4º SBRC
RECIFE - 24 A 26 DE MARÇO 86

Recent developments in protocol specification, validation and testing*

Gregor v. Bochmann
Département d'IRO
Université de Montréal, Canada

Abstract:

The orderly introduction of new communication protocols, such as the standards for Open Systems Interconnection, requires a careful analysis of the proposed protocols and much effort for the development and testing of protocol implementations. Much research has been done recently in the area of formal description techniques (FDT) and their use for protocol design validation, implementation development and testing. This paper gives an introduction to these issues, and provides a review of recent research in the area. An attempt is made to explain the general direction of the work on formal protocol specifications, their validation, and conformance testing of protocol implementations. The use of formal specifications for deriving protocol implementations is also discussed.

1. Introduction

The development of distributed computer systems requires the establishment of communication protocols and their implementation in the different system components. Presently, a large standardization effort is under way with the purpose of establishing standardized communication protocols which allow the interworking of distributed computer application processes among heterogeneous computer systems, called Open Systems Interconnection (OSI) [OSI 83]. The orderly introduction of new communication protocols, for proprietary systems or Open Systems Interconnection, requires a careful analysis of the proposed protocols and much effort for the development and testing of protocol implementations.

Much research has been done recently in order to improve the working methods for these activities. In this context the use of formal description techniques (FDT) for the specification of communication protocols and services has received much attention, since such techniques allow a more systematic approach to protocol validation, implementation and testing as compared to the traditional use of protocol specifications given in natural language.

This paper gives an introduction to the issues of protocol specification, validation, implementation and testing, and gives a review of recent research on methods and tools in this area. Section 2 contains an explanation of the meaning of protocol and service specifications, the use of such specifications in the development and implementation of distributed computer systems, and a review of formal description techniques used for protocol and service specifications. Section 3 addresses the issues related to the validation of protocol specifications. Implementation issues are discussed in Sections 4 and 5. The development of protocol implementations based on the protocol specifications

* This work was supported by the Department of Communications Canada through research contract OST 85-00155.

is discussed in Section 4 with special emphasis on the use of formal protocol specifications. In Section 5, the issues related to the testing of protocol implementations are discussed. This includes in particular the verification that a protocol implementation conforms to the requirements defined by the protocol specification. This aspect is of particular interest in the context of OSI where systems developed by different manufacturers, possibly in different countries are expected to interwork in a compatible manner.

2. Specification methods

2.1. The meaning of service and protocol specifications

As in [Boch 80], we assume that the communication architecture of a distributed system is structured as a hierarchy of different protocol layers, for example as defined in the OSI Reference Model [OSI 83]. Each layer provides a particular set of Services to its users above. From their viewpoint, the layer, together with the layers below, may be seen as a "black box" or machine which allows a certain set of interactions with other users (see Fig. 1). A user is concerned with the nature of the service provided, but not with how the protocol manages to provide it.

This description of the input/output behavior of the protocol layer constitutes a Service Specification of the protocol. It should be "abstract" in the sense that it describes the types of commands and their effects, but leaves open the exact format and mechanisms for conveying them (e.g. procedure calls, system calls, interrupts, etc.). These formats and mechanisms may be different for users in different parts of the system, and are defined by an Interface Specification.

Service Specifications

Specifying the service to be provided by a layer of a distributed communication system presents problems similar to specifying any software module of a complex computer system. Therefore methods developed for software engineering are useful for the definition of communication services. Usually,

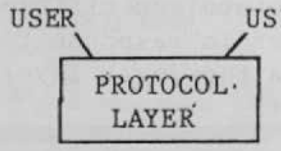

Fig. 1. Services provided by a protocol layer

a service specification is based on a set of Service Primitives which, in an abstract manner, describe the operations at the interface through which the service is provided. In the case of a transport service, for example, some basic service primitives are Connect, Disconnect, Send, and Receive. The execution of a service primitive is associated with the exchange of parameter values between the entities involved, i.e. the service providing and using entities of two adjacent layers. The possible parameter values and the direction of transfer must be defined for each parameter.

Clearly, the service primitives should not be executed in an arbitrary order and with arbitrary parameter values (within the range of possible values). At any given moment, the allowed primitives and parameter values depend on the preceding history of operations. The service specification must reflect these constraints by defining the allowed sequences of operations directly, or by making use of a "state" of the service which may be changed as a result of some operations.

In general, the constraints depend on previous operations by the same user ("local" constraints), and by other users ("global" constraints or "end-to-end properties"). Considering again the example of a transport service, a local constraint is the fact that Send and Receive may only be executed after a successful Connect. An example of a global constraint is the fact that the "message" parameter value of the first Receive on one side is equal to the message parameter value of the first Send on the other side.

