Some experience with the use of formal specifications†

G.V. Bochmann, E. Cerny, M. Gagne, C. Jard*, A. Léveillé, C. Lacaille, M. Maksud, K.S. Raghunathan, and B. Sarikaya**
Département d'informatique et de recherche opérationnelle
Université de Montréal
C.P. 6128, Succursale A
Montréal, P.Q.
H3C 3J7

Abstract

Experience with the use of formal descriptions of communication services and protocols is described. The paper focusses on the experience by the authors with the extended state transition model which is proposed as a standard formal description technique (FDT) for the services and protocols in the OSI environment. The first part of the paper refers to various example specifications, including Transport protocol and service specifications, and discusses the suitability of the specification method and possible extensions. In the remaining part, the use of such formal specifications during the phases of system design, implementation and testing is described. Various approaches to protocol design validation, implementation, and assessment of implementations are discussed, with emphasis on the last point. The experience with several of these approaches is described in the paper, while further details may be found in the references.

†Work partly supported by the NSERC and research contracts from the Departement of Communications of Canada.
*CNET, Lannion, France
**School of Computer Science, McGill University, Montreal

1. Introduction

Formal methods are considered important tools for the reliable design and implementation of communication protocols. These methods are always based on some formal specification of the communication protocol and/or services given in some appropriate formal description technique (FDT). Among the different FDT that have been proposed and used (see for example [Boch 80]), we consider in this paper mainly the extended state transition model [FDT 81] developed by Subgroup B of the ISO TC97/SC16/WG1 ad hoc group on FDT (or similar dialects). This is a descriptive model which combines the state transition nature of finite state machines with the power of a high-level programming language (Pascal). Similar approaches to the specification of protocols have been described in the literature [Boch 77, Boch 80b, Tenn 81].

The different activities during the design and implementation of protocols where formal specifications can be useful are summerized in [Boch 81]. The main activities are:

a) The elaboration of a reference specification of the communication protocols or services,

b) The validation of the design of a protocol specification,

c) The design and development of a protocol implementation based on the protocol specification obtained under point (b),

d) the validation of a protocol implementation obtained under point (c).

In this paper we discuss some experience we had in the use of the extended state transition FDT for the above mentioned activities. We also make some reference to similar work that is proceeding at other places, although we do not pretend to give a complete review of this area.

The paper is structured as follows: Section 2 relates our experience with the use of the ISO extended state transition model (or similar local dialects) for the writing of formal protocol and service specifications. Some critical comments based on this experience are given in section 2.5. The sections 3, 4, and 5 deal with the activities (b), (c), and (d) mentioned above. The main part of each of these sections gives a description of recent work done by our group in these areas. Due to lack of space, the discussions are relatively short, and references are provided for more detail.

2. Example specifications

2.1. The Transport layer as a test case

The ISO ad hoc group on FDT has chosen the Transport layer as the principal test case for comparing different FDT's proposed for the specification of OSI protocols and services. As a result, many different formal and semi-formal specifications of the Transport protocol and service have been developed (see for example the papers presented at the ad hoc group's meetings).

The Transport layer service [CCITT/ISO a] is a connection-oriented communication service that supports normal and expedited data transfer. Different classes of protocols [CCITT/ISO b] are defined, each providing a different set of functions. The available functions are

a) connection establishment with the selection of an appropriate protocol class and options,

b) addressing of Transport service access points (TSAP),

c) multiplexing,

d) error detection, and (possibly) recovery,

e) independent flow control for normal and expedited data over different connections,

f) recovery of Network connection failures, etc.

Since the above mentioned CCITT/ISO documents are relatively recent, most work with FDT's is based on previous CCITT, ISO or ECMA documents, and are often restricted to the protocol classes 0 and 2.

2.2. Specifications of the Transport protocol

Different versions of Transport protocol specifications have been produced by our group as contributions to the discussion on FDT's. We mention here only the following two versions which are of different scope:

The class 0 protocol specification in [TP a] is written in a local dialect [FDT 81b], and was later rewritten according to the ISO syntax [TP b]. The purpose was the description of the basic rules of the Transport protocol in a most simple manner. Therefore the specification considers only a single Transport connection (multiplexing is not allowed for class 0), and only the

"abstract protocol" (see [ISO 81a] section 4.3) is defined, ignoring the mapping of the protocol data units into the Network service primitives. The transitions may either be grouped by major states [TP a] or by incoming interaction [TP b].

