Experience with Formal Specifications Using an Extended State Transition Model

GREGOR V. BOCHMANN, EDUARD CERNY, MEMBER, IEEE, MICHEL GAGNÉ, CLAUDE JARD, ALAIN LÉVEILLÉ, STUDENT MEMBER, IEEE, CLÉMENT LACAILLE, MICHEL MAKSUD, MEMBER, IEEE, K. S. RAGHUNATHAN, AND BEHCET SARIKAYA, STUDENT MEMBER, IEEE

Abstract—**Experience with the use of formal descriptions of communication services and protocols is described. The paper focuses on the experience of 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, and further details may be found in the references.**

I. 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 using an appropriate formal description technique (FDT). Among the different FDT's that have been proposed and used (see, for example, [8] or [54]), we consider in this paper mainly the extended state transition model [26] 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 [4], [9], [14], [55].

The different activities during the design and implementation of protocols where formal specifications can be useful are summarized in [11]. The main activities are

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

2) the validation of the design of a protocol specification,

3) the design and development of a protocol implementation based on the protocol specification obtained under point 2), and

4) the validation of a protocol implementation obtained under point 3).

In this paper we discuss our experience with 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 II 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 II-E. Sections III, IV, and V deal with the activities 2), 3), and 4) 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.

II. EXAMPLE SPECIFICATIONS

A. 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 semiformal 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 [18] is a connection-oriented communication service that supports normal and expedited data transfer. Different classes of protocols [19] are defined, each providing a different set of functions. The available functions are

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

2) addressing of transport service access points (TSAP),

3) multiplexing,

4) error detection and (possibly) recovery,

Manuscript received March 3, 1982; revised July 27, 1982. This work was supported in part by the Natural Science Engineering Research Council of Canada and research contracts from the Department of Communication of Canada. This paper was presented in part at the Second International Workshop on Protocol Specification, Testing, and Verification, May 1982.

G. V. Bochmann, E. Cerny, A. Léveillé, C. Lacaille, M. Maksud, and K. S. Raghunathan are with the Department d' Informatique et de Recherche Operationnelle, Université de Montréal, Montréal, P.Q., H3C 3J7, Canada.

M. Gagné is with the National Research Council of Canada, Ottawa, Ont., Canada.

C. Jard is with the Department of Evaluation and Validation of Protocols, Centre Nationale d'Etudes des Télécommunications, Lannion, France.

B. Sarikaya is with the School of Computer Science, McGill University, Montréal, P.Q., Canada.

0090-6778/82/1200-2506$00.75

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

6) recovery from 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 is often restricted to the protocol classes 0 and 2.

B. 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 [57] is written in a local dialect [27], and was later rewritten according to the ISO syntax [58]. The purpose was to describe 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 [33, sect. 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 [57] or by incoming interaction [58].

Reference [58] gives a 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 possibility 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 nondeterminism of the FDT (see Section II-E1) below) is an essential feature for leaving certain implementation choices undefined.

Many different formal transport protocol specifications have been written using Pascal [39], Ada [17], extended Petri nets [3], and other methods [28], [61]. Space limitations prevent us from providing further references and comparisons.

C. 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 Fig. 1, dashed box). This approach has been taken for the specification of [59] which describes the properties of the transport service as seen by the users through the service access points. As in [58], an arbitrary number of simultaneous transport connections is considered. A simplified version, considering only a single connection and ignoring the problem of addressing, is given in [60] using a local dialect [27] of the FDT.

Many other transport service specifications have been written by different groups [3], [28], [61]. There is not enough space to discuss them all. However, we would like to mention here the question of whether it is useful to separate, in the service specification, the local and global [9] sequencing rules for the execution of service primitives. A general framework

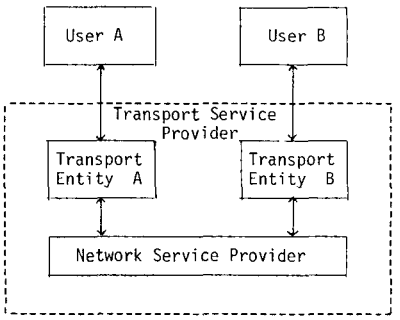


Fig. 1. Relationship between transport service and transport protocol.

for such a separation is given in [12]. While the state transition model seems adequate for the specification of the local rules [29], its use for the global rules may be questionable. While different specification languages (such as temporal logic [10], or abstract data types [41]) may be used 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.