Protocol Specifications

Although it is irrelevant to the user, the protocol designer must be concerned with the internal structure of a protocol layer. In a network environment with physically separated users, a protocol layer must be implemented in a distributed fashion, with Entities (processes or modules) local to each user communicating among one another via the services of the lower layer (see Fig. 2). The interaction among entities in providing a

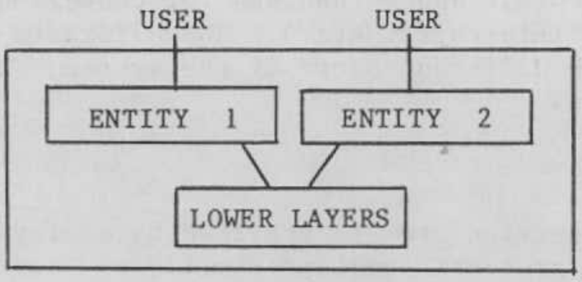

Fig. 2. Internal structure of a protocol layer.

layer´s service constitutes the actual·Protocol. Hence a protocol specification must describe the operation of each entity within a layer in response to commands from its users, messages from the other entities (via the lower layer service) and also internally initiated actions (e.g. timeouts)

Abstraction and Stepwise Refinement

The specifications described above must embody the key concept of Abstraction if they are to be successful. To be abstract, a specification must include the essential requirements that an object must satisfy and omit the unessential. A service specification is abstract primarily in the sense that it does not describe how the service is achieved (i.e., the interactions among its constituent entities), and secondarily in the sense that it defines only the general form of the interaction with its users (not the specific interface).

A protocol specification is a refinement or distributed "implementation" of its service specification in the sense that it partly defines how the service is provided (i.e., by a set of cooperating entities). This "implementation" of the service is what is usually meant by the design of a protocol layer. The protocol specification should define each entity to the degree necessary to ensure compatibility with the other entities of the layer, but no further. Each entity remains to be implemented in the more conventional sense of that term, typically by coding in a particular programming language. There may be several steps in this process until the lowest-level implementation of a given protocol layer is achieved, as shortly discussed in Section 4.

2.2. Review of specification methods

Various specification methods have been used for the definition of communication protocols and services. Besides informal methods, many more or less formal specification methods have developed and used in this area. In many cases, the specified system is modelled as a finite state machine [Boch 78h, Zafi 80, Chow 84] or a Petri net [Diaz 82], or in some more general state transition formalism which allows the consideration of the parameter values of exchanged interactions. In other cases, high-level programming languages have been used for defining protocol behavior. Other methods, which are less "state-oriented", define the system behavior by specifying allowed interaction sequences by using various kinds of sequencing expressions or grammar formalisms. A review of specification techniques for communication protocols and services can be found in [Boch 80]. Much recent work is presented in [Sun 82, Rudi 83, Yemi 84]. A tutorial on protocol specification and validation is given in [Rudi 85].

Most of the specification methods were originally developed in a different or more general context. This is also the case for algebraic specification methods (see for example [Miln 80]) some of which are related to abstract data type definitions [Guta 78, Logr 82]. This discussion cannot be considered complete without mentioning the application of temporal logic specification methods to the protocol area [Schw 82, Schw 83], and the use of logic programming techniques as specification language or as support for building validation tools [Sidh 83, Ural 84, Boch 85].

It is generally desired that the specification methods developed for protocol specifications also be applicable for the specification of communication services. For example, a simple finite state transition formalism which captures an important part of most protocol specifications is less useful for the specification of communication services, since the service interactions with two different users (see Figure 1) are usually not closely synchronized with one another. Instead, most specifications of communication services involve some kind of queues which correspond to the propagation of information from one service user to the other.

Standardization

In recent years, the use of formal methods for the description and validation of communication protocols has been a topic for discussion of standardization bodies, such as ISO and CCITT. In the context of the OSI standardization effort [OSI 83] some work on the standardization of formal description methods, so-called "formal description techniques" (FDT) has been persued [Viss 83, Dick 83]. Presently, three FDT´s are under consideration by ISO and CCITT for application to the specification of OSI protocols and services. These methods are the following (in alphabetical order):

(a) Estelle [Este 85] is developed by Subgroup B of the ISO TC97/SC21/WG1 ad hoc group on FDT. It is a method based on a finite state machine model extended by the use of Pascal programming language elements to handle data structures and more complex operations. A specified system may consist of a larger number of interconnected state machine modules.