Reference [TP c] contains a quite complete protocol specification for classes 0 and 2. It considers an arbitrary number of simultaneous Transport connections over an arbitrary number of Network connections, including the possiblity of multiplexing. The mapping of PDU's into Network service primitives is also defined, except for the detailed coding of the different PDU parameters. The mapping function considers possible concatenation of several PDU's to form a single Network service data unit, and the priorities of different connections and different kinds of PDU's. It seems that the possible non-determinism of the FDT (see section 2.5.1 below) is an essential feature for leaving certain implementation choices undefined.

Many diffirent formal Transport protocol specifications have been written using Pascal [Lefe 81], Ada [Buhr 81], extended Petri mets [Bert 81] and other methods [FDT 81c, Vogt 82]. Space limitations prevent us from providing further references and comparisons.

2.3. Specifications of the Transport service

The Transport service may be specified with the same FDT giving a specification of the Transport layer and the layers below considered as a black box (see figure 1, dashed box). This approach has been taken for the specification of [TS a] which describes the properties of the Transport service as seen by the users through the service access points. As in [TP c], an arbitrary number of simulataneous Transport connections is considered. A simplified version, considering only a single connection, and ignoring the problem of addressing, is given in [TS b] using a local dialect [FDT 81b] of the FDT.

Many other Transport service specifications have been written by different groups [FDT 81c, Burt 81, Vogt 82]. There is not enough space to discuss them all. However, we would like to mention here the question whether it is useful to separate, in the service specification, the local and global [Boch 80b] sequencing rules for the execution of service primitives. A general framework for such a separation is given in [Boch 82]. While the state transition model seems adequate for the specification of the local rules [FDT 82a], its use for the global rules is more questionable. While reference [Boch 80c] uses a different specification language (related to temporal logic) for the global rules, we are presently experimenting with separate specifications for the local and global rules using the extended state transition

model for both [Ragu 82].

2.4. Other example specifications

As mentioned in the Introduction, the general approach of using an extended state transition model for protocol specifications is not new. Some of our previous work on HDLC [Boch 77b], X.25 [Boch 79], and the Message Link Protocol [Boch 79b] lies on this line. More recently, the NBS (USA) has funded the developement of a FDT (similar to [FDT 81]) for use in their protocol development project [Tenn 81]. The formal protocol specifications developed in this context are interesting examples.

Another example is some effort [Boch 81b] for developing a formal specification of the Teletex control procedures. In this effort the Teletex Session and Document layers where described using the local FDT dialect [FDT 81b]. In order to clarify the relation between the different layers (including the underlying Transport layer), an attempt was first made to give a formal description of the services provided by the Session and Document layers. The protocol specifications are then given in reference to these service specifications. It may be interesting to note that the selection of the Document service primitives seems to necessitate some arbitrary choices. Some of these primitives are related to a document file store.Some kind of "virtual file store" was defined in a semi-formal manner (see section 2.5.5 below). A last example is the specification [Boch 81 c,d,e] of a Virtual File system developed by the Hahn-Meitner-Institute, Berlin. The specification is given in two parts: The first part is the specification of a virtual file server, i.e. it defines the local input/output behavior of a file server in terms of file service primitives exchanged with it local environment. This part of the specification defines the meaning of such primitives as OPEN, READ, WRITE, etc. The specification uses a dialect of the extended state transition model, however, several extensions to the syntax of [FDT 81] seemed necessary for this example, as explained in the section below. The second part of the specification defines the communication protocol used for the evocation of the file service primitives over distance. The system is characterized by three protocol sub-layers (above the Transport service) which are specified separately.

2.5. Suitability of the FDT and possible extensions

We conclude from the above mentioned experience of writing formal specifications that the FDT of [FDT 81] is a flexible tool which leads to relatively readable specifications. One of the main problems to be decided for each specification was

the overall order in which the different transitions of the specification should be arranged, in order to arrive at a most understandable presentation. Such decisions are sometimes quite arbitrary, often related to the personal taste and prejudices of the person writing the specification. Consequently, some informal guidelines would be useful for this purpose. Reference [Chun 82] tries to give some objective arguments for a particular organization of the transitions (ordering by incoming interactions) and shows how such a discipline can be useful for the systematic development of protocol specifications during the design phase.

The following subsections contain comments on certain points of the specification language and its use, and they point out some possible extensions.

