NOTES ON AUTOMATED PROTOCOL ANALYSIS

H. Rudin

Rapporteurs: Dr. S.K. Shrivastava
Mr. D.H. Mundy

NOTES ON AUTOMATED PROTOCOL ANALYSIS

Harry Rudin
IBM Zurich Research Laboratory
8803 Ruschlikon, Switzerland
Telephone- 01 - 724 27 27
Telex- 53626 ibmru ch
ARPANet- <hr.zurlvm1@ibm-sj.csnet>

Abstract

It is generally accepted that the advantages of using a formal protocol description are commanding. Once such a description is available, particularly in executable form, 1) the protocol specification can be examined for certain kinds of errors; 2) its performance can be estimated; and 3) the implementation can be tested for architectural conformance -- all in partially automated fashion. After a look at formal protocol specification, topics 1) and 2) will be discussed. Topic 3 will be briefly mentioned here; it is covered in another talk in this seminar.

18th IBM / University of Newcastle upon Tyne International Seminar on Network Protocols, 3 - 6 September, 1985

printed 31/08/85

Note: Part of this material is taken from H. Rudin, "An Informal Overview of Formal Protocol Specification," IEEE Communications Magazine, March 1985, Vol. 23, No.3, pp. 46 - 52.

I. INTRODUCTION

With the passing of time, information-handling systems now handle increasing function and have become increasingly coupled with one another. This coupling among the various, often distributed parts is achieved via "protocols". The complexity of even a stand-alone information system is now so great that it must be partitioned into modules of comprehensible size, creating the necessity for synchronization and communication between these modules: more protocols.

These information-handling systems or networks are often put together from subsystems from many different sources. The component systems -- be they from a single supplier or from different suppliers -- must function well together. It is hardly necessary to say that the suppliers must have identical interpretations of the protocols they are to implement. Prose definitions have been found to be far too imprecise; thus, there is a growing reliance on formal specifications.

Protocols should be free from errors. The use of formal specifications opens the door to **validation** -- showing that a protocol is free from certain kinds of syntactic errors -- and to **verification** -- showing that a protocol provides some specific function.

Once implemented, a protocol should perform reasonably well. It is important to have an indication of **performance** early in the protocol design process.

Finally, the implemented protocol must be a faithful instantiation of the specified protocol. This could conceivably be guaranteed by a proven **compilation** procedure, driven direct from the formal protocol specification. Failing such an (unlikely) procedure, the protocol implementation should be **tested** or **certified** relative to the formal specification.

These requirements lead to the topics covered in this paper: 1) formal protocol specification; 2) trying to ensure that these specifications are error-free; 3) showing that the architected protocol will perform reasonably well when implemented; 4) producing an implementation of the protocol which corresponds to the formal specification from which it is derived; and 5) testing the implementation for conformance.

These same topics also form the basis of a system approach to "protocol engineering", a term coined by Tom Piatkowski in 1981. (Piatkowski81) This author's view of such a system is shown in Figure 1. We shall now discuss, in sequence, the aspects of protocol specification and analysis just enumerated.