(b) Lotos [Loto 85] is developed by Subgroup C of the ISO TC97/SC21/WG1 ad hoc group on FDT. It is a method based on the formalism of Milner's CCS [Miln 80] which is combined with an abstract data type definition facility, called ACT ONE [ACT ONE]. Similar to the other methods, it allows the construction of a specification from several smaller components.

(c) SDL [SDL 84] was originally developed by CCITT for the description of switching systems. It has also been found useful for the description of communication protocols [Dick 83]. The method is based on an extended finite state machine model, and is largely oriented towards a graphical representation.

Many trial specifications of various communication protocol standards in the OSI area have been developed using these methods. The possibility has been considered that such a formal protocol specification could in the future represent the protocol standard.

2.3. The use of formal specifications

Most communication protocol and service specifications today are given in an informal manner using natural language, sometimes complemented by semi-formal tables or diagrams. It is important to note that the "official" authoritative specification of a protocol or service is used for the different activities discussed below. If a formal specification can be used as the authoritative definition, many aspects of these activities may be automated, as discussed in the following sections. Protocol and service specifications are used for the following purposes:

(a) For the validation of the service and protocol designs: In this area the methods described in Section 3 may be applied.

(b) For the development of an implementation of the protocol: as noted above, one major goal of protocol specification is to provide a basis for implementation. Some specification methods facilitate this goal more than others. Programming language specifications may be quite close to implementations (but often lack the desired degree of abstraction). Direct implementation of "state machine" specifications by some form of translation into a programming language is also relatively straightforward [Boch 79]. In many cases, these implementation methods have been at least partially automated [Goud 76, Blum 82, Boch 85d, Kaji 85, Ausa 83].

(c) For validating an implementation of the protocol, and in particular, for assessing that the implementation conforms to the protocol specification: This activity, also called "conformance testing", has received much attention recently, in particular in relation with the development of OSI protocol standards [OSI C, IHLPS 84, IHLPS 85].

3. Protocol design validation

3.1. General considerations

As explained in [Boch 80] validation is essentially a demonstration that an object meets its specifications. Recalling from Section 2.1 that Services and Protocol Entities are the two major classes of objects requiring specification for a protocol layer, we see there are two basic validation problems that must be addressed: (1) the protocol's design must be verified by analyzing the possible interactions of the entities of the layer, each functioning according to its (abstract) protocol specification and communicating through the underlying layer's service, to see whether this combined operation satisfies the layer's service specification; and (2) the implementation of each protocol entity must be verified against its abstract protocol specification.

The somewhat ambiguous term "protocol verification" is usually intended to mean this first problem, also called "protocol design validation". Because protocols are inherently systems of concurrent independent entities interacting via (possibly unreliable) exchange of messages, validation of protocol designs takes on a characteristic communication oriented flavor. The validation of a protocol implementation, usually done by "ordinary" programming techniques, is often called "protocol implementation assessment", and is further discussed in Section 5.

The service specification itself cannot be verified, but rather forms the reference against which the protocol is verified. However, the service specification can be checked for consistency. It must also properly reflect the users' desires, and provide an adequate basis for the higher levels which use it. Unfortunately, techniques to achieve these latter goals are still poorly understood.

It is important to note that protocol design validation also depends on the properties of the lower-layer protocol. In verifying that a protocol meets its service specification, it will be necessary to assume the properties of the lower layer's service. If a protocol fails to meet its service specification, the problem may rest either in the protocol itself, or in the service provided by the lower layer.

The overall validation problem may be divided into two categories, general and service specific properties.

General properties are those properties common to all protocols that may be considered to form an implicit part of all service specifications. Foremost among these is the absence of deadlock (the arrival in some system state or set of states from which there is no exit). Completeness, or the provision for all possible inputs, is another general property. Progress or termination may also be considered in this category since they require minimal specification of what constitutes "useful" activity or the desired final state.

Specific properties of the protocol, on the other hand, require specification of the particular service to be provided. Examples include reliable data transfer in a transport protocol, copying a file in a file transfer protocol, or clearing a terminal display in a virtual terminal protocol. Definitions of these features make up the bulk of service specifications.

3.2. Validation methods

The methods for protocol design validation are usually closely related to the specification methods used for the definition of the communication protocol and services. Many results have been obtained with simple state transition models since they are relatively easy to handle. However, they usually concern only a simplified model of the protocol and tend to lead to a very large number of cases (states) to be considered. More powerful specification techniques often allow for more powerful verification methods, however, they are often difficult to apply. Recent work in the area of protocol design validation can be found in [Sun82, Rudi 83, Yemi 84].