2.5.1. Non-determinism

The FDT of [FDT 81] is based on a model of a non-deterministic state machine. It is sometimes argued that non-deterministic behavior is not required, or desirable, for protocol specifications. We have found non-determinism an important element of the specification language in the case of service specifications, where the relation between the interaction at the different service access points is not deterministic, as well as in the case of protocol specifications, where (in [TP c] for instance) the priority of certain possible operations of the protocol entity is not always defined; for example priority of different multiplexed connections, extent of concatenation of multiple PDU's into service data units, possible overtaking of data by disconnects, etc.

2.5.2. Incomplete specifications

A specification of a protocol entity or a communication service makes usually some assumptions on the behavior of other modules in the system. Under these assumptions not all possible interaction patterns will occur. Therefore, it seems reasonable to give specifications that are incomplete in the sense that they define the behavior of the specified system module only for the case that the above mentioned assumptions are satisfied. We assume the following convention concerning completeness of a module specification: If for some given input interaction (with some particular parameter values) and some given module state no possible transition is defined, then the specification is incomplete and the behavior of the module is not defined for this situation.

Such a situation should not occur under the assumptions mentioned above. In the case that "in the real world" no transition is specified for an input that occurs we can therefore say that the above mentioned assumptions are not satisfied, and that an "unforeseen error" occured in the environment of the specified module.

It is certainly desirable to foresee some of the possible errors of the environment, in particular misbehaviors of the peer protocol entity. Transitions for these error cases should therefore be included in the formal specification and not be left as "unforeseen errors". How much of such error cases should be included seems to be a matter of taste. Some, but not all, protocol specifications try to specify actions for every possible misbehavior of the peer entity. In the OSI environment, transitions treating user misbehavior should probably not be defined, since they may be considered part of the service interface which is a local implementation issue.

The above discussion of the meaning of incomplete specifications becomes more subtle in the context of non-determinism. We propose the following definition: An input interaction from the environment to the specified module is an unforeseen misbehavior if the specification of the module provides for the possiblity that the sequence of preceeding interactions leads to a state of the specification for which there is no transition specified for the interaction under consideration [Jard 81].

2.5.3. Special syntax for major states

We are not convinced that the special syntax for the major module state (FROM and TO clauses) are warranted for the specification of protocols and services, since the PROVIDED clause and assignment statements could be used instead. This seems to be more flexible for specifying multiple connection endpoints.

2.5.4. Use of assertions

The use of assertions for the specification of software is well known. We found that this method could naturally be incorporated into the extended state transition model by using assertional specifications in the following three cases:

a) The meaning of procedures and functions used in transitions can be specified by input/output assertions on the parameter values.

b) Sometimes, individual statements within a transition may be considered largely implementation dependent; however, the specification may state some essential properties. These properties may be defined by assertions on the module state variables (possibly relating the values before and after the execution of the statements).

c) There are situations where the action of a whole transition may be best defined by assertions which relate the state values before and after the transition (instead of defining a statement sequence which performs the state transformations). Such an approach is similar to the definition of O-functions in Special [Robi 79].

The cases (a) and (b) have been used in [NS a, TS a, TP c], and the case (c) would have been useful in the specification of the Virtual File server [Boch 81d] for defining the meaning of the PDS primitive which positions a pointer in the hierarchical structure of a file.

2.5.5. Abstract data types

Certain aspects of a specification are usually left informal, since the specification language is not well suited to describing these aspects (it would usually lead to unnatural, lengthy descriptions). Such is often the case with data buffers that are used in the descriptions of the Transport protocol [TP a,b,c] or service [TS a,b]. In the example of the Teletex Document protocol [Chow 82b], the "virtual file store" mentioned above and a "document manager" were described along similar lines. Usually semi-formal descriptions are given, declaring a number of "primitive" procedures and/or functions that may be called and explaining their meaning in natural language.

These are examples where formal descriptions based on the formalism of abstract data types (as developed for software engineering) may be useful. Note, that these difficulties could possibly be avoided by adopting an FDT that is different from the extended state transition model assumed in this paper. However, we have not been convinced that such an approach is preferable.

3. Protocol design validation

The objective of protocol design validation (see for example [Boch 80]) is to verify that a given protocol specification for layer N, together with the given service specification for layer (N-1) imply that the (N)-layer service is provided by the layered system architecture shown in figure 1.