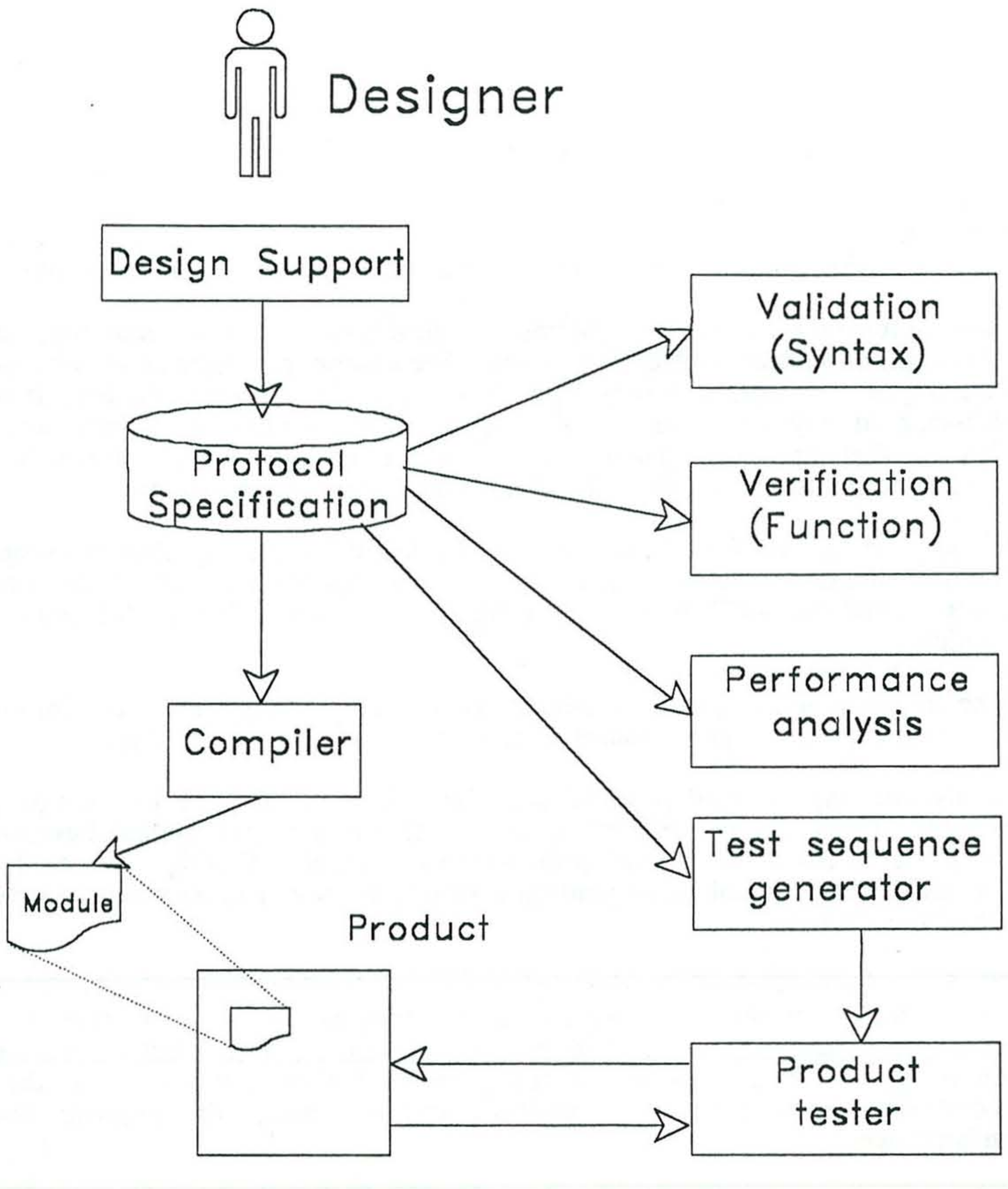

Figure 1. One View of Protocol Engineering

II. FORMAL DESCRIPTION TECHNIQUES

There is a number of formal description techniques in use for formally specifying protocols and still more have been proposed. The author's experience has been mainly with finite-state-machine (FSM) based descriptions which results in an FSM bias here of which the reader has now been warned!

II.1 A simple finite-state-machine model

The CCITT (Consultative Committee for International Telephone and Telegraph) has long been active in the formal specification of protocols. The X.21 and X.25

interface recommendations used a combination of finite-state-machine diagrams and natural language in the versions approved in 1976 (CCITT76) and revised in 1981 (CCITT81).

What does such a formal description look like? Figure 2 is an FSM (Finite-State Machine) model of a simple call-setup protocol between two processes, the terminal on the left and the network on the right. Each of these processes may be in one of two states: READY or CONNECTED. In the former state the process is waiting for a connection to be initiated; when both processes are in the latter state the connection exists and data can be sent back and forth. The terminal can initiate the call-connection procedure by sending a CALL_REQUEST message when it is in the READY state. The (-) sign indicates the generation of this message. This is an asynchronous transition, i.e., at any time when the terminal is in the READY state, it may send this message to the network. When the network is in the READY state and receives this CALL_REQUEST message, indicated by the (+) sign, the network makes the transition from the READY to the CONNECTED state.

It should also be possible for the terminal to receive an incoming call from the network. Such a possibility is also indicated in Figure 2 wherein the network can send an INCOMING_CALL message which can in turn be received by the terminal. In Figure 2 we have, then, the beginning of a formal specification for a call-setup protocol, similar to that used for the specification of X.21 and X.25 but simpler.