D. 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 [5], X.25 [6], and the message link protocol [7] lies in this line. More recently, the NBS and DoD (USA) have funded the development of FDT's (similar to [26]) for use in their protocol development projects [53], [55]. The formal protocol specifications developed in this context are interesting examples.

Another example is some effort [22] for developing a formal specification of the Teletex control procedures. In this effort the Teletex session and document layers were described using the local FDT dialect [27]. 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 the reference to these service specifications. It may be interesting to note that the selection of the primitives for the document service was made in a somehow arbitrary manner. Some of these primitives are related to a document file store. Some kind of "virtual file store" was defined in a semiformal manner (see Section II-E5) below).

Finally, we would like to review briefly the specification [15] of a virtual file system developed by the Hahn-Meitner-Institute, Berlin. This 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 its 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 [26] seemed necessary for this example, as explained in the following section. 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 sublayers (above the transport service) which are specified separately.

E. Suitability of the FDT and Possible Extensions

We conclude from the above mentioned experience of writing formal specifications that the FDT of [26] 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 the most understandable presentation. Such decisions are sometimes quite arbitrary, often related to the personal tastes and prejudices of the person writing the specification. Consequently, some informal guidelines would be useful for this purpose. Reference [21] 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 features of the specification language and its use, and they point out some possible extensions.

1) Nondeterminism: The FDT of [26] is based on a model of a nondeterministic state machine. It is sometimes argued that nondeterministic behavior is not required, or desirable, for protocol specifications. We have found nondeterminism an important element of the specification language in the case of service specifications, where the relation between the interactions at the different service access points is not deterministic, as well as in the case of protocol specifications, where (in [58] 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) Incomplete Specifications: A specification of a protocol entity or a communication service usually makes 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" has occurred 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 nondeterminism. 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 possibility that the sequence of preceding interactions leads to a state of the specification for which there is no transition specified for the interaction under consideration [34]. We note that sometimes a different convention is used where an input interaction for which no transition is specified is ignored and is not necessarily considered an "unforeseen error."

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

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 value 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 0-functions in Special [50].

Cases a) and b) have been used in [46], [59], [58], and case c) would have been useful in the specification of the virtual file server [15], for defining the meaning of the POS primitive which positions a pointer in the hierarchical structure of a file.

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 [57], [58] or service [59], [60]. In the example of the Teletex document protocol [22], the "virtual file store"

mentioned above and a "document manager" were described along similar lines. Usually semiformal 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 in which formal descriptions based on the formalism of abstract data types (as developed for software engineering) may be useful [41]. It seems that abstract data type specifications could be combined with the extended state transition model; however, further research seems to be required in this area.

6) Liveness and Performance Issues: Most applications of finite state or extended state transition models do not consider performance issues, and liveness considerations are usually limited to showing absence of deadlocks and loops without progress [8], [54]. For defining the liveness properties of an extended state transition specification, certain transitions may be defined to be "live," where a live transition, if it is enabled, will eventually be executed unless some other transitions leads into a state where it is not enabled any more. A typical example is a time-out transition. Liveness properties of finite state machines are also considered in [23].

Performance considerations may be integrated into the extended state transition model by associating probabilities with the different transitions that are possible from a given state, and by defining a transition time for each type of transition, either a minimum and/or maximum value or a probabilistic time distribution [16], [44].

III. PROTOCOL DESIGN VALIDATION

The objective of protocol design validation (see, for example, [8]) is to verify that a given protocol specification for layer N, together with the given service specification for layer $(N-1)$, implies that the (N)-layer service is provided by the layered system architecture shown in Fig. 1.

A. Protocol Design Verification

Under this heading we consider static analysis of the specifications. The different approaches to verification are reviewed in [8], [54] where further references may be found. Techniques that are relevant for the extended state transition model are reachability analysis for finite state machines, invariant analysis for Petri nets [1], [3], 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 usually require some verification methods related to program proving (for example, assertions and invariants, symbolic execution, etc.; a simple example is discussed in [4]).

We are presently working on the verification of a class 0 transport protocol based on the specifications given in [46], [57], [58] and the standard mapping of PDU's into network service primitives. Globally, the verification proceeds through the following three steps.

1) The three modules shown in Fig. 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, [43] or [24], [48]). Special attention must be given to the interaction parameters and additional state variables.

2) From the viewpoint of the user service, the input-output interactions between the combined modules may be ignored. This view may be obtained by projections [43], or Petri net reductions [2], [24], [48]. In order to reduce the complexity of the problem it may also be useful to consider only one particular service property at a time, as explained in [37].