3.1. Protocol design verification

Under this heading we consider static analysis of the specifications. A review of different approaches to verification is given in [Boch 80]. Techniques that are relevant for the extended state transition model are reachability analysis for finite state machines, invariant analysis for Petri nets [Azem 80, Bert 81], and program proving techniques. When a "major state abstraction" of the system is considered (which ignores the interaction parameters and additional state variables of the model) the techniques developed for finite state machines and Petri nets are applicable, and often provide useful insight into the possible interaction sequences. For a complete verification, however, the interaction parameters and additional state variables must be considered and require usually some verification methods related to program proving (for example assertions and invariants, symbolic execution, etc. ; a simple example is discussed in [Boch 77]).

We are presently working on the verification of a class 0 Transport protocol based on the specifications given in [TP a, TS b, NS a] and the standard mapping of PDU into Network service primitives. Gobally, the verification proceeds through the following three steps:

1) The three modules shown in figure 1 (protocol entities and Network service provider) are combined into a single machine. In the "major state abstraction", this combination corresponds to the formation of a product finite state machine or a Petri net, where it is important to consider the direct coupling of the input/output interactions between the combined modules (see for example [Merl 82] or [Devy 79]). Special attention must be given to the interaction parameters and additional state variables in the combined modules.

2) From the viewpoint of the service user, the input/output interactions between the combined modules may be ignored. This view may be obtained by projections [Merl], or Petri net reductions [Bert 76, Devy 79]. In order to reduce the complexity of the problem, it may also be useful to consider only a particular service property at a time, as explained in [Lam 81].

3) Finally, the abstracted machine specification obtained under point (2) must be compared with the given Transport service specification. For the verification of the safeness properties, it is necessary to show that all execution sequences obtained from the machine specification of point (2) are allowed according to the service specification. In

addition, it is necessary to show that all liveness properties of the service specification are provided by that machine (which includes the absence of deadlocks and similar general properties). It is hoped that the specification obtained in point (2) is not very different from the given specification of the Transport service. Any difference found may point to an inconsistency in the specifications. The detailed application of these ideas to the verification of the Transport protocol may be found in [Laca 82].

3.2. Testing of protocol designs

Under this heading we consider testing of protocols by directly executing their specification. This is a kind of simulation approach, where the three modules shown in figure 1 are executed in some simulated environment and the behavior of the simulated system is observed and compared with the given service specification. Such approaches can be used for analyzing the logical behavior of the system (in which we are interested at this point), as well as the performance characteristics [Lela 79, Didi 80].

For the realization of the simulation, the automatic implementation approaches discussed in the next section may be used. Another problem is the automatic comparison of the behavior resulting from the simulated system with the given service specification. In the case that the behavior of the service is non-deterministic (which is usually the case), the different choices possible according to the service specification must all be explored, in order to check whether one of them corresponds to the behavior observed. The creation of such a checking module from the formal specification of the service is explored in [Jard 81].

The simulation requires the generation of user input interactions which must be chosen in such a way as to maximize the probability of detecting any possible malfunctions. The problem is similar to the selection of test sequences for protocol implmenetation testing, as discussed in section 5.

4. Automating protocol implementation

Since the extended state transition model combines elements from finite state machines and programming languages, it is relatively easy to obtain an implementation for a given specification in the form of a program. Implementations of finite state machines in software are straightforward, and the other elements are already in a programming language form. The typical approach is to implement a specification as a looping program where each cycle of the loop executes a transition. The transition is either initiated by some input interaction or by some internal

condition that makes its execution possible. The loop could consist of a CASE statement with one case per kind of input interaction (including "no input"). For each of these cases the internal conditions may be tested again by a CASE statement testing the major state of the module, or by successive IF statements to select the appropriate transition to be executed.

The implementation of the interactions with the other modules in the system (input and output interactions) is very system dependent. In an implementation of the Transport protocol on our PDP-11 computer [Leve 82] the Transport service interactions are realized by the exchange of messages passed via a shared memory region between the users and the Transport module, which are separate tasks under the RSX-11M operating system; whereas the Network layer (X.25 software) is incorporated in the operating system and accessed through supervisor calls.

Automatic translation of formal specifications into programs is also possible [Tenn 81,Gagn 82]. Usually the specifications are translated into some program elements (as described above) which call upon a system dependent run-time support implementing interactions with the other modules in the system, buffer management, time-outs etc.