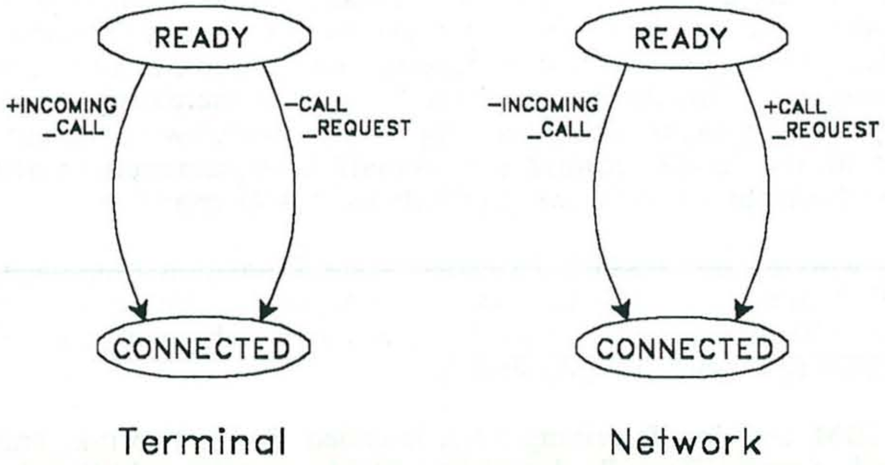

Figure 2. Finite-state-machine representation of a connection protocol

Once such a formal description is available, much can be achieved with it. For example, the specification can be "validated", i.e., checked for certain kinds of syntactical errors. There are in fact several syntactical errors which exist in the protocol as defined in Figure 2. We shall revert to these errors later.

II.2 Extended finite-state-machine models

The most popular techniques for formal protocol specification are based on finite-state machines, as in the example, but with extensions. One reason for the incorporation of extensions is that most protocols are so complex that it is impractical and certainly unwieldy to record all the necessary information to describe the dynamics of the protocol as different explicit states. Rather, the conceptually less important information -- such as addresses and sequence numbers -- are stored as auxiliary variables. Decisions can be based and actions can be performed on these auxiliary variables. This results in an "extended" FSM description.

CCITT has had an effort for some time to create SDL (Specification and Description Language) which is an extended FSM language (Dickson83). Experience had shown that while the formal part of the X.21 and X.25 definitions was very useful, these formal definitions could only be interpreted with heavy reference to the text which accompanied them. SDL improves this situation inasmuch as more of the protocol description can be formalized. A refinement of SDL in a Pascal-oriented language is being proposed within CCITT as Recommendation X.250.

ISO's work on formal description techniques (FDT's) in connection with the Open System Interconnection (OSI) Architecture (Zimmermann80) is proceeding in three parts. Subgroup A is concerned with architectural concepts, to be used in turn by the other two subgroups, concerned with FDT languages, per se. The concept of "modules" which contain the description of the function of the protocol, and the "channels" which provide communication between the modules have been specified.

An important concept in any structured design is the notion of consecutive refinement. With this concept a high-order module can be defined which simply specifies that some function be carried out. As the specification develops, such a module may be refined repeatedly, decomposing the function at one level into the required subcomponent functions at a lower level, thus increasing detail at every refinement step. This process facilitates a high-order overview of the entire system at one end and all the details required for compatible implementation at the other. This concept is also included in the work of Subgroup A (Vissers83).

Subgroup B is working on "ESTL", Extended State Transition Language, which is an extended FSM approach. ESTL, also called "Estelle", is based on the language Pascal. Various features have been added, particularly those which facilitate the definition of finite-state machines (Vissers83).

The language IBM uses for describing SNA is called FAPL (Format and Protocol Language). It is derived from PL/I and like ESTL contains additional constructs for handling finite-state machines and processes (Pozefsky82).

An important question in any formal description technique is "To what degree should the specification affect the implementation?" Clearly implementations of the same architecture must be compatible but how many hints should the architects place in the specification to influence the implementers direct? SNA provides many such hints via FAPL and even calls its FAPL definition a "Meta-Implementation". The definition itself comes close to a real implementation. This has the advantage that the definition can be executed and many questions about how the architecture performs can be answered by exercising the definition itself. Parts of the implementation are deliberately left to the implementers and appear in the formal specification only as "undefined protocol machines".

II.3 Other formal approaches

Subgroup C of ISO's effort on formal description techniques is working on an approach which has its roots in a formally complete theory of the sequence of messages exchanged by communicating processes: R. Milner's Calculus of Communicating Systems (Milner80). The language being developed by Subgroup C is LOTOS (Language for Temporal Ordering Specification) (Vissers83) and (Brinksma84). Since an effort is made only to describe the message sequences, there is a minimum impact on the specification of an implementation. This leaves implementers the maximum amount of freedom but still provides sufficient guidance to ensure compatibility.