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 likely 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 [36].

B. 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 Fig. 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 [25], [38].

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 nondeterministic (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 [34].

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 implementation testing, as discussed in Section V.

IV. 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 much system dependent. In an implementation of the transport protocol on our PDP-11 computer [40], 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.

Partly automated translation of formal specifications into programs is also possible [30], [52], [55]. Usually the specifications are translated into some program elements (as described above) which call upon a system dependent run-time support package 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 [6]. 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.

V. 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 exist in several countries (see, for example, [49], or several papers in [31], [32]). 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 be 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 [13], the remote testing (Fig. 2) and a supplementary local tester (directly connected to the UUT) are currently receiving the 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 tester and an automatic tester. The following two subsections describe their objectives and general organization, and the last subsection explains an effort towards developing meaningful test sequences.

A. The Interactive Tester

Here, the objective is to provide a flexible tool destined mainly for debugging protocol implementations and, to a limited extent, 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 the response of the UUT for that input. Hence, the main function of the module is to create an easily useable interface for the human operator, freeing him from 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 V-B) could be used.

B. Automatic Remote Tester

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 Fig. 2), 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.

2) The Approach: In order to achieve the flexibility required by the objectives, we have opted for an organization

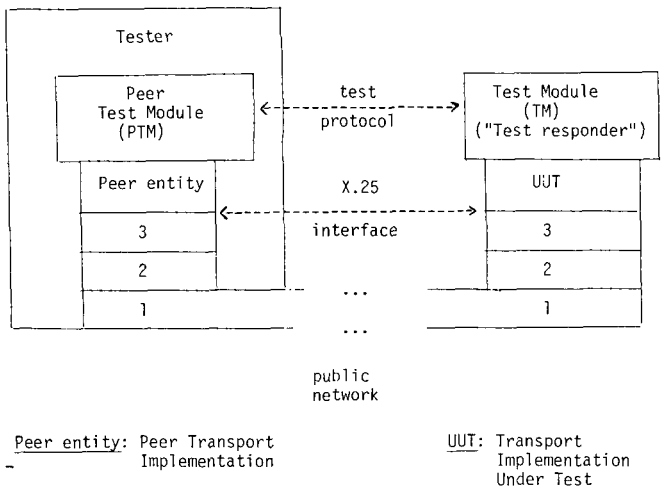


Fig. 2. Remote test architecture for the transport layer.

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 behavior of the TM and PTM is described using the FDT [26]. 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 downloading of detailed tests to the TM), and
- PDU mapping module.

In the case of the TM (passive responder) the modules are

- initial connection establishment test,
- test loader/TM protocol handler, and
- 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 the 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 [42].

C. 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 necessarily 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 behavior of the UUT under unspecified or erroneous inputs in order to obtain a complete characterization ("friendliness") of the implementation [13].

The techniques for deriving test sequences for protocols are, thus, an open research area. As starting points [13] could be considered the existing approaches in the area of microprocessor testing (e.g., [56]), machine identification [35], [45], and certain software testing techniques [20]. It may be necessary to test protocol implementations by functional submodules [47] 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 [51]. 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 of 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 [47].

VI. CONCLUSIONS

The discussions in the preceding sections show how a formal specification of communication services and protocols can be 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 [26], which is proposed to be used in the OSI environment, we feel that approaches similar to those described here would be useful in many other situations, including the design and implementation of nonstandard communication protocols, distributed application development, and modular system design in general.

REFERENCES

[1] P. Azema, B. Berthomieu, and P. Decitre, "The design and validation by Petri nets of a mechanism for the invocation of remote servers," in *Proc. IFIP Conf.*, pp. 599–604.

[2] G. Berthelot and G. Roucairol, "Reductions of Petri nets," in *Proc. MFCS 1976 Symp.* (Lecture Notes in Computer Science, vol. 45). New York: Springer-Verlag, 1976.

[3] G. Berthelot and R. Terrat, "Petri net theory for the correctness of protocols," this issue, pp. 2497–2505.

[4] G. V. Bochmann and J. Gecsei, "A unified model for the specification and verification of protocols," in *Proc. IFIP Cong.*, 1977, pp. 229–234.

[5] G. V. Bochmann and R. J. Chung, "A formalized specification of HDLC classes of procedures," in *Proc. Nat. Telecommun. Conf.*, Los Angeles, CA, Dec. 1977, Paper 3A.2.

[6] G. V. Bochmann and J. Tankoano, "Development and structure of an X.25 implementation," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 429–439, Sept. 1979.