It is not always desirable to implement each separately specified module as a separate program or task. It is therefore interesting to investigate methods by which different separately specified modules may be combined into a single implementation module. A method for combining separately specified protocol phases (which are related by a "hierarchical dependence") into a single implementation module is described in [Boch 79]. A similar approach can also be used for combining the protocol entities of different layers, provided that the condition of hierarchical dependence between the protocols is satisfied. This is, for instance, the case for the CCITT Teletex Transport, Session and Document protocols.

5. Assessment of protocol implementations

We consider here all activities used for verifying whether a particular protocol implementation adheres to the corresponding protocol specification. If such checking is performed by an official organization against a standard reference specification, then the activity may be called "protocol implementation certification". The assessment activity consists of applying tests to the implementation (or "unit under test", UUT). The tests are qualitative or quantitative depending on their objective that is, either checking the logical conformity of the implementation to its specification, or measuring certain

performance parameters such as throughput, delays, reliability, etc.

Plans for instituting "certification centers" for OSI protocols exists in several countries [Ansa 80, NPL 81, McCo]. Different approaches may be considered for the certification of an Open System for its conformance with OSI protocol standards. The validation can be made most complete when the system provides access to the interfaces between the different protocol layers, such that effectively each layer of the system to the validated may be tested separately. It is also possible to make some overall tests involving many layers at once, for example from the Transport layer up through the Presentation layer using the lower-level network access protocols and the application interface to the Presentation layer as "access points" to the module under test. There may, however, be limitations as to the effectiveness of such a combined multilayer testing procedure.

Among the various test architectures [Boch 82b] the Remote tester (Figure 2) and a supplementary Local Tester (directly connected to the UUT) are receiving currently most attention. Similarly our efforts are directed towards gaining experience in constructing a Remote test system. Two versions of the system are under development, an interactive one and an automatic one. The following two subsections describe their objectives and general organization, while the last subsection explains an effort towards developing meaningful test sequences.

5.1 The interactive tester

Here, the objective is to provide a flexible tool destined mainly for debugging protocol implementations, and to a limited extend for qualitative testing.

The interactive tester module is placed as a peer entity with respect to the UUT. Using a computer terminal, the user can construct arbitrary (also erroneous) interactions (PDU's and control service primitives) to be sent to the UUT, and examine those arriving from the tested unit. Hence, the main functions of the module is to create an easily useable interface for the human operator, freeing him for performing all coding and decoding functions for the various PDU's as well as handling the necessary underlying connections.

A similar module can be connected to the service interface of the UUT, interacting with the UUT by service primitives. Alternatively, an automatic responder (see Section 5.2) could be used.

5.2 Automatic Remote Tester

5.2.1 The objective

The objective in this case is to develop an experimental installation aimed at

a) studying the structure of the Peer Test Module (PTM) and the Test Module (TM) (See figure 1), so as to obtain a system least dependent on the type of protocol tested, and

b) providing a vehicle for experimental evaluation of the techniques used for deriving various test sequences.

Naturally, the ultimate goal is to use the results of the experimentation towards the development of an assessment system that is efficient, reliable, and easy to use.

5.2.2 The approach

In order to achieve the flexibility required by the objectives, we have opted for an organization providing a set of support modules for applying various tests. One of the modules takes care of sequencing the various tests, based on a high level sequencing scenario and the outcome of previous tests.

For each individual test, the behaviour of the TM and PTM are described using the FDT [FDT 81]. These descriptions are then compiled (manually or automatically) into executable programs which are loaded by the support modules, and use their services.

The main support modules in the PTM (Active Tester) are:

- test sequencer,
- report generator,
- test loader/TM protocol handler,
- initial connection establishment test module (needed for down-loading of detailed tests to the TM),
- PDU mapping module.

In the case of the TM (Passive Responder) the modules are:

- initial connection establishment test,
- test loader/TM protccol handler,
- SDU mapping module.

The implementations at both the PTM and the TM will be running on a PDP-11 computer under the RSX-11M operating system. Communication between the various modules is achieved through a

shared memory region and synchronization services of the operating system. The PDU and SDU mapping modules, and the individual test sequences are currently adapted towards testing a Transport protocol implementation.

5.3 Test sequences

Although a wealth of information is available on software and hardware testing techniques, very little is so far known about testing protocol implementations, and unfortunately the hardware and software methods are not directly applicable here. It is important to note that the protocol implementation details (software listing, plans, etc.) are not always available. Consequently, the testing techniques must treat the UUT as a "black box", and the adherence of the UUT to the specification must be deduced purely from its responses to appropriate test sequences. In addition, it is useful to determine the behaviour of the UUT under unspecified or erroneous inputs in order to obtain a complete characterization ("friendliness") of the implementation [Boch 82b].

