RECENT DEVELOPMENTS IN PROTOCOL TESTING

Behçet Sarikaya

Concordia University

Electrical and Computer Engineering Department

1455 de Maisonneuve W.

Montréal, Québec, Canada H3G 1M8

(invited paper)

ABSTRACT

With wide-spread acceptance of ISO-OSI reference model and its standardized protocols in the areas of computer communication and information exchange, protocol testing has become an area of active research and development. This paper attempts to survey recent developments in this area. First, an overview of basic notions regarding OSI model, formal specification, local/ conformance testing of implementations and test sequence generation is given. This is followed by a discussion on three important aspects of protocol testing: test architectures, sequences and theory. Test architectures are defined, observability and error detection limitations are addressed. Coverage of test sequence generation is linked to formal specification techniques. Test derivation from Estelle, Lotos, Prolog and finite-state machine models are reviewed. Specifying resulting tests with TTCN, a recently defined notation, is discussed with some examples. Theory part considers two stream-lines of research activity, one which tries to relate the notions of reduction, extension, refinement and tests to formal specification; the other which sees the testing activity from the point of observable behavior consisting of traces of interaction sequences.

1. INTRODUCTION

Open Systems Interconnection (OSI) model developed by International Standards Organization (ISO) is best known with its layered architecture which is a distributed replica of a design technique called *onion skin*. Historically onion skin approach was used in designing centralized systems as a way of simplifying system architecture by modelling inner components (kernel, etc.) as abstract machines. Since OSI model is intended to provide applications distributed over heterogenous systems, it has a major ingredient, called the concept of *protocol* which does not exist in the onion skin approach [Pouz 86].

A system willing to interwork (defined as an **open system**) has to implement the 7-layers of the OSI model, forming one slice of the global system. A complicated application at layer 7 is only possible by sharing the communication and other tasks all over the 7 layers. Each layer communicates with its peer layer, but physical communication takes place only in layer 1, the physical layer. Rules governing information exchange between peer layers at layer N define an (N)-protocol. Since open systems can be developed by different people, protocols have to be standardized. An (N)-protocol's task is to ease the task of layer (N+1) at some abstraction level by providing *(N)-service*. Thus the entity at layer (N+1) will use this (N)-service to communicate with the peer (N+1) entity using (N+1)-protocol and will provide an (N+1)-service to the layer above and so on as shown in Fig. 1. Since different layers in a given open system can be developed by different people, services also have to be standardized. ISO has so far managed to produce natural language definitions of protocols of all layers and services of most of the layers along with the standard definition of the OSI model [ISO 84].

The rest of the paper is organized as follows:
Section 2 is on protocol specification and testing, Section 3 is on protocol

test architectures, Section 4 is on test sequences, Section 5 is on theory and Section 6 contains author's conclusions.

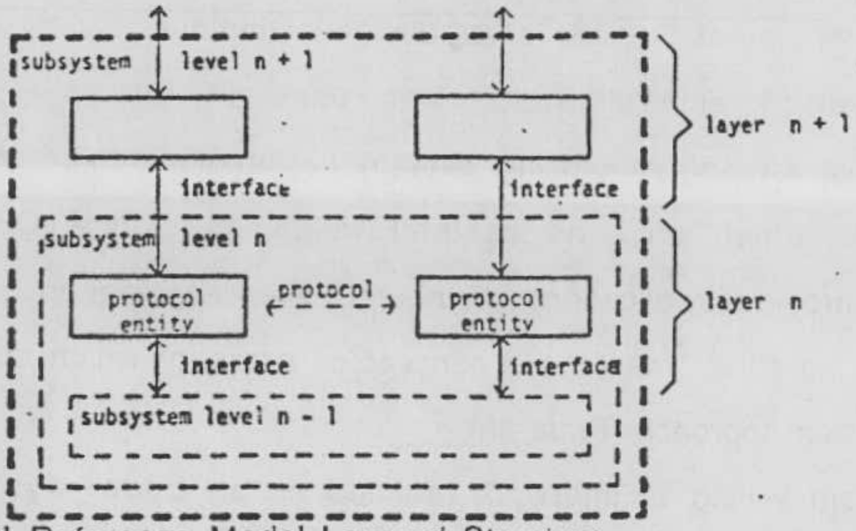


Figure 1. OSI Reference Model Layered Structure

2. PROTOCOL SPECIFICATION AND TESTING

Since natural language definitions of protocols/ services suffer from various deficiencies such as ambiguity and difficulty with automated treatment, formal description techniques (FDT) have been developed and are being standardized. There are two FDTs developed by ISO: Estelle and Lotos and one by CCITT: SDL. Since SDL is very similar to Estelle (except its graphical part) and that it is expected to converge with Estelle, we will not treat SDL in the rest of the paper. Both Estelle [Este 85] and Lotos [Loto 86] can be seen as languages for defining *transition systems*, but their underlying model and syntax are completely different.

A protocol/ service specification has three components: interaction primitives, control, and data. A service specification contains a definition of the service primitives and their parameters (fields) while a protocol specification contains definitions of (N+1)- and (N)-service primitives as well as the protocol data units, or (N)-PDUs.