The greatest promise of LOTOS lies in the fact that it allows as many levels of refinement as are needed, through the use of two language operators "parallel composition" and "restriction". This is not the case for ESTL. On the other hand, LOTOS appears to be more abstract, compared to FSM-based languages. The names of the states in an FSM description provide a convenient "crutch" for human interpretation.

There are a number of other approaches to formal specification. Of the ones not mentioned here, the Petri net is probably the most popular (Diaz82) and (Diaz83). Petri nets have been used for the specification of protocols but their main utility seems to be for the analysis of protocols rather than as a means of disseminating protocol specification. Petri nets consist of "places" and "transitions" to represent conditions and events. A very readable general introduction is given in (Peterson77).

It will be some time before the pros and cons of all these different formal description techniques are known. The only way of evaluating these is to use them on various protocols and this is just what is happening at the present time. There is a great deal of cooperation, particularly among the CCITT and ISO efforts.

It is quite clear that a formal protocol specification is mandatory since only in this way can one be sure that all the implementers have the same mental picture of the protocol they are implementing. In the following we assume that such a formal description is available, and further that it is machine readable -- as are all the techniques listed above. So given such a formal specification for purposes of communicating the protocol specification to its implementers in an unambiguous fashion, what else can be achieved?

III. FUNCTIONS BASED ON A MACHINE-READABLE SPECIFICATION

In the following it is assumed that the formal protocol specification is also machine-readable. Given that assumption some of the functions shown in Figure 1 can be obtained in semi-automatic fashion. Ideally, a designer should express his protocol formally, be able to test this specification for correctness (validation and verification), obtain some early indication of how it would perform, compile major parts of the implementation direct from the formal specification, and finally test the resultant implementation to assure that it conforms to the specification. Let us now look at these various functions and the state-of-the-art.

III.1 Validation

A test for syntax, usually called validation, can be performed on protocols. The kinds of errors which can be automatically detected are deadlocks, incomplete design, and unexecutable code when the specification is of the finite-state machine type (Zafiropulo80). What are some of these errors?

Reverting to Figure 2, suppose that both terminal and network try to initiate a call at the same time. "At the same time" means for example, that the network sends an INCOMING CALL message after the terminal has sent a CALL REQUEST message but before the network has received this CALL REQUEST message. (Part of the abstraction of this model is that it can take some time for messages to travel between the communicating processes; the processes are in effect connected by queues. The actual time delay introduced by these queues, per se, and the permutation of message interleavings made possible by a random distribution of delays are what make protocol analysis difficult.)

Thus there has been a collision of messages. The network will be offered the CALL_REQUEST message in the CONNECTED state and the terminal will be offered the INCOMING_CALL message in its CONNECTED state. But the protocol designer has not specified what should be done in these circumstances. This is an example of incomplete design and is the most common error we have seen in protocol designs. The 1976 versions of both X.21 and X.25 had errors of this type, long since corrected.

But suppose there were no collision and consider the case where the terminal has successfully initiated a connection... what happens then? Notice that nothing more can happen. Both processes are in the CONNECTED state and neither is free to generate a message. This is an obvious deadlock situation. It can be automatically identified and flagged via validation. It is conceivable but unlikely that the designer wanted interaction to end at this point and that there is no error. Usually deadlocks occur in undesired and much more subtle ways.

Figure 3 shows repairs for both of these errors. The colliding messages have been discarded and provision has been made for terminating the connection so that other connections can be initiated at later times. Is the protocol now error free in the syntactical sense? No. The reason is that the network can receive the TERMINATE message twice in succession. This is probably not obvious but in any case can be repaired as indicated in Figure 4 which represents a syntactically correct protocol. Other errors -- such as dead code -- can also be detected by validation. The two error types just given are sufficient to demonstrate the results of the technique.

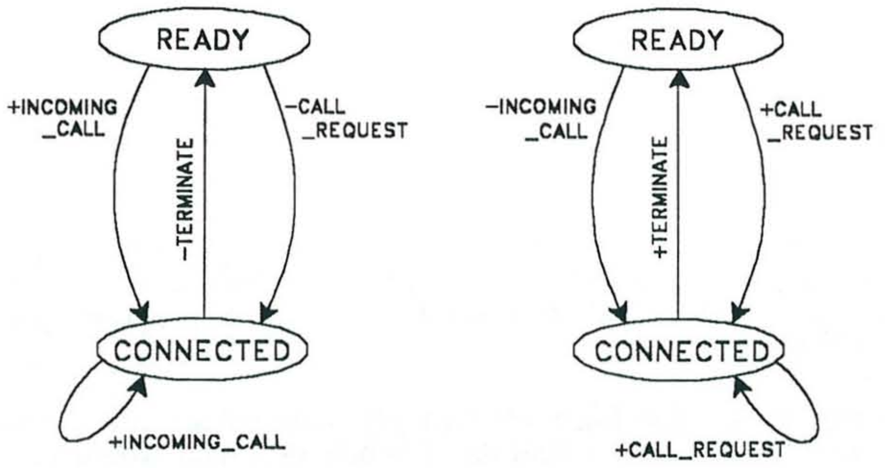

Figure 3. Improved specification of the protocol shown in Figure 2.