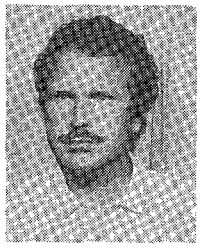
[7] G. V. Bochmann, "Formalized specification of the MLP," "Specification of the services provided by the MLP," and "An analysis of the MLP," Dep. d'Informatique et de Recherche

Operationnelle, Univ. de Montreal, Montreal, P.Q., Canada, June 1979.
[8] G. V. Bochmann and C. A. Sunshine, "Formal methods in communication protocol design," *IEEE Trans. Commun.*, vol. COM-28, pp. 624–631, Apr. 1980.
[9] G. V. Bochmann, "A general transition model for protocols and communication services," *IEEE Trans. Commun.*, COM-28, pp. 643–650, Apr. 1980.
[10] ——, "Concept for the specification of protocols and services," INWG Note; contrib. to ISO TC97/SC16/WG1 ad hoc group on FDT meet., Amsterdam, The Netherlands, 1980.
[11] ——, "The use of formal description techniques for OSI protocols," in *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, Dec. 1981, pp. F8.6.1–F8.6.6.
[12] G. V. Bochmann and M. Raynal, "Structured specification of communicating systems," *IEEE Trans. Comput.*, vol. C-32, Feb. 1983.
[13] G. V. Bochmann and E. Cerny, "Protocol assessment," Dendronic Decision Ltd., Montréal, P.Q., Canada, 1982, under contr. for Dep. Commun. Canada.
[14] G. V. Bochmann, *Concepts for Distributed Systems Design*. Springer-Verlag, to be published, chs. 8–10.
[15] G. V. Bochmann, L. Henckel, and R. P. Zeletin, "Formalized specification and analysis of a virtual file system," Hahn-Meitner-Institut, Berlin, West Germany, Tech. Rep. HMI-8307, Feb. 1982.
[16] T. L. Booth, "Performance optimization of software systems processing information sequences modeled by probabilistic languages," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 31–44, Jan. 1979.
[17] R. J. A. Buhr and D. A. MacKinnon, "The transport layer in OSI," Res. Rep. DOC-CR-CS-1980-0008, prepared for Dep. Commun. Canada, 1981.
[18] "Information processing systems—Open systems interconnection—Transport service definitions," ISO/DP 8072, 1982.
[19] "Information processing systems—Open systems interconnection—Transport protocol specification," ISO/DP 8073.
[20] T. S. Chow, "Testing software design modeled by finite state machines," *IEEE Trans. Software Eng.*, vol. SE-4, May 1978.
[21] R. Chung and G. V. Bochmann, "Principles and application of a formal description technique to Teletex protocols," Tech. Rep., in preparation.
[22] R. J. Chung and G. V. Bochmann, "A formal specification of the Teletex session and document procedures," Dep. d'Informatique et de Recherche Operationelle, Univ. de Montreal, Montreal, P.Q., Canada, Docu. de Travail, 1982.
[23] E. M. Clark and E. A. Emerson, "Design and synthesis of synchronisation skeletons using branching time temporal logic." Aiken Computation Lab., Harvard Univ., Cambridge, MA, Tech. Rep. TR-12-81.
[24] M. Devy and M. Diaz, "Multilevel specification and validation of the control in communication systems," in *Proc. 1st Int. Conf. Distributed Comput. Syst.*, AL, Oct. 1–4, 1979.
[25] M. Didic and B. Wolfinger, "Simulation of a local computer network architecture applying a unified modeling system," *Comput. Networks*, vol. 6, pp. 75–91, May 1982.
[26] "A FDT based on an extended state transition model," Working Doc. of Subgroup B of ad hoc group on FDT of ISO TC97/SC16/WG1, Dec. 1981.
[27] "Tutorial on formal description techniques," Canadian contrib. to ISO TC97/SC16/WG1 ad hoc group on FDT, Jan. 1981; also annex to [30].
[28] "Interaction primitives in formal specification of distributed systems," Working Doc. of Subgroup C of ad hoc group on FDT of ISO TC97/SC16/WG1, Sept. 1981; also "Temporal ordering specification..." in [32].
[29] "Proposal on different forms of FDT," Canadian contrib. to CCITT SGVII Rapporteurs meet. on FDT, Mar. 1982.
[30] M. Gagné, "Un compilateur pour la traduction de spécifications de protocoles en Pascal," Dép. d'I.R.O., Univ. de Montréal, Montréal, P.Q., Canada, Docu. de travail 120, Feb. 1982.
[31] *Proc. 1st Int. Workshop Protocol Specification, Testing, Verification* (IFIP WG 6.1). North-Holland, 1982.
[32] *Proc. of 2nd Int. Workshop Protocol Specification, Testing, Verification* (IFIP WG 6.1). North-Holland, 1982.