In this area, the distinction between the logical analysis and analysis by simulation and testing is useful. A validation method following the "logical analysis" paradigm determines through some logical reasoning that the protocol specification has certain required properties. The best known method of logical analysis used in protocol design validation are
(a) reachability analysis, applied mainly with specifications given in the form of finite state machines (FSM),
(b) derivation of invariants about reachable states based on the structure of specifications given in the form of Petri nets,
(c) inductive proof techniques involving assertions about states and interaction parameters, which are mainly applied when the protocol is given in some kind of programming language (these methods were originally developed for the verification of programs), and
(d) symbolic execution which, similar to program proving, handles program-like specifications, and follows all possible execution branches analytically (it is in that sense similar to reachability analysis, but like program proving requires the proof of assertions on program variables and interaction parameters).

The logical analysis of performance questions is usually called "analytic performance analysis". Performance analysis usually requires the establishment of some kind of performance model. Some experience with the inclusion of performance parameters in the formal specifications of the qualitative properties in an extended FSM model are reported in [Boch 84]. Such integrated specifications make it possible to link the performance model directly to the qualitative specification of the system, and simplifies the overall system specification. This idea is not new; performance parameters have often been associated with FSM and Petri-net models. Some new results of performance analysis in this area are reported in [Rudi 83b, Razo 84].

In contrast to logical analysis, the methods of simulation and testing provide results which are less definite. The absense of design errors cannot be guaranteed (it can only be stated that no error was found), and performance results are often approximate. On the other hand, these methods are usually easier to be applied, at least in the case where the specifications are relatively complex. One of the first applications of this method is described in [LeLa 78]. More extensive performance simulation studies for protocol systems involving several layers were described in [Wolf 82]. The use of simulation for validating the qualitative properties of protocols is emphasized in [Jard 83, Prob 84]. The use of a specification technique which allows the writing of "executable specifications" is an important consideration for these applications.

Many published papers on protocol verification present some particular verification technique, and demonstrate this technique by discussing its application to a simple protocol of more or less academic nature. Some papers treat more complex, realistic protocol examples. Some a posteriori verifications of protocol standards or standardization proposals have also been presented pointing out certain difficulties with the adopted or proposed procedures. Most of these verification efforts were based on a state reachability analysis, and in some cases automated systems were used. The results have had some influence on the standardization process, and have probably influenced the implementation of these procedures.

4. Implementation

As mentioned in Section 2.1, a protocol specification usually defines only those aspects of the behavior of a protocol entity which are required for compatibility with the other entities within the given protocol layer. This means that many aspects which characterize a protocol implementation must be defined by the protocol implementor; these implementation choices include the determination of the interface through which the communication service is provided to the users and other aspects which determine the performance and usability of the protocol implementation.

This situation applies not only when informal protocol specifications are used, but also when the implementation is based on a formal specification of the protocol which models precisely the authoritative definition. Formal specifications, based on the formal nature of their language, lend themselves to automated implementation (see references in Section 2.3). However, only part of the code of an implementation is obtained automatically, since many implementation choices are not defined by the formal specification, and general-purpose support routines are generally needed for user interfaces, buffer management and timer functions.

For the development of protocol implementations, a formal specification may be useful for the following reasons:

(a) A formal specification is normally already structured in such a way that the development of an implementation in a programming language based on the formal specification is easier than the development of an implementation based on the informal protocol specification.

(b) In many cases, a relatively straightforward transformation from the formal specification to the implemented program is possible [Boch 79, Serr 85].

(c) Part of the implementation may be obtained automatically by the use of an FDT compiler [Blum 82, Gerb 83].

It is our experience that in the case of an implementation based on a formal specification, the main implementation effort is involved with the development and implementation of the interfaces with the operating system and users, and buffer management. The development of the procedures for coding and decoding PDU´s usually also require much effort, unless the protocol uses some standard coding scheme, such as ASN1 [ASN1] for which these functions may be obtained through automated tools.

An example of a Transport protocol implementation based on a formal specification and the use of an FDT compiler is described in [Serre 86]. The proposed implementation methodology involves the following phases of implementation refinement:

(1) The starting point for the implementation is a formal specification of the protocol, called "reference specification", which is assumed to be the authoritative definition of the protocol. (Note that in many cases, the reference specification must be checked against the authoritative definition given in an informal manner).