The techniques for deriving test sequences for protocols are thus an open research area. The starting points [Boch 82b] could be considered the existing approaches in microprocessor testing [e.g. That79], machine identification [Koha 78, Nait 81] and certain software testing techniques [Chow 78]. It may be necessary to test an protocol implementation by functional sub-modules and/or to introduce a protocol-specific fault model.

In our group, finite state machine testing techniques are currently being explored. A number of interesting results are reported in [Sari 82]. They include the derivation of checking sequences, transition tours and characterization sequences for protocol machines. The major problems encountered are related to the incompleteness of the specification, the synchronization of the PTM and TM, the length of the test sequences, and the existence at parameters and secondary state variables in the specification.

The last two items imply that the tests will not be complete (except in some trivial cases) in the sense of completely verifying the absence of design faults in the implementation [Piat 80].

6. Conclusions

The discussions in the preceeding sections show how a formal specification of communication services and protocols can be a used for the various activities during the design and

implementation of distributed systems. Although the discussion focuses on the experience of our group with a particular formal description technique [FDT 81] which is proposed to be used in the OSI environment, we feel that approaches similar to those described here would be useful in may other situations, including the design and implementation of non-standard communication protocols, distributed application development, and modular system design in general.

References

[Azem 80] P. Azema, B. Berthomieu, P. Decitre, "The Design and Validation by Petri Nets of a Mechanism for the Invocation of Remote Servers", Proc. IFIP Conf. 1980, pp. 599-604.

[Boch 77] G.V. Bochmann and J. Gecsei, "A Unified Model for the specification and verification of protocols", in Proc. IFIP Congress, 1977, pp. 229-234.

[Boch 77b] G.V. Bochmann and R.J. Chung, "An Formalized Specification of HDLC Classes of Procedures," in Proc. Nat. Telecommun. Conf., Los Angeles, CA, Dec. 1977, Paper 3A.2.

[Boch 79] G.V. Bochmann and T. Joachim, "Development and Structure of an X.25 Implementation," IEEE Trans. Software Eng., vol. SE-5. pp. 429-439, Sept. 1979.

[Boch 79b] G.V. Bochmann, "Formalized Specification of the MLP", "Specification of the Services Provided by the MLP", and "An Analysis of the MLP". Département d'informatique et de recherche opérationnelle, Université de Montréal, June 1979.

[Boch 80] G.V. Bochmann and C.A. Sunshine, "Formal Methods in Communication Protocol Design", IEEE Trans. COM-28,4 (April 1980), pp. 624-631.

[Boch 80b] G.V. Bochmann, "A General Transition Models for Protocols and Communication Services", IEEE Trans. Com. 28, 4 (April 80), pp. 643-650.

[Boch 80c] G.V. Bochmann, "Concept for the Specification of Protocols and Services", INWG Note; contribution to ISO TC97/SC16/WG1 ad hoc group and FDT meeting, Amsterdam 1980.

[Boch 81] G.V. Bochmann, "The Use of Formal Description Techniques for OSI Protocols", Proc. National Telecom. Conf., New Orleans, December 1981, pp. F8.6.1-6.

[Boch 82] G.V. Bochmann and M. Raynal, "Structured Specification of Communicating Systems", Publ. 428, Département d'informatique et de recherche opérationnelle, Université de Montréal, 1982.

[Boch 82b] G.V. Bochmann and E. Cerny, "Protocol Assessment", Dendronic Decision Ltd., 1982, prepared under contract for DOC,

Canada.

[Bert 76] G. Berthelot, G. Roucairol, "Reductions of Petri Nets", Proc. of the MFCS 1976 Symp., Lecture Notes in Computer Science 45, Springer Verlag ed.

[Bert 81] G. Berthelot and R. Terrat, "Modelisation et validation de protocoles de Transport par réseaux de Petri à prédicats", Techn. Report (Sept. 1981), Institut de programmation, Université Curie, Paris.

[Buhr 81] R.J.A. Buhr and D.A. MacKinnon, "The Transport Laryer in OSI", Research report DOC-CR-CS-1980-0008 prepared for Department of Communications of Canada (1981).

[CCITT/ISOa] "Draft Connection-Oriented Transport Service", ISO TC97/SC16 N860 (1982).