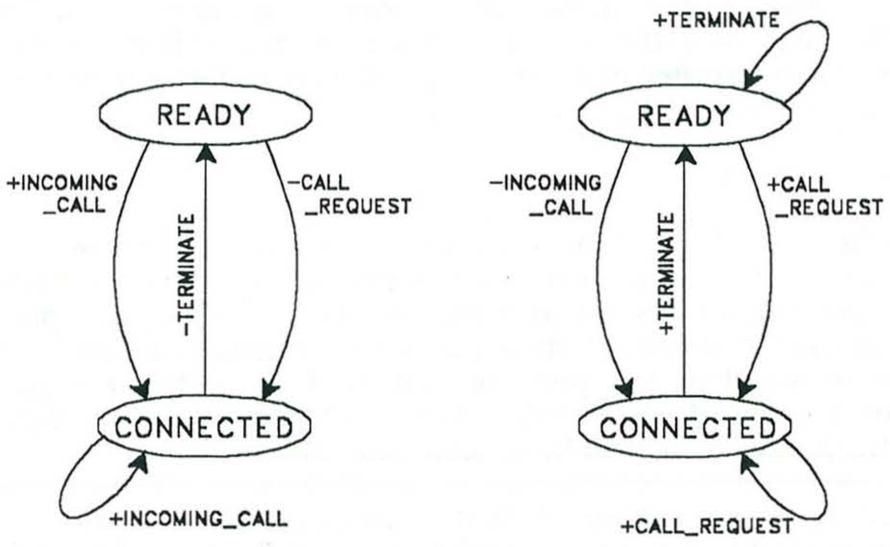

Figure 4. Final specification of the protocol shown in Figure 2.

One of the advantages of validation is that it is applicable to virtually all protocols. Further, because we are dealing with parallel processes which may interact with each other in many ways, it is difficult for human designers to anticipate all the different interaction sequences and produce an error-free design. The author and his colleagues have used or seen validation tools used on twenty-odd protocols and only one of these was error free (Rudin82c)! All the rest had errors of varying degrees of severity. An effective and proven means of performing validation by using state exploration is described in (West78b).

III.2 Verification

Validation helps to ensure lack of syntactical errors, errors which would prevent the protocol from fulfilling its function. But, even in the absence of syntactic errors, will the protocol achieve its functional objective? This is a question of protocol verification.

An example from the author's experience occurred in the analysis of a token-ring local-area network. Here a special station on the ring ensured that transmission noise would not cause mutilation, corruption, or loss of a token in such a way that the system would be unable to recover. An automated verification of the system, using state exploration, was performed (Rudin82b) to show that the architected system did indeed have this desirable property.

The verification process just described was tuned to the token-ring protocol at hand. In contrast, there are a number of semi-automatic, generally applicable verification packages available. These and experiences with these program packages are summarized in (Sunshine83).

Methods of calculus, in the form of partially automated, interactive theorem provers, show promise for the verification of protocols which would be difficult to handle using state exploration techniques. An example of this is work going on at the University of Karlsruhe (Krumm84).

At this point in our design system we have a formal, machine-readable definition of the protocol. By using validation we have removed its syntactic errors and by using verification have demonstrated that the protocol does in fact provide at least the specific function for which it has been examined.

III.3 Performance Prediction

How well will the protocol perform in terms of throughput or response time? Until recently there were two main means of ascertaining performance. One was to make a model of the protocol and its environment and then to use various mathematical tools -- such as traffic theory -- to make a performance analysis. The other possibility was to simulate the protocol and then to gather statistics on the performance of the model simulated. The availability of a machine-readable description of the protocol ought to be of some help, and it is.