[33] "Concepts for describing the OSI architecture," Working Doc. of Subgroup A of ad hoc group on FDT of ISO TC97/SC16/WG1 Nov. 1981; also "Temporal ordering specification..." in [32].
[34] C. Jard and G. V. Bochmann, "An approach to testing specifications," Dep. d'Informatique et de Recherche operationnelle, Univ. de Montreal, Montreal, P.Q., Canada, Publ. 430, 1981.
[35] Z. Kohavi, *Switching and Finite Automata Theory*, 2nd ed. New York: McGraw-Hill, 1978.
[36] C. Lacaille, Master's thesis, Univ. de Montréal, Montréal, P.Q., Canada, in preparation.
[37] S. S. Lam and A. U. Shankar, "Protocol projections: A method of analyzing communication protocols," in *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, Dec. 1981, pp. E3.21–E.3.2.8.
[38] G. LeLann and H. LeGoff, "Verification and evaluation of communication protocols," *Comput. Networks*, vol. 2, pp. 50–69, Feb. 1978.
[39] M. LeFevre and O. Rafic, "Pascal description of *P* machine," Projet RHIN, Agence d'Informatique, France, Doc. FDT 750g, Sept. 1981.
[40] A. Léveillé, Master's thesis, Univ. de Montréal, Montréal, P.Q., Canada, in preparation.
[41] L. Logrippo, "Specification of transport service using finite-state transducers and abstract data types," contrib. to ISO TC97/SC16/WG1 ad hoc group on FDT, Apr. 1982.
[42] M. Maksud, Master's thesis, Univ. de Montréal, Montréal, P.Q., Canada, in preparation.
[43] P. Merlin and G. V. Bochmann, "On the construction of submodule specifications and communication protocols," *Ass. Comput Mach. TOPLAS*, to be published.
[44] M. K. Molloy, "On the integration of delay and throughput measures in distributed processing models," Ph.D. dissertation, Dep. Comput. Sci., Univ. California, Los Angeles, 1981.
[45] S. Naito and M. Tsunoyama, "Fault detection for sequential machines by transition tours," in *Proc. IEEE Conf. Fault Tolerance*, 1981.
[46] G. V. Bochmann, E. Cerny, and C. Lacaille, "Formal description of a network service," Dep. d'Informatique et de Recherche Operationnelle, Univ. de Montreal, Montreal, P.Q., Canada, Docu. de Travail, 1982.
[47] T. F. Piatkowski, "Remarks on the feasibility of validating and testing ADCCP implementations," in *Proc. Trends and Applications (NBS)*, Gaithersburg, MD, 1980.
[48] J. B. Postel and D. Fowler, "Graph modeling of computer communications protocols," in *Proc. 5th Texas Conf. Comput. Syst.*, Austin, TX, Oct. 1976.
[49] D. Rayner, "A system for testing protocol implementations," *Comput. Networks*, to be published.
[50] L. Robinson, K. N. Levitt, and B. A. Silverberg, *The HDM Handbook*. vols. I–III, SRI Int., 1979.
[51] B. Sarikaya and G. V. Bochmann, "Some experience with test sequence generation for protocols," in [32].
[52] G. Schultz, D. B. Rose, C. H. West and J. P. Gray, "Executable description and validation of SNA," *IEEE Trans. Commun.*, vol. COM-28, pp. 661–677, Apr. 1980.
[53] G. Simon and D. Kaufman, "An extended finite state machine approach to protocol specification," in [32].
[54] C. Sunshine, "Formal modeling of communication protocols," in *Computer Networks and Simulation*, S. Schoemaker, Ed. North-Holland, 1982.
[55] R. L. Tenney and T. P. Blumer, "An automated formal specification technique for protocols," *Comput. Networks*, vol. 6, pp. 201–217, July 1982.
[56] S. M. Thatte and J. A. Abraham, "Test generation for general microprocessor architectures," in *Proc. 9th Int. Symp. Fault Tolerant Comput.*, June 1979, pp. 203–210.
[57] "Formal specification of a transport protocol," Canadian contrib. to ISO TC97/SC16/WG1 ad hoc group on FDT, 1981.
[58] Annex 1 and Annex 2 of "Examples of transport protocol specifications," Canadian contrib. to CCITT SGVII Rapporteurs meet. on FDT, Mar. 1982.
[59] "Formal specification of a transport service," contrib. to CCITT SGVII Rapporteurs meet. on FDT, FDT-21, Oct. 1981.
[60] "Formal specification of a transport service," contrib. WASH-9 to ISO TC97/SC16/WG1 ad hoc group on FDT, Sept. 1981.
[61] F. Vogt, Ph.D. dissertation, HMI, Berlin, West Germany, Mar. 1982, also in [32].