Control component deals with sequencing service primitives and/ or protocol data units. For connection-oriented protocols we have *phases* of operation, i.e. connection establishment, data transfer and disconnection. Simplest transition systems known as finite-state machines (FSM) have widely been used in modelling the control component of protocols/ services [Boch 78, Dant 80].

Data component is the processing of interaction primitive parameters. Negotiation of various connection establishment options, data transfer mechanisms such as error processing, acknowledgement schemes, flow control, etc. are all dealt with by data component.

2.1 FDTs

Lotos is an algebraic specification language based on a calculus of communicating systems (CCS of Milner [Miln 80]) which is used to define dynamic (observable) behavior of processes. For the specification of data as well as interaction primitives, Lotos is complemented with a powerful abstract data type language called ACT ONE [EhMa 85]. ACT ONE is used to define interaction primitives and static data; control component is basically expressed as a process algebra in terms of a set of operations including (recursive) procedure calls. Data component is specified with the use of abstract data type operations in processes and by process functionalities, i.e., processes which accept/ pass data before/ after execution. Lotos primitive interaction type is rendez-vous, i.e., syncronous. Rather than simple value passing, Lotos processes can agree on a common value when rendez-vous is achieved.

Estelle is a procedural language based on an extended finite-state machine model. Service primitives are declared as part of channel definitions using a Pascal like syntax. PDUs need not be defined explicitly since they are sent using (N-1) service primitives. The control component is specified as an FSM: each module has a reserved major state variable and dynamic part of the modules comprise of transitions with FROM and TO clauses to specify state changes. Data component is specified as global types/ variables some of which can be abstract data types. Estelle primitive interaction type is queued communication which applies to communication between layer entities as well as intermodule communication.

2.2 Implementation

OSI protocols are presently implemented by a combination of hardware/ software with hardware in layers 1 and 2 and hardware and software in layer 3 and software in layers 4 and above. One advantage of formal specifications of protocols is that protocol implementations can be semi-automatically obtained [SeCeBo 86]. There exists various translators of Estelle which translate a given Estelle specification into some conventional languages such as Pascal and C [VuLa 87]. Implementing protocol systems from Lotos is also desirable but has not yet been accomplished.

Protocol implementations can be tested by considering a single-layer or multi-layer entity as a whole and stimulating the entity from the layers above and below and observing the reactions of the implementation under test (IUT). Stimulation/ observation is done by sending/ receiving (N)-service primitives, also called (N)-Abstract Service Primitives (ASP) and (N-i)-ASPs for an i-layer entity by an entity called Tester. Service access points (SAP) used by the tester for this purpose are called points of control and observation (PCO). When IUT and Tester reside on the same machine, the testing is local, otherwise it is remote.

2.3 Conformance Testing

In order to achieve the goals of OSI, it is necessary that (N)-protocol implementations in each open system conform to the protocol standard. By protocol standard, natural language definitions produced by ISO/ CCITT are usually understood; while the ultimate goal is to consider the formal specifications in one of the standardized languages. The testing activity done for the purpose of checking the capabilities and behavior of an IUT against the conformance requirements given in the protocol standard is defined as conformance testing. The aim is to increase the probability that different implementations are able to interwork. A standard document is being prepared to define a framework (in terms of architectures, definitions and procedures to be followed during testing) to conformance testing [ISO 86].

Testing an IUT involves applying one or more test cases (or equivalently, test sequences), a complete set of actions required to achieve a specific test purpose. Conformance testing is performed by executing a test suite which is defined as a set of test groups together with an ordering information. Test groups are a set of related test cases such as connection establishment, normal data transfer, multiplexing/ splitting. A test case is made up of a number of test events, an indivisible unit of test such as sending or receiving a single PDU [ISO 86].

3. TEST ARCHITECTURES

In this section we discuss the local and remote test architectures and their error detection limitations.

3.1 Error Detection

In any testing activity, it is assumed that the tester is able to judge correctly the results of observation of the IUT's behavior. The entity capable of making this kind of judgement is called a test oracle be it a program or a human being. In protocol testing, the tester which applies the test suite is the entity which decides on the conformance of IUT to the protocol specification. If a given behavoir is not in conformance to the specification, an error is detected. Error detection capability of protocol testers depends on test architecture, test suite and the design of testers [Dsso 86].

3.2 Local Test Architecture

In a local test architecture (L) IUT is stimulated directly by ASPs from the layer above and below (see Fig. 3.1). The points of observation and control (PCO) are (N)- and (N-1)-SAPs. Note that in this architecture, the tester should have the capability of acting as a peer entity, at least in some of the test cases. Due to the fact that the tester has full control and observation of all the external access points of an IUT, the local test architecture has a complete error detection capability. The only restriction in this architecture is the limitations of any testing, i.e., it is impossible to make a complete test of all cases.