(2) A so-called "detailed protocol specification" is elaborated which makes many of the implementation choices mentioned above, but which leaves those choices open which are related to the run-time environment under which the protocol implementation is to operate. This specification is written in the same FDT as the reference implementation, and is obtained from the latter by careful additions and transformations. In the example considered [Serre 86], the detailed specification provides a refined user interface, defines the reaction of the protocol entity for certain error situations, and reduces the non-determinacy related to PDU concatenation, acknowledgements, and the management of underlying network connections.

(3) In the last phase, the detailed specification is adapted to the particular run-time environment, and the program code of the implementation is generated. [Serre 86] contains a comparison of manual code development and the use of an FDT compiler. As mentioned above, only part of the code can be directly generated by such a compiler. During this phase, the issues of buffer management, process structure, and the details of the user interface must be added.

It is important to note that the same "detailed protocol specification", as defined above, may be useful for a large number of different implementation projects.

5. Conformance testing

The validation of a protocol implementation is usually performed mainly through testing. Two concerns must be distinguished in this context:

- **protocol conformance assessment** is concerned with the question whether a given protocol implementation conforms to all rules defined in the corresponding protocol specification, and which of the defined options are supported by the implementation.

- **implementation assessment**, in a more general context, is concerned in addition with other properties of the implementation which are not specified by the protocol specification, such as "how does the implementation react to unexpected (invalid) user interactions?", "how many simultaneous connections can be supported?", or "what is the performance of the implementation?".

Possible architectures for the testing of higher-level communication protocols have been elaborated by the standardization committees to be used for the conformance testing of OSI protocol implementations [OSI C]. The most commonly used architecture [Rayn 82] foresees so-called "remote" (or "distributed") testing where the test system, also called "lower tester", acts as the peer protocol entity of the implementation under test (IUT), as shown in Figure 3. It is important to note that the complete testing of the IUT requires the presence of a test user, also called "upper tester", which verifies that the executed service primitives relate correctly to the protocol data units exchanged between the IUT and the remote test system.

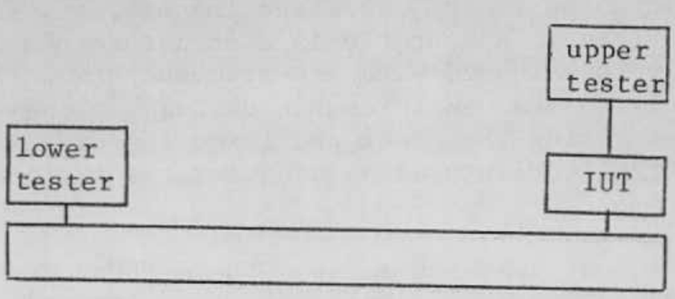

Figure 3

The remote testing architecture has been widely adopted for the assessment of higher-level OSI protocols. There are, however, the following difficulties associated with this approach. First, the presence of an intermediate network makes it difficult to use the architecture for performance measurements, and for testing the implementation under test (IUT) for cases of certain network errors. A portable test unit is proposed to solve these problems [Rayn82, Ansa 84]. Second, the architecture (as well as the portable test unit) assumes that there is no direct synchronization between the lower tester and the upper tester. Several variations of this basic testing architecture have been proposed [Boch 83f, Rafi 85]. These variations address in particular the question of how the operations of the lower and upper tester may be synchronized. It has been noted that certain test sequences present synchronization problems [Sari 83, Sari 84], and that therefore certain protocol functions may be difficult to test.

While in the past, most issues in the protocol conformance area have been resolved by pragmatic considerations, formal methods seem to be useful in this area for the following purposes:
(1) Formal specification methods may be used for specifying test sequences.
(2) The formal specification of the protocol standard may be used as the basis for deriving test sequences [Sari 84, Sari 85b or 85].
(3) The observed sequence of interactions of an implementation under test may be checked against the formal protocol specification in order to detect any deviation from the protocol specification [Jard 83b, Ural 84, Boch 85].

To answer the question in which form test sequences should be specified, the following issues must be considered:

(a) Due to the distributed test architecture, there are two sides that participate in the definition of the applied test sequences: the upper and lower testers. There seem to be the following two approaches to the specification of test sequences:
(1) Separate specifications for the upper and lower tester are given for each test case.
(2) The possible expected interactions of the IUT during the test, including test input and the allowed outputs generated by the IUT, are specified in a closed form.

