FORMAL SPECIFICATION BASED CONFORMANCE TESTING

Behçet Sarikaya

Concordia University
Dept. of Electrical Engineering, 1455 de Maisonneuve W.
Montreal, Quebec, Canada H3G 1M8

Gregor v. Bochmann, Michel Maksud, Jean-Marc Serre

Université de Montréal
Dépt. d' I. R. O., C.P. 6128 Succ. A
Montreal, Quebec, Canada H3C 3J7

ABSTRACT

The paper discusses the use of formal specifications for conformance testing of OSI protocols and divides the discussion in two parts: test design and tester design. A draft standard formal specification of the Class 4 transport protocol in Estelle is taken as the starting point for test design. The test design technique used to derive a conformance test suite is semi-formal based in part on the formal specification and also the informal specification. The tests obtained are expressed in natural language. In the tester design part, we discuss the distributed test architecture of ISO and give the detailed designs of upper and lower testers. It is shown that, in testing Class 4 transport implementations, a parametrized protocol implementation approach in the lower tester design, renders the tests easier to implement.

1. INTRODUCTION

Due to wide spread acceptance of International Standards Organization (ISO)-Open Systems Interworking (OSI) protocols [DaZi 83], testing the implementations of these protocols has become an important activity. By testing a protocol implementation it is meant that a set of tests, called a **test suite** is applied to the implementation with the aim of determining whether the implementation conforms to the relevant OSI protocol standard. ISO subcommittee that deals with conformance testing for OSI has recently developed a methodology which defines various architectures for protocol testing [Rayn 85]. After defining the methodology and framework for specifying conformance test suites, it is hoped that test suites will be adopted for each of the OSI protocols.

Formal description techniques have now matured to be used in formally defining the OSI protocols and in various other related activities. There exist two techniques developed by ISO ([Linn 86], [Brin 86]): Estelle based on an extended finite-state machine model and Lotos based on a calculus of communicating systems. Estelle specifications of several OSI protocols have been developed, compilers exist of earlier dialects and implementations of OSI protocols are quite easy to obtain using them. Lotos is expected to catch up with Estelle in this respect in a near future.

Acknowledgements. This research has been supported by the Department of Communications of Canada under contract DSS File Number 4ER.36100-5-0149.

It is desirable to use formal specifications of protocols for conformance testing, since formal specifications do not possess the deficiencies of natural language specifications, i.e., being ambiguous and imprecise. It can be expected that formal specifications will eventually replace todays so-called protocol standards.

This paper focuses on the use of formal specifications of protocols as the basis of implementation and testing activity. In particular, we consider the transport protocol service class 4 [IS 8073] and its specifications in Estelle [NBS 85]. The next section discusses the method we used in deriving tests and Section 3 overviews the conformance test suite for class 4 transport protocol (TPCL4). Tester design is discussed in Section 4 and test implementation in Section 5. Section 6 contains the authors' conclusions on this research activity.

2. TEST DESIGN

The ideal approach to test design would be to use as reference the formal specification in Estelle [NBS 85] and apply to it the test design methodology developed previously [Sari 84]. This section gives an overview of the methodology and explains why the forseen approach was not feasible and discusses the method used instead.

The test design methodology is based on identifying the functions of a protocol from the formal specification and then deriving test sequences to test each such function. The formal specification analysis is done by symbolically executing the specification. The methodology has manually been applied to transport protocol classes 0 and 2 [Sari 84].

The formal specification of TPCL4 [NBS 85] is much more complex than TPCL2 since its structure was conceived to include all classes of the protocol and also Class 4 is a complex protocol in itself. The number of various local procedures/ functions, internal variables, channels, spontaneous transitions, etc. are such that it is extremely difficult to mentally follow the behavior of the specification and apply the symbolic execution on it without automated tools.

The specification [NBS 85] contained various syntactic and semantic errors. These errors were revealed by a limited manual application of symbolic execution [SaBoSe 86]. Later, the errors were considered for correction and new versions of the specification appeared which has invalidated the partial analysis performed since these changes not only affected the value assignments to internal variables of the protocol, but also modified the major state transitions. In short, the formal specification's use has been limited to be a starting point for both implementation [Serr 86] and test design.