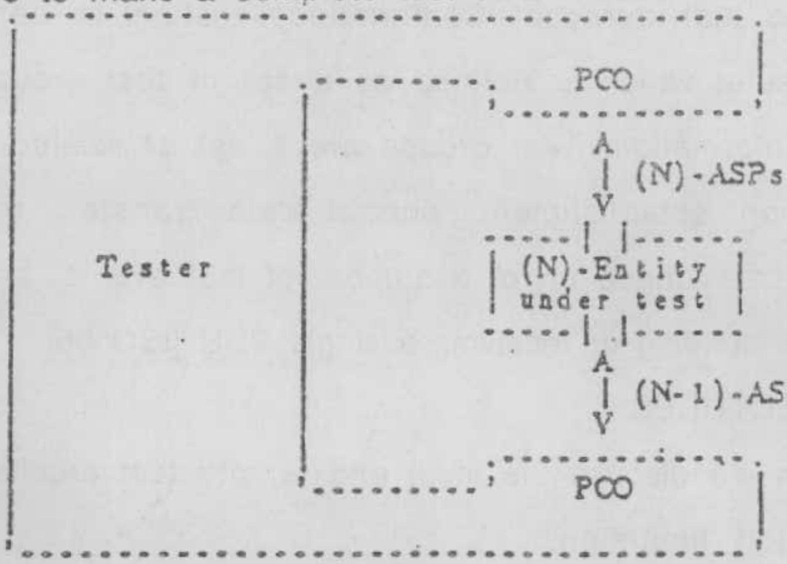


Figure 3.1. Local Test Architecture

3.3 Remote Test Architectures

The need for remote test architectures arises due to the fact that conformance testing can be the responsibility of a national/ international institution that undertakes the activity from a centralized site. At the implementation site, it is also possible that more than one machine is involved in testing the IUT.

In remote test architectures, stimulation and observation is distributed to local and remote sites. The testing entity on the remote site is called Lower Tester (LT). LT controls and observes (N-1)-ASP"s on the remote system. Locally there is no PCO on the (N-1) service boundary but direct or indirect control and observation of (N)-ASPs are assumed.

In the distributed (D) test architecture (see Fig. 3.2) direct control and observation of (N)-ASPs are done with a testing entity called Upper Tester (UT). Since realization of service primitives is implementation dependent, upper tester should be designed in such a way that it should be easily portable [SaBoMaSe 86]. In architecture D a given test case is applied by both LT and UT.

It has been shown that some test cases may result in **synchronization problems** between LT and UT [SaBo 84]. The synchronization problem arises when, in a given test case, LT (UT) should send an ASP/ PDU while in previous step LT (UT) had not received/ sent any ASP/ PDU.

There is a need to coordinate the actions of the two testing entities in order to achieve the test's aims. The means to achive it is called the **test coordination procedures** There seems to be various ways of designing the test coordination: defining a protocol (**test management protocol**) between LT and UT (see the achitecture C below) [Rayn 82]; integrating the coordination in individual tests which are executed by UT and LT [BoCeMaSa 83]; astride responder architecture where UT design is simplified by creating an extra connection and implementing the UT functions on the LT site [Rafi 85].

Error detection power of the architecture D is reduced compared with the local architecture since (N-1)-ASPs are controlled and observed from a distance over (N-1)-service of the network. Certain inputs (Network reset) are impossible to apply. Compared with other remote architectures, this architecture has improved error detection due to the existence of UT.

The coordinated (C) test architecture (see Fig. 3.2) is nothing but the architecture D in which test coordination procedures are implemented as a test management protocol.

The remote (R) test architecture assumes that it is not possible to observe and control (N)-ASPs of the IUT which happens when the IUT is located in a file server with limited capabilities, etc.. Test cases are defined in terms of (N-1)-ASP"s and (N)-PDUs. Since there is a need to have a limited upper tester functionality on the IUT site, they have to be provided by the system under test, possibly by a human operator. Architecture R offers the most limited error detection power because of the fact that upper boundary of the IUT is not observed/ controlled. On the other hand, it is considerably easier to do the tests by an entity called **arbiter**. The arbiter passively observes the interactions between the LT and the IUT. In a broadcast environment (a local area network) the arbiter reads the PDUs exchanged [MoDiAy 85], otherwise arbiter creates (N)-connections with both LT and IUT and then it becomes transparent to these two entities [Zhao 86].