[CCITT/ISOb] "Draft Connection-Oriented Basic Transport Protocol Specification", ISO TC97/SC16 N861 (1982).

[Chow 78] T.S. Chow, "Testing Software Design Modeled by Finite State Machines", IEEE Trans. Software Eng. SE-4, No. 3 (May 1978).

[Chung 82] R. Chung and G.V. Bochmann, "Principles and Application of a Formal Description Technique to Teletex Protocols", Techn. Report, in preparation.

[Chun 82b] R.J. Chung and G.V. Bochmann, "A Formal Specification of the Teletex Session and Document Procedures", Document de travail, Département d'informatique et de recherche opérationnelle, Université de Montréal, 1982.

[Devy 79] M. Devy, M. Diaz, "Multilevel Specification and Validation of the Control in Communication Systems", 1st Int'l. Conf. on Distributed Computing Systems, Alabama, October 1-4, 1979.

[Didi 80] M. Didic and B. Wolfinger, "Simulation of a Local Computer Network Architecture Applying a Unified Modeling System", Techn. report, Karlsruhe Nuclear Research Center, D-75 Karlsruhe.

[FDT 81] "A FDT Based on an Extended State Transition Model", working document (Dec. 1981) of Subgroup B of the ad hoc group on FDT of ISO TC97/SC16/WG1.

[FDT 81b] "Tutorial on Formal Description Techniques", Canadian contribution to the ISO TC97/SC16/WG1 ad hoc group on FDT, Jan. 1981; also annex to [Gagn 82].

[FDT 81c] "Interaction primitives in formal specification of distributed systems", working document (Sept. 1981) of Subgroup C of the ad hoc group on FDT of ISO TC97/SC16/WG1.

[FDT 82a] "Proposal on Different forms of FDT", Canadian contribution to CCITT SGVII Rapporteurs meeting on FDT, March 1982.

[Gagn 82] M. Gagné, "Un compilateur pour la traduction de spécifications de protocoles en Pascal", Document de travail 120, Département d'informatique et de recherche opérationnelle, Université de Montréal, Février 1982.

[ISO 81a] "Concepts for Describing the OSI Architecture" working document (Nov. 1981) of Subgroup A of the ad hoc group on FDT of ISO TC97/SC16/WG1.

[Jard 81] C. Jard and G.V. Bochmann, "An Approach to Testing Specifications", Publ. 430, Département d'information et de recherche opérationnelle, Université de Montréal, 1981.

[Koha 78] Z. Kohavi, "Switching and Finite Automata Theory", McGraw-Hill, 1978, Second Edition.

[Laca 82] C. Lacaille, Master's thesis, in preparation.

[Lam 81] S.S. Lam, A.V. Shankar, "Protocol Projections: A Method of Analyzing Communication Protocols", Proc. Nat. Telecomm. Conf. New Orleans, Dec. 1981, pp. E3.2.1-E.3.2.8.

[LeLa 78] G. LeLann and H. LeGoff, "Verification and Evaluation of Communication Protocols", Comput. Networks, vol. 2 pp. 50-69, Feb. 1978.

[LeFe 81] M. LeFevre and O. Rafic, "Pascal description of P machine", Projet RHIN, doc. FDT 750g (Sept. 1981), Agence d'informatique, France.

[Leve 82] A. Léveillé, Master's thesis, in preparation.

[McCo 81] W.H. McCoy, R.P. Colella, M.A. Wallace, "Assessing the Performance of High Level Computer Network Protocols", in Proc. INWG/NPL Workshop on "Protocl Testing-Towards Proof?, May 1981.

[Merl 82] P. Merlin and G.V. Bochmann, "On the Construction of Submodule Specifications and Communication Protocols", to be published in ACM TOPLAS.

[Nait 81] S. Naito, M. Tsunoyama, "Fault Detection for Sequential Machines by Transition Tours", Proc. IEEE Conf. on Fault Tolerance, 1981.

[NSa] "Formal Description of the Network Service", Document de travail Département d'informatique et de recherche opérationnelle, 1982.

[Piat 80] T.F. Piatkowski "Remarks on the Feasability of Validating and Testing ADCCP Implementations", Proc. Trends and

Applications (NBS), 1980, Gaithersburg, MD(USA).

[Rayn 81] D. Rayner "Protocol Implementation Assessment: Philosophy and Architecture", Proc. National Telecom. Conf. New Orleans, December 1981, pp. F8.4.1-5.