(b) Some aspects of a protocol specification usually allow certain freedom in choices made by the implementation (sometimes called "non-determinism"). Therefore, it is not always known, when the test sequences are designed, in which way the IUT will respond to a given test input. Depending on the observed output from the IUT, different subsequent test inputs may be required. This presents a problem to the specification of tests sequences by scenarios. (In the simplest case, a scenario is a predetermined sequence of input interactions, possibly interleaved with expected outputs). In order to be able to adapt conveniently to the unpredictable choices of the IUT, test cases may be specified by describing the upper and lower testers as processes [Boch 83f]; the same specification language as used for the protocol specification may be used.

Concerning the selection of test sequences, the methods developed for software [Howd 78, Chow 78] and hardware [That 79, Nait 81] testing can be largely adapted to the area of protocol testing. The use of techniques developed for systems described as FSM is described in [Sari 82, Ansa 84]. However, these techniques do not handle the more complex aspects of interaction parameters and additional program variables.

Most testing methods for hardware [That 79] assume a certain fault model for the IUT. It is not clear what kind of fault model could be assumed for protocols, or design errors, in general. A simple example of a fault model about the coding and decoding function of a protocol is described in [Sari 84b].

While many software testing techniques are based on the knowledge of the structure of the tested program, in the case of protocol assessment the IUT must be considered in general as a black box. However, the software engineering methods based on the program structure can be applied by guiding the test case selection by the structure of the protocol specification.

Assuming as objectives for the selected test sequences the coverage of all branches in all transitions of the protocol specification (which is assumed to be given in an extended FSM model), and the coverage of all data flows between parameters of input and output interactions, as well as variables of the protocol specification, a test design methodology is described in [Sari 84b] which takes into account the possible parameter values of interaction primitives. It is interesting to note that this approach also allows to partially formalize the decomposition of a protocol specification into a number of "functional" blocks. An ad hoc functional decomposition was the basis for the test sequences described in [Boch 83f], which turn out to be similar to those obtained by the systematic application of the test methodology described in [Sari 84b]. The result of applying these tests to two implementations of the Transport protocol is described in [Cern 84].

The question whether an IUT satisfies the requirements of the protocol specification is usually solved by including the expected behavior of the IUT in the specification of the test sequences (e.g. by including expected outputs in test scenarios). The approaches present the disadvantage that it is usually quite difficult to derive the necessary elements of the test specification from the reference specification of the protocol (on which such decisions should be based).

An alternate approach to deciding the conformance is the use of an "arbiter" (sometimes called "oracle") which determines whether the sequence of observed input and output interactions in which the IUT was involved is a possible sequence according to the protocol specification [Jard 83, Prob 84]. A method for deriving such an arbiter from the reference specification of the protocol is described in [Jard 83] for the case of an extended FSM model, including the possibility of non-determinism. The construction of such arbiters is complicated by the fact that the protocol specification may allow for non-deterministic behavior.

6. Conclusions

Much work on formal description techniques and their use for protocol design, implementation and testing has been performed in recent years. It seems that some of these methods have advanced enough to make them usable in the design and implementation of real systems involving real-life protocols, including standards such as the OSI protocols being developed. Many of these real-life protocols are sufficiently complex to warrent the use of automated tools. These tools may be used for some of the work related to the validation of the design, the development of implementations, or the testing of the protocols implementations. Experience with certain tools of this kind has already been reported in the literature. It can be expected that such tools will be improved in the near future in order to make their application more simple and efficient.

References

[ACT ONE] H.Ehrig and B.Mahr, Fundamentals of Algebraic Specifications 1, Springer Verlag, 1985.

[ASN1] ISO TC97/SC21, DP...., 1985, "Specification of Abstract Syntax Notation One".

[ASN1] ISO TC97/SC21, DP...., 1985, "Specification of Abstract Syntax Notation One".

[Ansa 83] J.P. Ansart, V. Chari, and D. Simon, "From formal description to automated implementation using PDIL" in Protocol Specification, Testing and Verification (IFIP/WG6.1), H.Rudin and C.H.West, eds., North Holland, 1983.

[Blum 82] T.P. Blumer and R. Tenney, "A formal specification technique and implementation method for protocols", Computer Networks 6,3 (July 1982), pp. 201-217.

[Boch 78g] G.v. Bochmann, "Finite state description of communication protocols", Computer Networks 2 (1978), pp. 361-372, reprinted in "Communication Protocol Modeling", edited by C. Sunshine, Artech House Publ., 1981.

[Boch 79] G.v. Bochmann and J. Tankoano, "Development and structure of an X.25 implementation", IEEE Tr. SE-5, No. 5 (Sept. 1979), pp. 429-439, reprinted in "Communication Protocol Modeling", edited by C. Sunshine, Artech House Publ., 1981.

