_201A:
DataIndicat DM A 1 - - - / DataRequest SABM A 1 - - - / -
failure

DL1_201B:
```

```
DataIndicat DM A 1 2 0 9 / Discard - - - - - - / -
pass

DL1_202:
DataIndicat SABM E 1 - - - / Discard - - - - - - / -
pass

DL1_203:
DataIndicat SABM A 1 9 0 5 / Discard - - - - - - / -
pass

DL1_204:
DataIndicat DM A 1 0 0 0 / Discard - - - - - - / -
pass

DL1_205:
DataIndicat DM A 1 9 0 9 / Discard - - - - - - / -
pass

DL1_207:
DataIndicat SABM A 1 9 0 9 / Discard - - - - - - / -
pass

DL1_208:
DataIndicat UA A 1 9 0 9 / Discard - - - - - - / -
pass

DL1_209:
DataIndicat RR A 1 9 0 9 / Discard - - - - - - / -
pass

DL1_210:
DataIndicat BNR A 1 9 0 9 / Discard - - - - - - / -
pass

DL1_211:
DataIndicat REJ A 1 9 0 9 / Discard - - - - - - / -
pass
```

```
DL1_215:
DataIndicat I E E E 0 E / Discard - - - - - - / -
pass

DL1_216:
DataIndicat DISC A 1 9 0 9 / Discard - - - - - - / -
pass

DL1_301:
DataIndicat I A 1 - - - / DataRequest DM A 1 - - - / -
pass

DL1_302:
DataIndicat RR A 1 - - - / DataRequest DM A 1 - - - / -
failure

DL1_303:
DataIndicat RNR A 1 - - - / DataRequest DM A 1 - - - / -
pass

DL1_304:
DataIndicat REJ A 1 - - - / DataRequest DM A 1 - - - / -
pass

DL1_305:
DataIndicat UA A 0 - - - / Discard - - - - - - / -
pass

DL1_306:
DataIndicat UA A 1 - - - / Discard - - - - - - / -
pass

DL1_307:
DataIndicat FRMR A 0 - - - / Discard - - - - - - / -
pass

DL1_308:
```

```
DataIndicat FRMR A 1 - - - / Discard - - - - - - / -
pass

DL1_309:
DataIndicat I A 0 - - - / Discard - - - - - - / -
pass

DL1_310:
DataIndicat RR A 0 - - - / Discard - - - - - - / -
pass

DL1_312:
DataIndicat REJ A 0 - - - / Discard - - - - - - / -
pass

DL1_313:
DataIndicat RR A 1 - - - / Discard - - - - - - / -
pass

DL1_314:
DataIndicat RNR A 1 - - - / Discard - - - - - - / -
pass

DL1_315:
DataIndicat REJ A 1 - - - / Discard - - - - - - / -
pass

DL1_316:
DataIndicat RR A 0 - - - / Discard - - - - - - / -
pass

DL1_317:
DataIndicat RNR A 0 - - - / Discard - - - - - - / -
pass

DL1_318:
DataIndicat REJ A 0 - - - / Discard - - - - - - / -
pass
```

```
DL1_319:
DataIndicat I A 0 - - - / Discard - - - - - - / -
pass
```

Appendix D

LAPB test cases and validation results for DL2

```
DL2_101:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat DISC A 0 - - - / DataRequest UA A 0 - - - / -
pass

DL2_102:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat DISC A 1 - - - / DataRequest UA A 1 - - - / -
failure

DL2_105:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat SABM A 0 - - - / DataRequest DM A 0 - - - / -
pass

DL2_106:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DisReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat SABM A 1 - - - / DataRequest DM A 1 - - - / -
```

```
pass

DL2_109:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat DM A 1 - - - / Discard - - - - - - / -
pass

DL2_110:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat DM A 1 - - - / Discard - - - - - - / -
pass

DL2_111:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat UA A 1 - - - / DiscInd
pass

DL2_201A:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat BAD E 0 - - - / Discard - - - - - - / -
pass

DL2_201B:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat DM A 1 - - - / Discard - - - - - - / -
pass

DL2_205:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat DISC A 1 9 0 9 / Discard - - - - - - / -
pass
```

```
DL2_207:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat SABM A 1 9 0 9 / Discard - - - - - - / -
pass