The test design is based on the informal specification [IS 8073], although some information derived from the formal specification is used to supplement it. This information includes primarily the control graph (finite-state machine) of TPCL4 which is used to derive the transition subtours [SaBo 84], and the data flow graphs of some protocol functions used to determine the ordering of the tests considering their functional dependencies. Transition subtours give the normal sequence of primitives to be applied to verify a given function. Other possible events should be considered during test specification as discussed in Section 5. From the informal specification, functions of TPCL4 are identified. Each test is designed to verify that the implementation conforms to the conditions listed in the specification for that function. It seems difficult to completely automate this test design approach. Test design experience for similar protocols is an important factor in this test design process.

3. TEST SUITE

A number of tests designed to verify the conformance of a protocol implementation to the protocol standard is called a **test suite.** In this section we discuss a test suite for TPCL4. The tests are designed to be applied in a distributed single layer architecture which is detailed in Section 4. This architecture has the components of the upper tester (UT) which is a service user of the implementation and the lower tester (LT) situated in the test center and acting as a peer protocol entity.

3.1. Classification of the Tests

From the informal specification of TPCL4 we have identified the following functions to be verified by the test suite:

- -Connection establishment and release,
- -Connection refusal,
- -Error release of connection,
- -Expedited data transfer with its flow control,
- -Data transfer, including:
  - -Segmenting and reassembling,
  - -Concatenation and separation,
  - -DT TPDU numbering,
  - -Flow control,
  - -Retention and retransmission,
  - -Credit reduction,
  - -AK TPDU retransmission,
  - -Mixed data and expedited data transfer,

- -Multiplexing and splitting,
- -Network failure,
- -Robustness, including:
  - -Unexpected peer stimulations,
  - -Unexpected user stimulations.

Based on these functions, the test suite is divided into 9 groups of tests: basic test, connection establishment, connection refusal, error release of connection, expedited data transfer, data transfer, multiplexing/ splitting, network failures and robustness. Some of these groups are discussed below.

3.2. Basic Test

This is the first test that has to be applied to the implementation. Its purpose is to verify that a transport connection with minimal connection parameters can be established and that a fixed length data message can be transfered in both directions (in one direction at a time).

3.3. Connection Establishment Tests

The tests in this group verify correct handling of the parameters of the CR primitive to be received by the implementation. The parameters of interest are: class and alternative class, maximum TPDU size, extended format option, checksum, expedited data option, credit, user data. Each of these parameters is enumerated in a test whenever possible exhaustively. A different value is tried in a consecutive connection. In the case of user data only the length parameter is enumerated exhaustively.

This group contains a test for the acceptance of retransmitted CR, whose purpose is to verify if the implementation accepts repeated identical CR TPDUs, which would occur when the peer protocol timer has elapsed.

There is a number of UT initiated tests in the connection establishment test group. The UT initiated basic connection establishment test tries to establish a simple transport connection originated from the UT. Other tests try to force the implementation to enumerate various parameters of the CR primitive to be received by the LT such as expedited data option, alternative class, maximum TPDU size, checksum. These tests are more restricted than the LT initiated tests since UT is a service user of the implementation and the TCONrequest service primitive parameters can control the corresponding TPDU parameters only indirectly. One of the UT initiated tests checks that the implementation does not refuse a connection in which the CC credit parameter specifies a different value for each consecutive connection.

3.4. Data Transfer Test Group

The data transfer test group contains a basic data transfer test, an UT originated data transfer test, two-way simultaneous data transfer test, and the tests for flow control, AK retransmission, credit reduction and data transfer timers. A test for mixed data expedited data transfer is also included. Some of the tests will be explained below.

3.4.1. Two-way Simultaneous Data Transfer Test

The purpose of the test is to exchange data messages in both directions at the same time. After connection establishment, LT starts sending DT TPDUs of length one octet and increasing by one octet each time up to a certain maximum. Various TSDU lengths are tried using the eot parameter of the DT, thereby testing the reaction of the IUT to fragmentation. Consecutive DT TPDUs are sent without waiting for an AK TPDU unless the window is closed. LT keeps a retransmission timer for every DT sent but not acknowledged. The UT receives the transport service data units (TSDU) and verifies that the data was not modified. At the same time, UT starts sending TSDUs (again of length one octet and increasing by one each time). Similarly, LT receives corresponding DT TPDUs and verifies their content and checksum and acknowledges them by sending an AK TPDU for each correctly received TPDU. At the end, UT sends an error report to the LT.