[Boch 80] G.v. Bochmann, C.A. Sunshine, (invited paper) "Formal methods in communication protocol design", IEEE Tr. COM-28, No. 4 (April 1980), pp. 624-631, reprinted in "Communication Protocol Modeling", edited by C. Sunshine, Artech House Publ., 1981.

[Boch 83f] G.v. Bochmann, E. Cerny, M. Maksud, B. Sarikaya, "Testing of Transport protocol implementations", Proc. CIPS Conference, Ottawa, 1983, pp.123-129.

[Boch 84] G.v. Bochmann, "Performance statements in Subgroup B specifications", Report for DOC research contract oST83-00082, CERBO Informatique Inc., Feb. 1984.

[Boch 85] G.v.Bochmann, R.Dssouli, W.Lopes de Souza, B.Sarikaya, H.Ural, "Use of Prolog for building protocol design tools", Proc. 5-th IFIP Workshop on Protocol Specification, Verification and Testing, Toulouse, June 1985.

[Boch 85d] G.v.Bochmann, G.Gerber, J.M.Serre, "Semi-automatic implementation of communication protocols", Proc. CIPS Congress 1985, Montreal, pp.

[Cern 84] E.Cerny, G.v.Bochmann, M.Maksud, A.Leveille, J.M.Serre, and B.Sarikaya, "Experiments in testing communication protocol implementations", Proc. FTCS ´84, IEEE, pp.

[Chow 78] T.S.Chow, "Testing design modelled by finite-state machines", IEEE Trans. SE-4, 3, 1978.

[Chow 84] C.H. Chow, M.G. Gouda, and S.S. Lam, "An Exercice in constructing Multi-phase Communication Protocols", Proc. Communications Architectures and Protocols, (ACM SIGCOMM), Montreal, June 1984, pp. 42-49.

[Diaz 82] M. Diaz, "Modelling and Analysis of Communication and Cooperation Protocols using Petri Net based Models", Computer Networks, Vol. 6, no. 6, Dec. 82, pp. 419-441

[Dick 83] G.J. Dickson, and P. de Chazal, "Application of the CCITT SDL to protocol specification", IEEE Trans. COM, to be published.

[Este 85] ISO DP 9074 (May 1985) "Estelle: A formal description technique based on an extended state transition model".

[Gerb 83] G. Gerber, "Une methode d´implantation automatisee de systemes specifies formellement", MSC thesis, Univ. of Montreal, 1983.

[Gerb 83] G. Gerber, "Une methode d´implantation automatisee de systemes specifies formellement", MSC thesis, Univ. of Montreal, 1983.

[Guta 78] J.V.Gutag, E.Horowitz and D.R.Musser, "Abstract data types and software validation", Comm.ACM, 21 (Dec. 1978).

[Howd 78] W.E. Howden, "A survey of dynamic analysis methods", in Software Testing and Validation Techniques, E. Miller and W.E. Howden eds., IEEE EHD 138-8, 1978.

[IHLPS 84] Proceedings of Conference on Introduction of High Level Protocol Standards for OSI (Department of Communications), Ottawa, May 1984.

[IHLPS 84] Proceedings of Conference on Introduction of High Level Protocol Standards for OSI (Department of Communications), Ottawa, May 1984.

[Jard 83b] C.Jard and G.v.Bochmann, "An approach to testing specifications", Journal of Systems and Software, Vol.3, 4(Dec. 1983), pp.315-323.

[Kaji 85] M.Kajiwara, H.Ichikawa, M.Itoh, and Y.Yoshida, "Specification and verification of switching software", IEEE Tr. Comm. COM-33 (1985).

[LeLa 78] G.Le Lann and H.LeGoff, "Verification and evaluation of communication protocols", Computer Networks 2, 1 (Febr. 1978), pp. 50-69.

[Logr 82] L. Logrippo, "Specification of Transport service using finite-state transducers and abstract data types", CCITT Q39/VII,FDT-77, Geneva, Dec. 1982.

[Loto 85] ISO DP 8807 (1985), "LOTOS: a formal description technique".

[Miln 80] R.Milner, "A calculus of communicating systems", Lecture Notes in CS, No. 92, Springer Verlag, 1980.

[Nait 81] S.Naito and M.Tsunoyama, "Fault detection for sequenctial machines by transition tours", Proc. FTCS, 1981, pp. 238-243.

[OSI 83] Special issue on Open Systems Interworking, Proc. of the IEEE, Dec. 1983.

[OSI C] ISO TC97/SC21/WG1 ad hoc group on conformance testing, 1985.