The first work to make use of a formal specification is that of Bauerfeld (Bauerfeld83) wherein he generates a simulation model from the formal description automatically. A next step was explored by the author (Rudin84b) wherein performance estimates are obtained direct from analysis of the protocol's overall finite-state diagram. This latter analysis is based on an FSM description. Similar results have been obtained based on Petri-net descriptions (Molloy82) and (Razouk84). Very recently an effort has been made to combine the techniques of queueing network theory in an analysis driven direct by the formal specification (Kritzinger84b). This last work is directed specifically at the OSI Reference Model and takes into account the interplay among various protocol layers as well as the coexistence of several protocol instances at any one layer.

An example of the kind of results which may be had, for the case of a data link control protocol, is shown in Figure 5 after (Rudin84b). This figure shows the average delay between successful receptions of messages (including retransmissions resulting from message loss or message corruption in the communications medium).

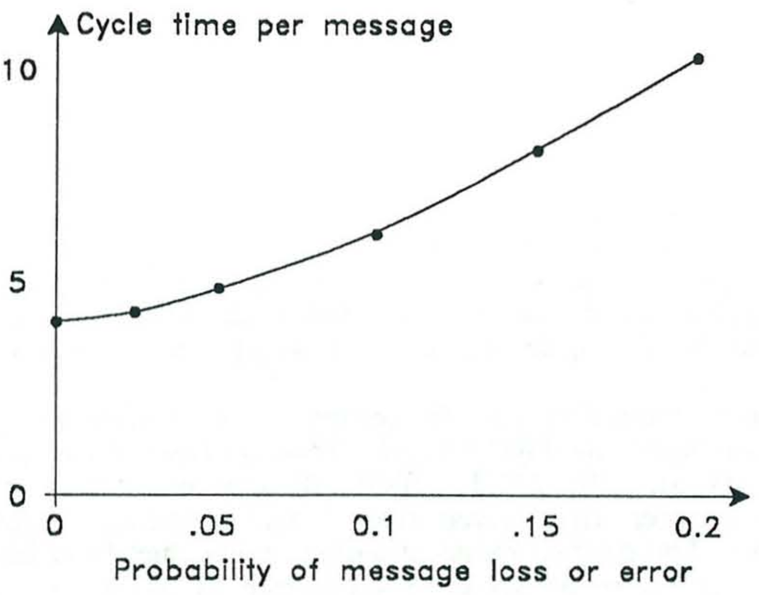

Figure 5. Delay in a simple data link control protocol

III.4 Automated Implementation

The final objective of any design is incorporation of the protocol in a successful implementation. Traditionally, this has been done by hand-coding. Again, the availability of a machine-readable specification opens the door to direct compilation of parts of the code required for an implementation. In the last few years, direct compilation of a protocol has been carried out in several instances. There will always be substantial portions of an implementation which must be hand-coded but there is the hope of being able to produce the bulk -- say sixty percent -- of the necessary code by automated means.

In the case of SNA, Nash has demonstrated automated generation of an implementation in work he did in 1979 (Nash83). Compilation of a protocol -- in this case part of SNA's Data Flow Control -- was carried out in two major steps. First the FAPL compiler was used to expand the FAPL specification into an intermediate language (PL/S). The required manual code was also written in PL/S. This PL/S code was then compiled and assembled into the required machine code. The two-step process means that only one FAPL compiler is required and takes advantage of existing PL/S compilers which produce code for various target machines. The particular application was for part of the IBM 8100 Information System; the results in terms of required path length (number of instructions to be executed) and memory space required were satisfactory and the product was shipped (Nash83 and Smith83). The use of an automated technique substantially reduced implementation time. Automatic compilation has been used several times in the case of SNA (Nash83) and (Smith83).

The NBS (National Bureau of Standards) has developed its own extended FSM language, actually a predecessor and subset of ESTL. The NBS language was used to describe a subset of the OSI File Transfer Protocol as part of a test tool to be described in the next section (Linn84) and (Mills84). The point here is that for the test tool developed at NBS, a reference model, i.e., an executable model of the file-transport protocol was required. The formal specification was compiled into the language "C". About 40 percent of the code required in "C" could be automatically compiled; the remainder had to be hand-coded.

III.5 Testing for Conformance

It is unlikely that the complete protocol specification can ever be compiled direct into a software or firmware implementation; there are simply too many idiosyncrasies for each particular microprocessor engine. The result is that there will always be a need for techniques to test an implementation to demonstrate that it does in fact correspond to the formal description from which it was derived.

An early use of a formal protocol definition for testing occurred in conjunction with SNA in the form of a tool called METROPOL (MEthod of Testing Real-Operation Protocols Off-Line) (Cork83). In METROPOL dialogs between a prototype implementation and its partner are derived from trace recordings made during conventional test sessions. The derived message sequence can then be used as input to the executable FAPL definition of SNA. SNA is rich in its internal checking function and any error detected via these checks -- as well as any other observed anomalies -- can be flagged to the operator. METROPOL found a number of errors in prototype SNA implementations.