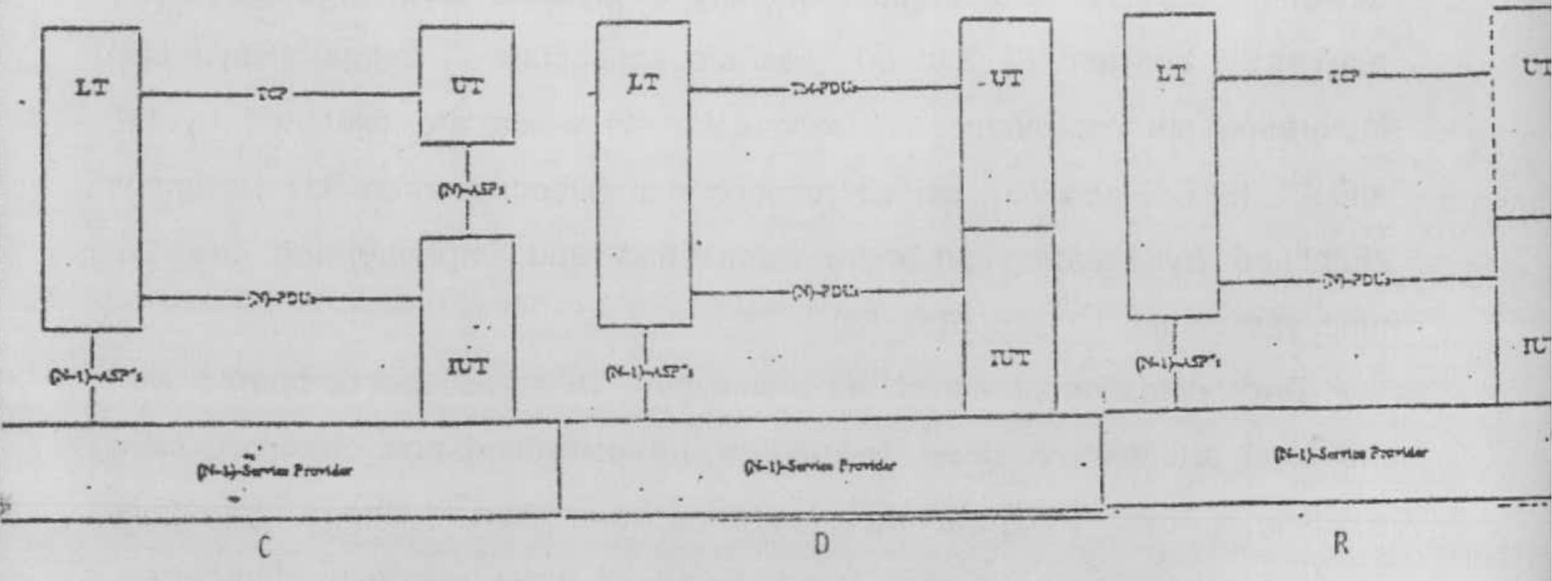


Figure 3.2. Remote Test Architectures

4. TEST SUITE DESIGN

Designing test cases (suites) can be considered to be the most active research area of protocol testing. Research in this area is inspired from the rich results obtained previously in hardware/ software testing, nevertheless, it is evolving towards its own set of techniques, tools and disciplines, possibly due to the distinct characteristics of protocols and their test architectures.

We will discuss the protocol test design in four categories: FSM, Estelle and Lotos based techniques and hybrid techniques. Finally in this section we discuss formal specification of the resulting tests.

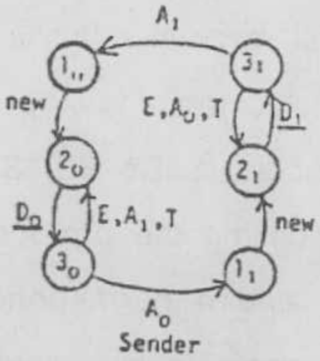
4.1 FSM Based Test Design

Early work on checking experiments on sequential machines [Koha 78] and FSM models of protocols give rise naturally to combining the two techniques together in order to derive tests for protocols [SaBo 82, SaBo 84]. The best results are obtained when the FSM modelling the protocol is modified such that all the transitions have an input and a corresponding output as labels. Test sequence derivation techniques come in two classes: transition tour which simply includes all the transitions defined at least once; methods which require that FSM possess a special sequence/ set of interactions such as characterization sets [Chow 78], distinguishing sequences [Koha 78], unique input/ output sequences [SaDa 85]. Since protocol FSMs are usually incompletely specified, it becomes more difficult to find these special sequences or sets.

It is possible to make an automated tool to obtain test sequences using one or more of the FSM techniques. Such a tool containing two methods of transition tour (random transition selection and depth-first search) and the characterization sequence techniques is presented in [Wan 87]. The tool also contains modifications of these techniques so as to generate synchronizable sequences. An example use of this tool is shown in Figure 4.1 where FSM model of the simplest protocol, the alternating bit protocol (a) is fed as input and a transition tour (b) is obtained as a result. There is some renaming of the states such as state 3_1 becomes 4, etc.

4.2. Test Design from Estelle

Since FSMs model only the control component of the protocols/ services, there is a need to extend the FSM based techniques to cover other aspects too. Since Estelle formal specification language can completely model all aspects of protocols/ services, it is desirable to be able to base the test design on an Estelle specification of the protocol/ service.



```
1 new   3 Time_out   3 Err   3 A1   3 A0    4 new
  D0      D0           D0      D0     nill    D1

6 Time_out    6 Err   6 A0   6 A1     1
  D1            D1      D1     nill
```

Figure 4.1. A Transition Tour for the Alternating Bit Protocol

A methodology that is inspired from software testing [Howd 86], FSM techniques and fault tolerant computing has been developed and successfully applied to various protocols [SaBoCe 86, Sari 84]. An Estelle specification is first simplified in order to obtain its normalized form, i.e., control paths from the transitions are removed, procedure/ function calls are replaced by symbolic evaluation [ClRi 81] and finally modules are combined. The normalized specification can easily be represented graphically. Two different graphs are obtained: control graph for major state changes and data flow graph to show the flow of data from input service primitive/ PDU parameters to the context variables and from the context variables to output service primitive/ PDU parameters.