Gregor V. Bochmann received the Diploma in physics from the University of Munich, Munich, Germany, in 1968, and the Ph.D. degree from McGill University, Montréal, P.Q., Canada, in 1971.

He has worked in the areas of programming languages and compiler designs, communication protocols, and software engineering. He is currently an Associate Professor in the Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal. His present work is aimed at design models for communication protocols and distributed systems. From 1977 to 1978 he was a Visiting Professor at the Ecole Polytechnique Federale, Lausanne, Switzerland. From 1979 to 1980 he was a Visiting Professor in the Computer Systems Laboratory, Stanford University, Stanford, CA.

★



Eduard Cerny (M'73) received the B.Sc. degree in engineering from Loyola College, Montréal, P.Q., Canada, in 1970 and the M. Eng. and Ph.D. degrees from McGill University, Montréal, in 1971 and 1975, respectively.

From 1971 to 1978 he was employed as a Lecturer and then as an Assistant Professor at Concordia University, Montréal. Presently he assumes the position of Associate Professor at the Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal. His interests are in digital systems design, fault-tolerant computing, and real-time computer applications.

Dr. Cerny is a Registered Engineer in the Province of Québec.

★

Michel Gagné received the B.Sc. degree in mathematics from the Université du Québec à Rimouski, Canada, in January 1978.

He is presently working at the National Research Council of Canada in Ottawa, Ont., Canada.

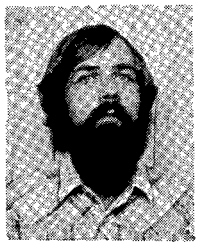
★

Claude Jard was born in Paris, France on April 12, 1959. He received the Engineer's Diploma of the Ecole Nationale Supérieure des Télécommunications de Bretagne, France, in 1981.

He currently works in the Département d'Evaluation et Validation de Protocoles at the Centre Nationale d'Etudes des Télécommunications, Lannion, France. His present activity is aimed at using simulation and testing specifications for the verification of protocols.

Alain Lêveillé (S'81) received the B.Sc. degree in computer science from the Université de Montréal, Montréal, P.Q., Canada, in 1980. He is currently working towards the M.Sc. degree in computer science on high-level protocols specification and implementation.

★



Clément Lacaille received the B.Sc. degree in 1979 and the M.Sc. degree in 1982, both in computer science, from the Université de Montréal, Montréal, P.Q., Canada.

His research interests are high level protocols and automatic systems assessment.

★

Michel Maksud (M'80) received the B.Sc. degree in computer science from the Université de Montréal, Montréal, P.Q., Canada, in 1979.

He is currently working on the development of interactive and automatic testers for high-level protocols. His interests include fault-tolerant computing and protocol specifications.

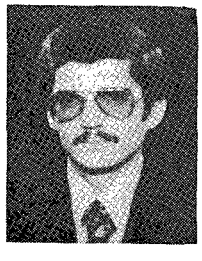
Mr. Maksud is a member of the Association for Computing Machinery.

★

K. S. Raghunathan received the B.E. (Hons.) degree in electronics and communication engineering from Madras University, Madras, India, in 1973, and the M. Tech. degree from the Indian Institute of Science, Bombay, India, in 1975.

He has since been working at Indian Telephone Industries, India, in the development of electronic switching systems. He is currently a Ph.D. student at the Département de'Informatique, Université de Montréal, Montréal, P.Q., Canada. His research interests include performance modeling and evaluation of distributed systems.

★



Behcet Sarikaya (S'80) received the B.Sc. degree in electrical engineering and the M.Sc. degree in computer science from the Middle East Technical University, Ankara, Turkey, in 1973 and 1976, respectively.

He is presently a Ph.D. student at the School of Computer Science, McGill University, Montréal, P.Q., Canada. His research interests include communication network protocols, testing theory, and software testing.

Mr. Sarikaya is a student member of the Association for Computing Machinery.