[Robi 79] L. Robinson, K.N. Levitt, and B.A. Silverberg "The HDM Handbook", vol. I-III.SRI Int. 1979.

[Tenn 81] R.L. Tenney and T.P. Blumer, "An Automated formal specification Technique for Protocols", Techn. Report (Jan. 1981), BBN, Cambridge, Mass.

[TPa] "Formal Specification of a Transport Protocol", Canadian contribution to the ISO TC97/SC16/WG1 ad hoc group on FDT, 1981.

[TPb,c] Annex 1 and Annex 2 of "Examples of Transport Protocol Specifications", Canadian contribution to CCITT SGVII Rapporteurs meeting on FDT, March 1982.

[TSa] "Formal Specification of a Transport Service", contribution to CCITT SGVII Rapporteurs meeting on FDT, Oct. 1981, (FDT-21).

[TSb] "Formal Specification of a Transport Service", contribution WASH-9 to ISO TC97/SC16/WG1 ad hoc group on FDT, Sept. 1981.

[Vogt 82] F. Vogt, HMI (Berlin), doctoral thesis, March 1982.

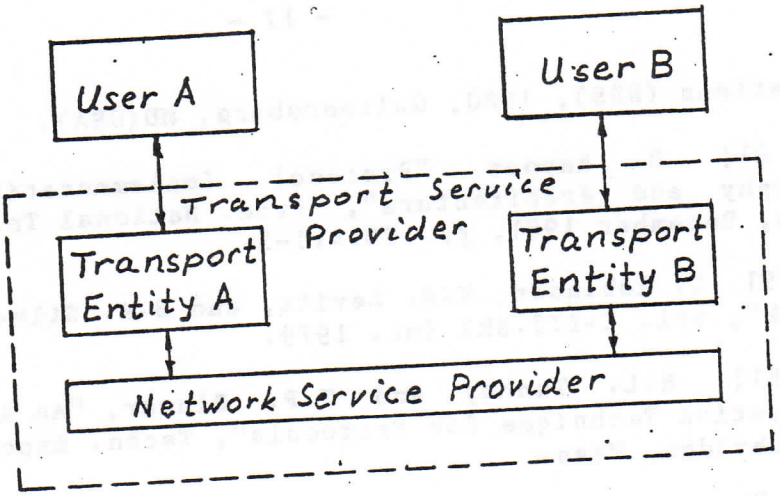
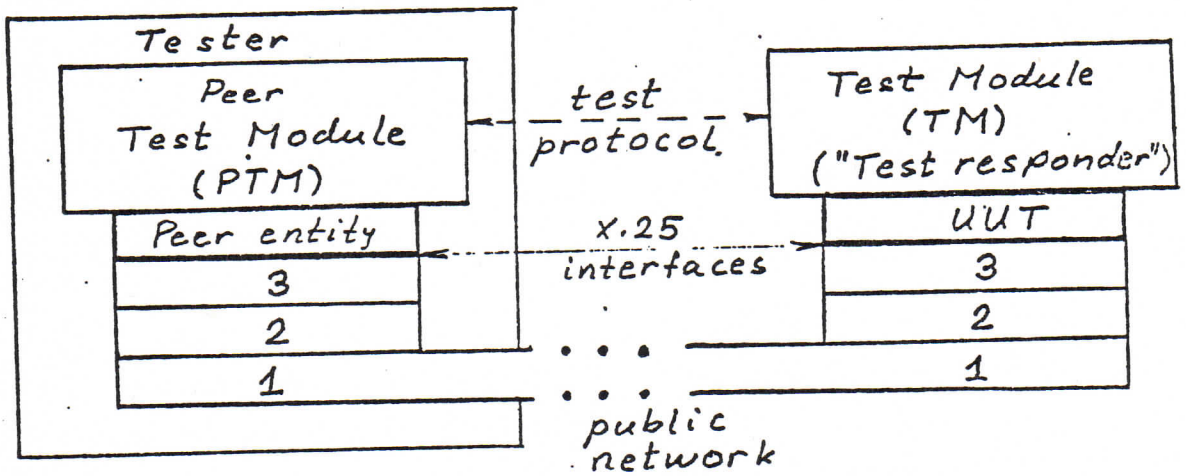


Figure 1: Relationship between Transport Service and Transport Protocol.



Peer entity: Peer Transport Implementation

UUT: Transport Implementation Under Test

Figure 2: Remote Test Architecture for the Transport Protocol Layer.