During this test the implementation could try to establish several simultaneous network connections to provide a better throughput by using the **splitting mechanism** of TPCL4.

3.4.2. Test for Data Transfer Timers

This test is designed to verify if the implementation retransmits unacknowledged DT TPDUs. LT establishes a connection and sends a DT TPDU containing 1 octet of data. UT should receive a T_DATA indication containing that data. Thereafter, UT sends a fixed number of TSDUs. LT receives the corresponding DTs, but does not acknowledge the DTs. After the implementation sends all DTs in its send window, it should stop sending new DTs. After W seconds (defined as window timer in [IS 8073]) from the sending of the first DT, the implementation should retransmit the DT. The implementation should continue retransmitting other DTs until the window is closed. Then the LT sends an AK TPDU acknowledging retransmitted DTs and including credit for more DT TPDUs to be sent. The test stops when UT sends all the data.

3.4.3. Mixed Data Expedited Data Transfer Test

The purpose of this test is to verify the correct transfer of TSDUs and Expedited data (ED) concurrently and the implementation's capability to stop the normal data transfer until an outstanding acknowledgement for an ED is received. After the connection is established, LT sends consecutive DT TPDUs without waiting for AK TPDUs until the window is closed. UT closes the flow control, i.e. it does not accept the T_DATA indications. When the window is closed the LT sends an ED TPDU and should receive its acknowledgement, i.e., the EA TPDU. LT sends a second ED and should receive a second EA.

UT does not receive TSDUs until the first T_EXDATA_indication arrives. After it receives the second expedited data, UT starts sending T_DATA_requests. When the sequence wrap around is achieved (assuming normal sequence numbers) the UT sends two consecutive T_EXDATA_indications. The implementation should send an ED TPDU, stop the normal data flow and wait for an EA. LT acknowledges the ED after having received all normal data previously sent by the IUT. The test is finished when the second ED is acknowledged and all normal data are transfered.

4. TESTER DESIGN

The distributed single-layer test architecture of ISO [Rayn 85] provides the most complete form of single-layer testing. In the case of the transport layer, the upper tester (UT) part of the architecture applies transport service primitives defined in [IS 8072] to the implementation under test (IUT) and the lower tester (LT) applies TPDUs over a network layer connection to the IUT. In this section we discuss the detailed designs of UT and LT.

4.1. Upper Tester

The upper tester uses the services provided by the IUT which are dependent on the programming environment in which the IUT runs. Although [IS 8072] defines the transport service primitives and their temporal ordering, their implementation details are left to the implementation. Given the fact that the tests described in Section 3 will be applied to various implementations, it is desirable to implement the upper tester part of the tests in an implementation independent manner. One way of achieving this independence is to create an extra network connection and design a simple responding unit which will send the responses of the IUT to the LT and receive and apply the next input to IUT. In this way, the complicated part of the UT is kept in the test center [ZeRa 86], [RaCaCh 86]. We have found this approach not feasible for testing TPCL4 implementations since the extra network connection can not be assumed to be reliable, i.e., the transport protocol class 4 is to be used over networks which are inherently unreliable.

In our design the independence is achieved by giving an implementation specification for the tests in Estelle. Each test consists of two specifications, one of which describes the UT behavior. We define a minimal transport service and use this service in all the test specifications. Adaptation of this minimal service to a given local service interface is done in a module of the UT called **permanent part.** Figure 3.1 shows the architecture of our upper tester design.

The permanent part (referred in what follows as UTP) is a program started at the beginning of the test session and runs until the last test is completed. It has the following components:

1. The **kernel** consists of the common region which is used to store the implementation parameters such as the number of multiplexed connections supported, the addresses, etc., and service conversion routines.
2. The test management protocol module. The module executes the test protocol which enables the LT to dictate which is the next test to be applied and to load the parameter region with updated values. This protocol uses the transport class 2 protocol and includes an initial basic test for the transport protocol class 2 implementation in the IUT.