Control graph is an FSM, thus the techniques described in Section 4.1 are applicable. The concept of subtours are introduced: a subtour is a test sequence that starts and ends in the initial state of the FSM. A transition tour is made up of one of more subtours. A subtour can be used in determining the execution sequence of a test case.

Data flow graph shows global flow of data over context variables, i.e. all the operations on the variables of each module. It is possible to partition a data flow graph into blocs each bloc containing the flow over only one variable, ideally these blocs are independent but in most cases a given protocol contains some blocs which contain dependencies on other blocs. Bringing together two or more blocs which are related is called merging. "Related" blocs can sometimes be automatically detected while in some cases interaction with an expert user is assumed. The result is a partitioned data flow graphs in which partitions correspond to functions of the protocol. The number of partitions obtained is directly proportional to the complexity of the protocol.

Test design with this methodology is based on obtaining the control sequence for a test case from the subtours and parameter enumerations of the interaction primitives. Each partition is tested with one or more subtours until all the arcs in the partition data flow graph are covered.

The methodology is partly automated on a Sun workstation. The normalization and generation of the flow graphs are automated. The data flow graphs are partitioned into blocs but the merging is left to the user [BaSa 87].

4.3 Test Design from Lotos

Lotos formal description technique can also be taken as a basis for test design. The method developed is based on behavior tree representations of Lotos process abstractions [Stee 86].

Protocol specification in Lotos is converted to a tree specification in which each branch represents a possible event and the nodes at the end represent the state of the protocol after the event has occurred. The branches leaving the node represent the possible choices of events at that node. Two different choice criteria are distinguished: a choice as a result of external action (offers from the environment) and internal decisions (refusing a connection, etc.). External choice nodes are represented as e and internal choice nodes are represented as i-nodes in the tree. This tree which represents a protocol entity is used to derive tests with the observation that the tests describe the behavior of the environment. The tree that represents the tests is obtained from the protocol tree with e and i-nodes inverted.

The test tree gives the interaction sequences to test a protocol. Individual test cases can be obtained by taking one of the branches at the i nodes and pruning the others. A test case is deterministic since there are no internal decisions left.

Since complex protocols have an unbounded number of possible interaction sequences, the behavior trees used have to be extended to recursive trees. The general case is covered in [Stee 86] by giving the inversion rules for all possible choice expressions. These rules can be used to obtain interaction sequences of test cases from Lotos specifications. The data part is addressed in terms of structured events of Lotos. Rules from software testing are suggested for parameter enumerations, as in the methodology discussed in Section 4.2. It should be noted that the Lotos test design methodology is also appicable to asynchronous communication since it is always possible to obtain a Lotos description of the protocol entity composed with buffer processes at the upper and lower boundaries.

4.4 Hybrid Methods

A combination of FSMs and knowledge from the informal specification can be used to design test cases. A protocol is specified as an FSM extended with context information such as firing predicates for the transitions, local and global variables. Local variables are assumed to be of discrete domain. This specification is usually obtained from the informal specification by inspection. Next is the transformation of the context automata: local variables are enumerated and predicates are removed if they are satisfied. The transformed automaton is used in test design. The remaining predicates are classified as settable and uncontrolled. Settable predicates are handled by parameter enumerations. Unsettable predicates represent nondetermisism and handling them can be done by expert testers with some knowledge of protocol functions [CaSi 86].

Prolog can be used in test suite design. Since Prolog interpreters have a built-in backtracking, it is possible to obtain enumerations easily from Prolog. A FSM can be specified in Prolog and the resulting program, when interpreted gives all possible sequences (transition tours) from the machine. It is also possible to obtain a Prolog equivalent of an Estelle specification, except for the dynamic process structure [Boch 85]. The resulting protocol specification in Prolog can be used for test case derivation. Due to the invertability of Prolog programs, the Prolog specifications can be used in two ways: to feed a sequence and see if it is acceptable and to generate test sequences. The disadvantage of this method is that the translation to Prolog is manual and for complex protocols methods should be developed to cope with infinite number of test sequences.

A tool is developed to interactively let the user specify the protocol in Prolog and guide the test derivation in terms of assigning default values to interaction primitive parameters or doing parameter enumeration, etc. as described in [UrSh 86].