[OSI C] ISO TC97/SC21/WG1 ad hoc group on conformance testing, 1985.

[Rafi 85] D. Rafiq, R. Castanet, C. Chraibi, j.P. Goursaud, J. Haddad, X. Perdu, "Towards and environment for testing OSI protocols", IFIP Workshop on Protocol Specification, Verification and Testing (Toulouse, 1985), North Holland.

[Rayn 82] D. Rayner, "A system for testing protocol implementations", Computer Networks 6,6 (Dec. 1982).

[Razo 84] R.R.Razouk, "The derivation of performance expressions for communication protocols from timed Petri net models", Proc. ACM SIGCOMM Symposium, 1984.

[Rudi 83] Proceedings of 3-rd Workshop on "Protocol specification, Testing and verification" (IFIP/WG 6.1), H. Rudin and C.H. West, eds., North Holland, 1983.

[Rudi 83b] H.Rudin, "From formal protocol specification towards performance prediction", in Protocol Specification, Testing and Verification, H.Rudin and C.West (eds.), North Holland Publ. Comp., 1983.

[Rudi 85] H.Rudin, "An informal overview of formal protocol specification", IEEE Communication Magazine, Vol. 23,3 (March 1985).

[Rudi 85] H.Rudin, "An informal overview of formal protocol specification", IEEE Communication Magazine, Vol. 23,3 (March 1985).

[Sari 82] B. Sarikaya and G.v. Bochmann, "Some experience with test sequence generation for protocols", Proc. 2-nd Int. Workshop on Protocol Specification, Testing and Verification, North Holland, 1982, pp. 555-567.

on Protocol Specification, Testing and Verification, (ed. C. Sunshine), North Holland 1982.

[Schw 83] R.L. Schwartz, P.M. Melliar-Smith, F.H. Vogt, "An interval logic for higher-level temporal reasoning", Proc. ACM Symp. on Princ. of Distr. Computing, Montreal, Aug. 1983, pp. 173-186.

[Serr 85] J.M.Serre, "Methodologie d'implantation du protocole Transport classe 0/2", MSc thesis, Dept. d'IRO, Universite de Montreal, 1985.

[Serr 86] J.M.Serre, E.Cerny, G.v.Bochmann, "A methodology for implementing high-level communication protocols", Proc. 19-th Hawai Int. Conf. on Systems Sciences, Jan. 1986.

[Sidh 83] D.P.Sidhu, "Protocol verification via executable logic specifications", Proc. IFIP Workshop on Protocol Specification, Verification and Testing, North-Holland Ed., pp. 237-248 (1983).

[Suns 82] Proceedings of 2-nd Workshop on "Protocol specification, Testing and verification" (IFIP/WG 6.1), C. Sunshine, ed., North Holland, 1982.

[That 79] S.M.Thatte and J.A.Abraham, "Test generation for general microprocessor architectures", Proc. 9-th Symp. on Fault-Tolerant Commputing, June 1979, pp. 203-210.

[Ural 84] H.Ural and R.L.Probert, "Automated testing of protocol specifications and their implementations", Proc. ACM SIGCOMM Symposium, 1984.

[Viss 83] C.A.Vissers, G.v. Bochmann and R.L.Tenney, "Formal description techniques by ISO/TC97/SC16/WG1 ad hoc group on FDT", Proceedings of the IEEE, vol. 71, 12, pp. 1356-1364, Dec. 1983.

[Wolf 82] B.Wolfinger and O.Drobnik, "Simulation of protocol layers of communication in computer networks", in Computer Networks and Simulation II, S.Schoemaker (ed.), North Holland Publ. Comp., 1982.

[Yemi 84] Proceedings of 4-th Workshop on Protocol specification, validation and testing, IFIP, Y.Yemini ed., North-Holland, 1984.

[Sari 83] B. Sarikaya and G.v. Bochmann, "Synchronization issues in protocol testing", Proc. ACM SIGCOMM Symposium, Austin, 1983, pp.121-128.

[Sari 84] B. Sarikaya and G.v. Bochmann, "Synchronization and specification issues in protocol testing", IEEE Trans. on Comm., COM-32, No.4 (April 1984), pp. 389-395.

[Sari 84b] B.Sarikaya, PhD thesis, McGill University, Montreal, March 1984.

[Sari 85b] B.Sarikaya, G.v.Bochmann, and E.Cerny, "A test design methodology for protocol testing", submitted for publication.

[Schw 82] R. Schwartz and P.M. Melliar-Smith, "From state machines to temporal logic: specification methods for protocol standards", IFIP Workshop