The test instance module referred in what follows as UTI is the executable code obtained from the Estelle specification of a test and some runtime routines. UTI's executable code is different for each test, but a single runtime routine package is used. The runtime routine package contains two sets of procedures: interprocess communication routines to communicate with UTP and utility routines provided by the Estelle translator.

The test_interface channel in Fig.3.1 is used to send the common region to UTI and to send a status report at the end of the test to UTP. The TS_primitives channel is used to exchange a representation of the service primitives between the UTP and UTI (in the form of Pascal record generated by the Estelle translator). The transport service channel is the transport service access point (TSAP) provided by the IUT. UTP converts the minimal service primitives of UTI to the service primitives supported locally and vice versa.

4.2. Lower Tester

The lower tester is the active part of the test system. It runs on the test center computer, therefore transportability is not an important issue. LT establishes network connections with the IUT and sends/ receives transport protocol data units (TPDU). A possible design is by implementing each test in a conventional procedural language, e.g. Pascal with some support routines for encoding/ decoding of PDUs, trace generation, etc. This approach has been taken in the transport protocol class 0 tests [BoCeMaSa 83]. With this approach, the test program usually contains a large part of the protocol tested. For simple protocols, such as TPCL0, this fact does not create large test programs, while for complicated protocols, such as TPCL4, test programs tend to be huge. This was the case when we tried to implement the *basic test* of Section 3 in this manner.

Another approach to LT design is to use a Reference Implementation (RI) of the protocol under test [LiNi 83]. The reference implementation is assumed to be tested independently and hence, assumed to be correct. The test program can use the services of RI simplifying many of its tasks. The LT implementation in [LiNi 83] contains an exception generator unit in order to introduce invalid PDUs. Although the exception generator increases the testing capabilities of the reference implementation approach, there are other behaviors needed in the RI which are different from the normal behavior of an ordinary implementation. The test suite described in Section 3 contains many examples of these behaviors, some of which will be discussed below. This leads us to a third approach to LT design: the use of a **parametrized protocol implementation** (PPI).

The architecture of our lower tester design with the parametrized implementation approach appears in Figure 3.2. The Parametrized Protocol Implementation (PPI) is a full, parametrized implementation of TPCL4 where most of the implementation decisions (e.g. CR options, use of splitting, acknowledgement scheme, etc.) are controlled by global variables that can be modified by the **test module.** The test module (TM) is different for each test and is responsible for setting the PPI in proper decision mode, as required by the test. TM interacts with PPI by using transport service primitives with additional parameters to increase observability. These parameters are used to transmit the parameters of the received TPDU. For example, the augmented T_CONNECT_conf primitive contains all parameters of the received CC.

4.2.1. Functionalities of PPI

Some of the behaviors needed in the PPI by the test suite of Section 3 are different from the behavior of an ordinary implementation. We call PPI **functionalities** these different behaviors which can be selected through appropriate choices of the implementation parameters. Some of these functionalities are the following:

1. Repeat CC (even if an AK TPDU received) until IUT disconnects. The default behavior of implementation is disabled to implement this functionality.
2. Do credit reduction after "x" DT TPDUs are received. The "x" is a parameter specified by the TM (default behavior is disabled as above).
3. Do not acknowledge some of the DTs, in order to see if the IUT retransmits them.

4. Maximum multiplexing on a single network connection.
5 Maximum splitting of each transport connection.
6. Concatenation, 3 different schemes: (a) none, (b) medium (wait for a small delay before sending TPDU), and (c) maximum concatenation used only in two-way traffic (TPDUs will not be sent before concatenation is made). The default behavior is (a).
7. Do nothing on a particular network connection (while splitting is used). This functionality is used in inactivity timer tests. The default behavior is disabled.

4.2.2. Interface between TM and PPI

The get/set channel in Fig. 3.2 provides the setting of the functionality required by TM and the reading of the **event indicators.** The TM uses a **set** interaction to initialize the values of the global parameter variables created to select functionalities, and a **get** interaction to read the values of the event indicators and the global variables. Corresponding to each functionality described above, there is a global variable to enable it. There are other global variables defined to facilitate test specification, for example a structure containing default values of the outgoing CR parameters, maximum number of retransmissions (global for all TPDUs), timer values and trace options.