The National Physical Laboratory in Teddington has long been active in developing protocol assessment techniques (Rayner83). As a result of the work done at NPL, a pilot assessment center has been set up at the National Computing Center in Manchester. In this approach the implementation under test is exercised by a test responder and an "encoder-decoder", capable of applying normal (protocol-specification conform) sequences as well as abnormal sequences to the implementation under test. The abnormal sequences are used to test the response of the implementation to error conditions.

In a similar approach the National Bureau of Standards (NBS) in the US uses the formal specification more direct as a reference implementation with which the implementation under test communicates; an exception generator is used to generate abnormal sequences (Linn84) and (Mills84). NBS has also used the formal protocol definition to generate automatically some of the sequences or "scripts" used to drive both the reference and test implementations in a manner indicated in Figure 1.

Automated testing systems -- and the importance of formal protocol definitions as well -- were recently given credence in a demonstration at the National Computer Conference held in Las Vegas in July 1984. The demonstration was coordinated by NBS and based on the ISO Class-4 Transport Protocol. About a dozen manufacturers successfully demonstrated communication among their equipment using an agreed file transfer protocol as the common application. The formal definition and the automated testing of the various implementations played a key role in the success of the demonstration (Linn84).

III.6 Change Management

Change management is not a concept which appears explicitly in the protocol engineering system of Figure 2. Consideration of the problem of managing change, as additions and improvements to the protocol system architecture are made and new products are introduced, results in an additional factor in favor of formal, machine-readable specification. To the extent that the procedures described above have been automated, the need for manual work to redo the various functions is minimized.

IV. SOME NOTES ON TECHNIQUES

In the work on validation, verification, and performance prediction with which the author has been associated, state exploration techniques have played a key role. The validation technique which we have used most, generates a tree of global states (West78b). A global state consists of the state of each of the component processes plus the ordered content of the communication media or communications channels (modeled as queues of messages or events), which may pairwise interconnect the processes.

This construction of a global-state tree, routed in the global state where each of the component processes is in its initial state and all of the communication channels are empty, is shown in Figure 6.

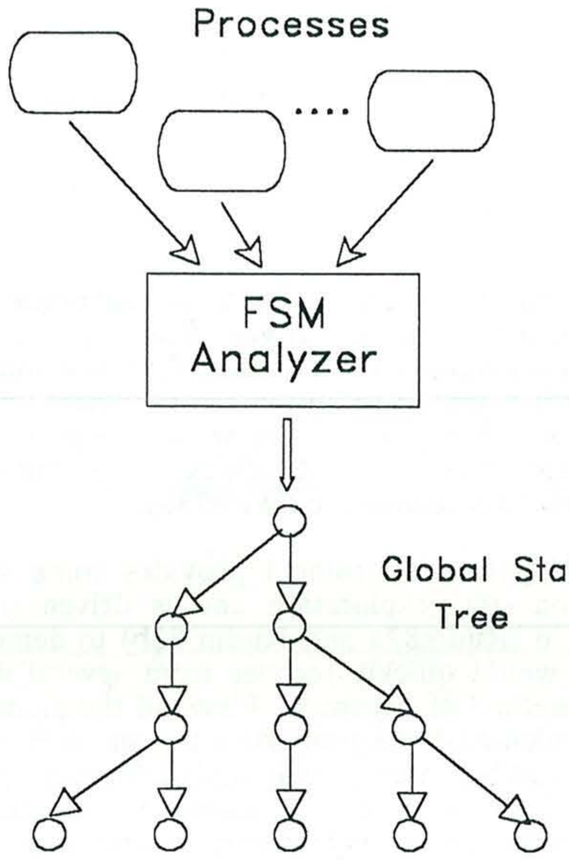

Figure 6. From the FSM representation to a global state tree

Starting from the initial states, all admissible successor states are identified. New global states are added to the tree; growth stops when no unknown global states can be generated. For each of the leaves in the tree, a pointer is kept to the (already generated) global states which could follow. This global-state tree is the basis for most of the analyses which we have made.