4.5. Specification of Test Suites

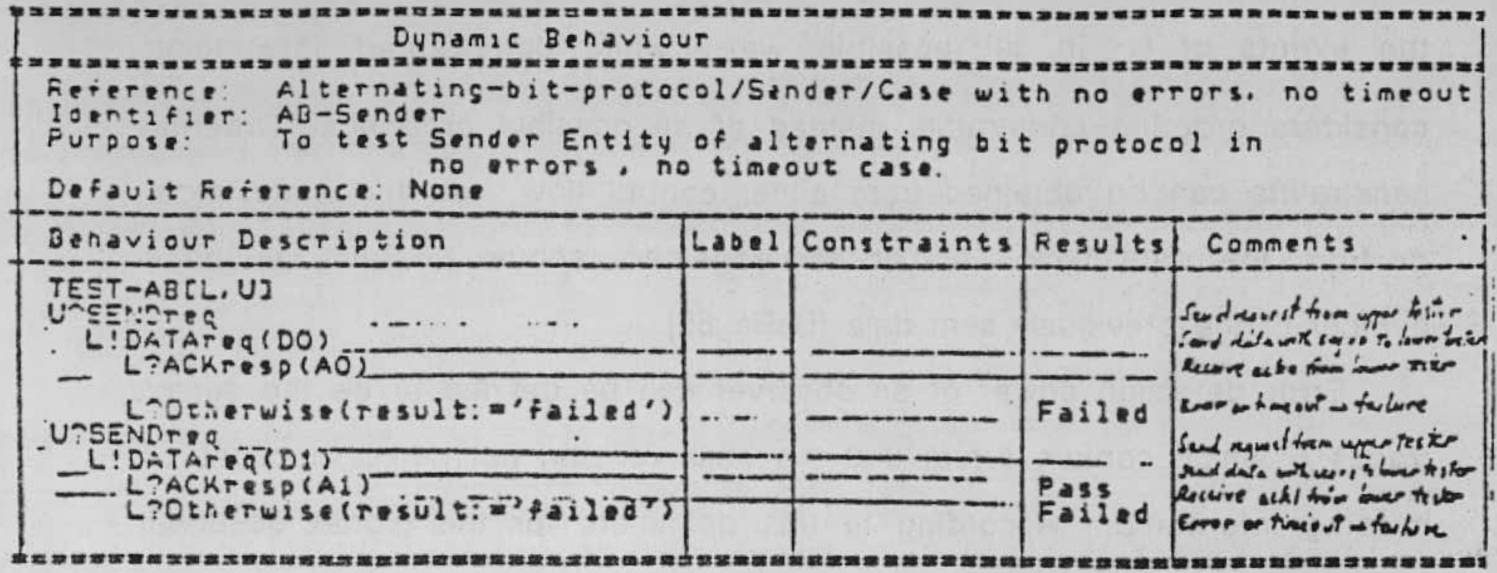
Test cases (suites) can be specified using either Estelle or Lotos. The advantage of specifying them in Estelle is that a large part of the executable code can be obtained [SaBoMaSe 86].

ISO conformance test subgroup has defined a notation for abstractly specifying test cases. This notation called Tree and Tabular Combined Notation (TTCN) specifies the test case as a tree of events input(?)/ output(!) from the service access points accesible to the upper/ lower testers. These access points, interaction primitives, etc. are defined in tables. Dynamic behavior table contains behavior description, label, PDU/ ASP reference, comments and result columns [ISO 86b]. It is possible to define the timers, start, cancel, suspend and resume the defined timers making it easy to define tests which involve timer management.

The aim of TTCN is to define a notation for standardized test suites. ISO committee is working towards standard test suites of individual protocols each specified in TTCN.

A tool with which user can interactively write and debug TTCN specifications has been realized [Wile 86]. Each test case specified can also be executed in a local test architecture.

Test cases derived by the methods of Sections 4.1 - 4.4 can be expressed in TTCN. As an example we give a TTCN description of a test of the alternating bit protocol in Figure 4.2. This sequence is derived from Figure 4.1.



Dynamic Behaviour

Reference: Alternating-bit-protocol/Sender/Case with no errors, no timeout
Identifier: AB-Sender
Purpose: To test Sender Entity of alternating bit protocol in no errors , no timeout case.
Default Reference: None

Behaviour Description	Label	Constraints	Results	Comments
TEST-AB[L,U]				
U?SENDreq				Send request from upper tester
L!DATAreq(D0)				Send data with seq 0 to lower tester
L?ACKresp(A0)				Receive ack 0 from lower tester
L?Otherwise(result:='failed')			Failed	Error or timeout → failure
U?SENDreq				Send request from upper tester
L!DATAreq(D1)				Send data with seq 1 to lower tester
L?ACKresp(A1)			Pass	Receive ack 1 from lower tester
L?Otherwise(result:='failed')			Failed	Error or timeout → failure

Figure 4.2. A TTCN Test Case Specification for the Alternating Bit Protocol

5. THEORY

Investigation of issues such as formally expressing error detection power of test architectures, deciding if a set of events that happens during a test (called trace) satisfies the protocol specification, evaluating fault coverage of the test suites and developing formal notions of implementations and using this notion to obtain formal specification of the conformance tests of implementations as well as the testers constitute the theory of protocol testing.

5.1. Error Detection and Trace Checking

A trace is an totally ordered set of observable events. Each member in the set is referred to as a sequence of events. There are two types of traces in the context of protocol testing: global trace is the trace observed in all the observable interaction points of the test architecture, local trace is the trace obtained from a subset of the observable interaction points. The projection operation can be applied on the traces to reduce the global trace to a partial trace observable from a subset of interaction points. Projection makes the events of the extracted interaction points unobservable, thus it is equivalent to restriction operator of Lotos in this sense. Arbitrary interlacing of two partial traces t_1 and t_2 is the set of sequences obtained by placing the events of t_2 in the events of t_1 in all possible ways while constrained interlacing considers ordering constraints instead of all possible orderings. Ordering constraints can be obtained from either control flow, i.e., the state order or from the data flow, i.e., acknowledgement should contain sequence number of the previously sent data [DsBo 85].