The event indicators are set by the PPI and read by the TM. Some of these indicators define the following:

- Incoming CR parameters, a structure similar to the one used for outgoing CR.
- Whether the transport connection was closed normally at the end of the test.
- Number of retransmissions received for each type of TPDU.
- Current state of receive and send windows (open, closed, reduced).
- Whether a protocol error was detected (the actual error should be visible from the trace file).

5. TEST SPECIFICATION

A dialect of Estelle is used to formally specify the tests. Each individual test consists of two Estelle specifications, one describing the UT behavior, the other describing the LT behavior.

The upper tester specification includes the definitions of the channels (TS_primitives for service primitive definitions, and test_interface to report end of test status), variables and local procedures/ functions used in the specification and finally the transitions to define the primitives to apply to the IUT and to receive/ verify its responses. The transitions are obtained from the subtour used for test design (see Section 2). Since the IUT can issue a T_DISCONNECT_indication anytime, there should exist some transitions to handle this situation.

The lower tester specification of the tests becomes much simpler in the parametrized reference implementation of Section 4. Apart from the local variables, global variables and event indicators are defined. The common region (see Section 4) is defined and updated by the lower tester. Individual tests can read values from the common region and modify it according to the test results (for example whether or not the IUT is an answer-only implementation, etc.). The specification of a test module includes a channel definition (TS_primitives to exchange augmented transport service primitives), local procedures/ functions and transitions. The global variables are set in the initial transition specified by the **initialize** keyword of Estelle. The transitions are obtained from the subtour of the test. Additional transitions should be foreseen to handle other possible events, such as the IUT disconnecting unexpectedly or the underlying network issuing a network reset.

6. CONCLUSIONS

A conformance test suite developed for a complicated protocol, i.e., the transport protocol class 4, is described. The tests are obtained in a semi-formal manner by using both formal and informal specifications of the protocol.

Various design possibilities for the distributed single-layer test architecture are considered to implement the test suite. Upper tester design is made by considering transportability and the upper tester part of the tests are specified formally in a formal specification language. Lower tester design is made to facilitate test specification. In this case, a parametrized protocol implementation approach seems to be the best design, since most of the control required by the tests can be accommodated and the tests are easy to specify (a test specification assumes a given parametrized transport protocol specification).

We have used a translator of a formal specification language to implement most of the components of the tester and all the tests. It would be useful to have some tools to automatically generate tests possibly based on the methodology of [Sari 84]. We believe that the tester design and the test suite design technique can easily be adapted to test the implementations of the protocols of Session and Presentation layer. The application layer protocols require a different test approach since there is often no direct access to the application service primitives.

7. REFERENCES

[BoCeMaSa 83] G.v. Bochmann, E. Cerny, M. Maksud, B. Sarikaya, "Testing Transport Protocol Implementations", Proc. of CIPS, Ottawa, Canada, pp.123-129, 1983.

[Brin 86] E. Brinksma, "A Tutorial on Lotos", Proc. of 5th Workshop on Protocols, M. Diaz(Ed.), North-Holland, 1986.

[DaZi 83] J. Day, H. Zimmerman, "The OSI Reference Model", Proceedings of the IEEE, Vol. 21, No. 12, 1983.

[IS 8072] Information Processing System - Open Systems Interconnection- Connection Oriented Transport Service Specification International Standard, 1985.

[IS 8073] Information Processing System- Open Systems Interconnection- Connection Oriented Transport Protocol Specification, International Standard, 1985.

[Linn 86] R.J. Linn, "The Features and Facilities of Estelle", Proc. of 5th Workshop on Protocols, M. Diaz(Ed.), North-Holland, 1986.

[LiNi 83] R. J. Linn, S. Nightingale, "Some Experience with Testing Tools for OSI Protocol Implementations" Proc. of 3rd Workshop on Protocols, H. Rudin, C. West (Eds), 1983, pp. 521-531.

[NBS 85]NBS, "Formal Description of the IS 8073 Transport Protocol", May 1985.

[RaCaCh 86] O. Rafiq, R. Castanet, C. Chraibi, "Towards an Environment for Testing OSI Protocols", Proc. of 5th Workshop on Protocols, M. Diaz(Ed.), 1986.