DL2_209:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat UA A 0 9 0 9 / Discard - - - - - - / -
pass

DL2_211:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat DM A 0 9 0 9 / Discard - - - - - - / -
pass

DL2_219:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat I A F H H H / Discard - - - - - - / -
pass

DL2_221:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat RR B 1 - - - / Discard - - - - - - / -
pass

DL2_223:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat BNR B 1 - - - / Discard - - - - - - / -
pass

DL2_225:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
```

```
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat REJ B 1 - - - / Discard - - - - - - / -
pass

DL2_227:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat DISC F 0 - - - / Discard - - - - - - / -
pass

DL2_229:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat DISC A 0 F F F / Discard - - - - - - / -
pass

DL2_231:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / dataRequest DISC B 1 - - - / -
DataIndicat DM A 0 - - - / Discard - - - - - - / -
pass

DL2_234:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat UA A 1 - - - / Discard - - - - - - / -
pass

DL2_306:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / conInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat RR A 1 - - - / Disccard - - - - - - / -
pass

DL2_308:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat RR A 0 - - - / Discard - - - - - - / -
```

```
pass

DL2_310:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat RNR A 1 - - - / Discard - - - - - - / -
pass

DL2_312:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat RNR A 0 - - - / Discard - - - - - - / -
pass

DL2_314:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat REJ A 1 - - - / Discard - - - - - - / -
pass

DL2_316:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat REJ A 0 - - - / Discard - - - - - - / -
pass

DL2_318:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
Dataindicat FRMR A 0 - - - / Discard - - - - - - / -
pass

DL2_332:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat FRMR A 1 - - - / Discard - - - - - - / -
pass
```

```
DL2_334:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat I A 0 0 0 0 / Discard - - - - - - / -
pass

DL2_336:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat I A 1 0 0 0 / Discard - - - - - - / -
pass

DL2_338:
DataIndicat SABM A 1 - - - / DataRequest UA A 1 - - - / ConInd
DiscReq - - - - - - / DataRequest DISC B 1 - - - / -
DataIndicat I A 0 - - - / Discard - - - - - - / -
pass
```

Bibliography

[1] W.Y.L.Chan,S. T. Vuong, and M. R. Ito,"An improved Protocol Test Generation Procedure Based on UIOs " *Proceedings of the ACM SIGCOMM '89 Symposium on Communciation Architecture and Protocols*, 1989.

[2] J.P. Wu and S. T. Chanson,"Test Sequence Derivation Based on External Behaviour Expression" *2nd Internal Workshop on Protocol Test System* , Germany, 1989.

[3] ISO Transport Protocol Specification *ISO DP 8073*, 1984.

[4] International Organization for Stansardization"The Tree and Tabular Combined Notation " ISO 9646 , 1989.

[5] Information Processing system, "Open System Interconnection -Specification of Abstract Syntax Notation One" *ISO 8824*, September, 1987

[6] Information Processing System, "Open System Interaction - Estelle- A Formal Description Technique Based on an Extended State Transition Model," *IS 9074*, 1989.

[7] Roelof J. Velthuys, "Protocol Conformance Testing with Communciation Rule System," IBM European Networking Center, 1989.

[8] ISO DIS 9646 " OSI Conformance Testing Methodology and Framework", March, 1989.

[9] M. Sample and G. Neufeld, "Support for ASN.1 within a Protocol Testing Enviroment", *The third International Conformance on Formal description Techniques*, Madrid Spain, November, 1990.

[10] H. Janssen, Y. Lu and P. Zhou, "Definition of a Protocol Data Structure Representation for Communciation Protocols", *UBC Technical Report planned for summer*, 1991

[11] JingLu "Lapb Estelle specification", 1990

[12] Russel Vuong "Lapb Estelle specification", 1989

[13] U. Bar, J.M.Schneider, "Automated Validation of TTCN Test Suites" *IBM European Networking Centre*

[14] Kshirasagar Naik, Behect Sarikaya, "Verification of Protocol Conformance Test Cases Using Reachability Analysis" *Dept. of Electrical and Computer Eng. Concordia University, Dept. of Computer Science and Operation Research, University of Montreal*

[15] M.C.Kim, Samuel T. Chanson and Son T. Vuong "Protocol Trace Analysis based on Formal Specification" *Department of Computer Science, University of British Columbia*