Error detection power of an observer can be defined to be the set of traces E which contain errors that the observer can not detect because of missing information. According to this definition, for the global observer which observes all the interaction points the following holds:

$$E = \emptyset$$

since the global observer has no missing information.

Trace checking done by the local and global observers is based on verifying local and global properties, respectively of a protocol on the observed trace. It has been shown that these properties can be obtained from formal specifications of the protocol in Estelle [Dsso 86] and Lotos [AhSa 87]. Automatic implementation of observers from formal specifications of the protocol under test is desirable and has been realized in the case where observation in only PDU level is assumed [Zhao 86]. Finite-state machine based implementations are also easy to obtain but the resulting observers are partial in the sense that they cannot verify local (global) properties which contain nondeterminism (a prefix of a given trace can be acceptable in several states of the observer) [Dsso 86, Jard 86].

Another dimension of the error detection power is the test suite applied. Fault coverage of the test suite should be as complete as possible. Quantitative ways of determining test suites fault coverage are desirable, but difficult to obtain in general. Again, for finite-state machine models, it is possible to measure the fault coverage of a given test sequence or thereby the method of determining the test sequence. [DaSa 86] contains a study of fault coverage of the test sequences obtained from the technique of unique input output sequences. The method used is to determine all possible equivalent machines implementing a given protocol expressed as a FSM and determining all correct ones among them. The computations involved tend to grow exponentially when the number of states of the FSM grows.

5.2. Canonical Testers

[BrScSt 86] shows that it is possible to obtain, from a Lotos specification of the protocol, a Lotos description of conformance tests of the protocols implementations and the associated testers. An implementation can be considered to be a reduction, an extension or a refinement of its specification. These notions can be formally defined and interrelationships among them can be established in terms of sets of observable actions and trace sets of transition systems. Based on these a canonical tester T(S) of a specification S can be defined to accept all the traces of S and deadlock in case only if the process is not an abstract implementation of S. Although in general, there exists no algorithms for constructing T(S), it is possible to find it for simple S. This is possible from the facts that no reduction of T may deadlock with any reduction of S and that a reduction of T(S) exists that deadlocks with processes that are not abstract implementations of S.

6. CONCLUSIONS

Protocol testing is undertaken either locally usually by the implementers or from a distance usually for the purpose of assessing its conformance to the protocol specification. Architectures developed for both local and remote testing show that protocol testing has distict characteristics compared with software and hardware testing, due to distributed control and access involved in these architectures.

Test case generation for protocols is closely related to protocol specification. Any method used exposes the capabilities and limitations of its underlying specification technique. For simple protocols the methodologies developed work well but for complex protocols there seems to be yet no technique found to cope with the complexity. On the other hand test case generation is very much related to expertise gained on the protocol. It seems easier for an expert designer to design tests manually rather than using an automated tool for complex protocols. This point needs further investigation.

Much remains to be done on issues such as fault coverage of test cases, designing intelligent observers and obtaining test suites automatically from the formal specification techniques as well as formal description of the testers.

7. REFERENCES

[AhSa 87] R. Ahooja, B. Sarikaya, "Unified Data Flow Models of Estelle and Lotos Specifications", Submitted for publication. Available from the author. February 1987.

[Barb 87] M. Barbeau, "Prototype d'un Système d'Aide à la Conception de Tests de Protocoles", MSc thesis, Université de Montréal, February 1987.

[BaSa 87] M. Barbeau, B. Sarikaya, "CAD-PT: A Computer Aided Design Tool for Protocol Testing", Submitted for publication, available from the author, February 1987.

[Boch 78] G.v. Bochmann, "Finite State Descriptions of Communication Protocols", Computer Networks, Vol.2, pp361-372, October 1978.

[Boch 85] G.v. Bochmann, R. Dssouli, W. de Souza, B. Sarikaya, H. Ural, "Prolog for Building Protocol Design Tools", Proc. of 5th IFIP Workshop on Protocols, June 1985, North-Holland 1986.

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

[BrScSt 86] E. Brinksma, G. Scollo, C. Steenbergen, "Lotos Specifications, their Implementations and their Tests", Proc. 6th IFIP Workshop on Protocols, June 1986, North-Holland 1987.

[CaSi 86] R. Castanet, R. Sijelmassi, "Methods and Semi-automatic Tools for Preparing Distributed Testing", Proc. 6th IFIP Workshop on Protocol Testing, June 1986, North-Holland 1987.

[Chow 78] T.S. Chow, "Testing Software Design Modelled by Finite-State Machines", IEEE Trans. on Software Eng., Vol. SE-4, No. 3, 1978.