[Rayn 85] D. Rayner, "Standardizing Conformance Testing for OSI", Proc. of COMNET'85, Budapest, Hungary, October 1985, pp.7.1-7.20.

[SaBo 84] B. Sarikaya, G.v. Bochmann, "Synchronization and Specification Issues in Protocol Testing", IEEE Trans. on Comm., April 1984, pp. 389-395.

[SaBoSe 86] B. Sarikaya, G.v. Bochmann, J-M. Serre, "A Method of Validating Formal Specifications", Research Report, Concordia Univ., Montreal, Canada, Jan. 1986.

[Sari 84] B. Sarikaya, "Test Design for Computer Network Protocols", PhD. Thesis, McGill Univ., Montreal, Canada, March 1984.

[Serr 86] J-M. Serre, "Implementation of the Transport Protocol Class 4", Doc. de Travail, Univ. de Montreal, Montreal, Canada, April 1986.

[ZeRa 86] H. X. Zeng, D. Rayner, "The Impact of the Ferry Concept on Protocol Testing", Proc. of 5th Workshop on Protocols. M. Diaz(Ed.), 1986.

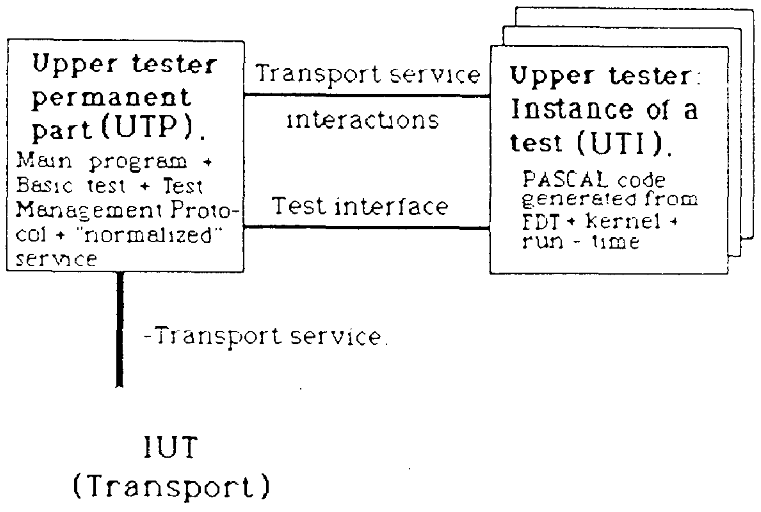
Upper tester permanent part (UTP).
Main program + Basic test + Test Management Protocol + "normalized" service
Transport service
interactions
Test interface
Upper tester: Instance of a test (UTI).
PASCAL code generated from FDT + kernel + run - time
-Transport service.
IUT (Transport)

Figure 3.1. Architecture of the Upper Tester

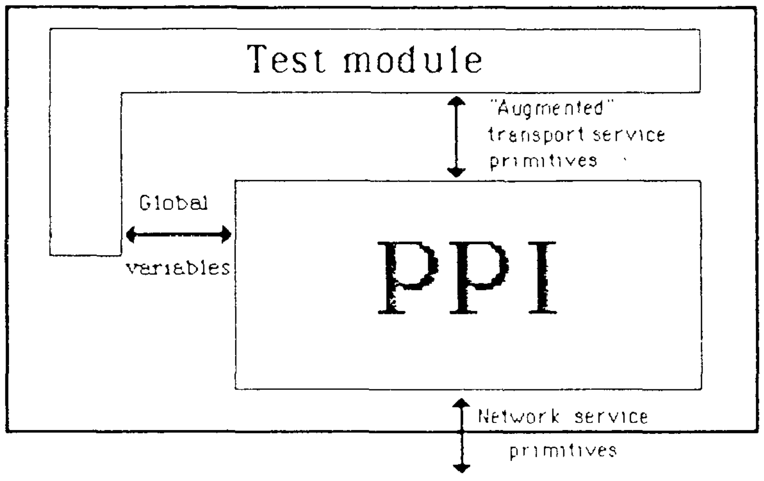

Figure 3.2. Architecture of Lower Tester