[ClRi 81] L.A. Clarke, D.J. Richardson, "Symbolic Evaluation Methods for Program Analysis", in Program Flow Analysis, Prentice Hall, 1981.

[Dant 80] A.S. Danthine, "Protocol Representation with Finite State Models", IEEE Trans. on Comm., Vol. COM-28, No. 4, 1980.

[Dsso 86] R. Dssouli, "Etude des Methodes de Test pour les Implantations de Protocoles de Communication Basées sur Les Specifications Formelles", PhD Thesis, Université de Montréal, Dec. 1986.

[DsBo 85] R. Dssouli, G.v. Bochmann, "Error Detection with Multiple Observers", Proc. of 5th IFIP Workshop on Protocols, June 1985, North-Holland, 1986.

[EhMa 85] H. Ehrig, B. Mahr, "Fundamentals of Algebraic Specification I", Springer-Verlag, Berlin, 1985.

[Este 85] ISO, "Estelle: A Formal Description Technique Based on an Extended State Transition Model", DP 9074, May 1985.

[Howd 86] W.E. Howden, "A Functional Approach to Program Testing and Analysis", IEEE Trans. on Software Eng., October 1986.

[ISO 84] ISO, "Open Systems Interconnection -Basic Reference Model", IS 7498, 1984.

[ISO 86] ISO, "OSI Conformance Testing Methodology and Framework Part1: General Concepts", DP 11, Sept. 1986.

[ISO86b] ISO, "OSI Conformance Testing Methodology and Framework Part 2: Abstract Test Suite Specification", DP 12, Sept. 1986.

[Jard 86] C. Jard, Personal communication, December 1986.

[Koha 78] Z. Kohavi, "Switching and Finite Automata Theory", Mc Graw Hill, 1978.

[Loto 86] ISO, "Lotos- A Formal Description Technique Based on the Temporal Ordering of Observational Behavior", DP8807, 1986.

[Miln 80] R. Milner, "Calculus of Communicating Systems", LNCS of Springer Verlag, 1980.

[MoDiAy 85] A.R. Molva, M. Diaz, J.M. Ayache "Observer: A Run Time Checking Tool for Local Area Networks", Proc. 5th IFIP Conf. on Protocols, Toulouse 1985, North-Holland 1986, pp. 495-506.

[Pouz 86] L. Pouzin, "OSI Progress and Issues", Proc. of 8th ICCC, September 1986, Munich, pp. 154-158.

[Rafj 85] O. Rafiq, R. Castanet, C. Chraibi, "Towards an Environment for Testing OSI Protocols", Proc. 5th IFIP Conf. on Protocols, Toulouse 1985, North-Holland 1986, pp. 533-544.

[Rayn 82] D. Rayner, "A System for Testing Protocol Implementations", Computer Networks, Dec. 1982.

[Sari 84] B. Sarikaya, "Test Design for Computer Network Protocols", PhD. thesis, Mc Gill University, March 1984.

[SaBo 82] B. Sarikaya, G.v. Bochmann, "Some Experience with Test Sequence Generation for Protocols", Proc. of 2nd Workshop on Protocols, May 1982.

[SaBo 84] B. Sarikaya, G.v. Bochmann, "Synchronization and Specification Issues in Protocol Testing", IEEE Trans. on Comm., Vol. COM-32, No. 4, pp.389-395, April 1984.

[SaBoMaSe 86] B. Sarikaya, G.v. Bochmann, M. Maksud, J-M. Serre, "Formal Specification Based Conformance Testing", Proc. SIGCOMM' 86, August 1986.

[SaBoCe 86] B. Sarikaya, G.v. Bochmann, E. Cerny, "A Test Design Methodology for Protocol Testing", to appear in IEEE Trans. on Software Engineering. Available as a research report from Concordia University.

[SaDa 85] K. K. Sabnani, A. Dahbura, "A New Technique for Generating Protocol Tests", Proc. of 9th Data Communications Symposium, ACM SIGCOM Computer Communication Reviews, Vol. 15, No. 4.

[SeCeBo 86]J-M. Serre, E. Cerny, G.v. Bochmann, "A Methodology for Implementing High-Level Communication Protocols", Proc. of 19th HICSS Vol. IIB, January 1986.

[Steen 86] C. Steenberger, "Conformance Testing of OSI Systems", MsC. Thesis, Twente University, August 1986.

[UrSh 86] H. Ural, R. Short, "An Interactive Test Sequence Generator", Proc. of SIGCOMM' 86, August 1986.

[VuLa 87] S.T. Vuong, A. Lau, "Semi-Automatic Implementation of Protocols-The ISO Class 2 Transport Protocol as an Example", Proc. IEEE INFOCOM' 87, March 1987.

[Wan 87] T. Wan, "CONTEST- Concordia Test Generation Package User's Manual", Concordia University, Electrical Eng. Dept., Available from the author.

[Wiles 86] A. Wiles, "ITEX-An Interactive TTCN Editor and Executer", Proc. GLOBECOM' 86, December 1986, pp. 22.3.1-22.3.5.

[Zhao 86] J-R. Zhao, "Protocol Testing with Arbiters", Research report, Université de Montréal, October 1986.