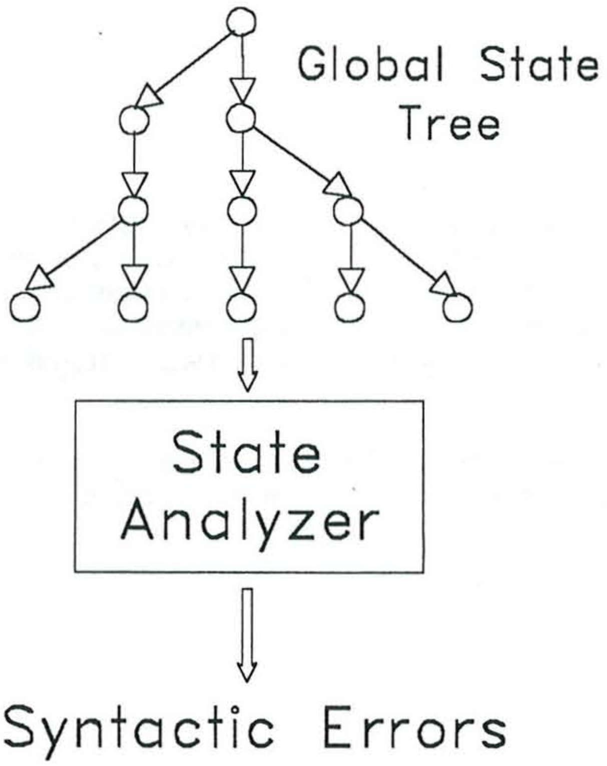

Figure 7. Validation: detecting syntactic errors

The process of validation can be achieved incrementally during the generation of the global-state tree or can be thought of as being driven from the completed global-state tree. The latter is shown schematically in Figure 7. Each one of the global states is checked to see whether it is free of reception errors (where one of the processes is offered a message from a channel in a state in which no provision has been made for its reception), does not result in a deadlock, or is unreachable (indicating "dead code"). Again the reader is referred to (West78b).

Figure 8 shows one means of verifying that a protocol provides some specified function. Again the process relies on state exploration and is driven from the global-state tree. The technique used in (Rudin82a and Rudin 82b) to demonstrate that a token-ring local-area network would quickly recover from several different kinds of transmission errors used the method of Figure 8. First, all the global states in the global-state tree were found which had the property that an error had just been introduced. (Errors were introduced by an artificially added "demon" process.) Paths were then generated starting from each one of these global states and continuing until a global state was reached which indicated that error recovery had successfully taken place. Count is kept of the number of iterations required to generate the paths; if this count exceeds a preset limit, the conclusion is that recovery has not yet occurred or may never occur.

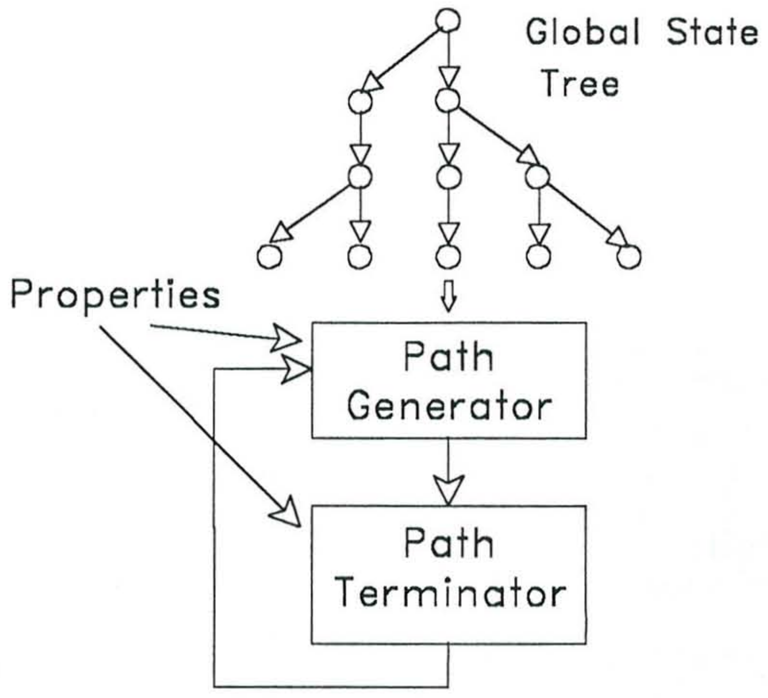

Figure 8. Verification: proving function

Finally, Figure 9 indicates one means of predicting protocol performance, again driven from the global-state tree and using state-exploration techniques. In order to be able to determine performance the finite-state-machine description must be augmented. Delays (usually average delays representing the time it takes to execute a transition) and branching probabilities (when one of several transitions may follow a global state) are added to the formal specification.

To calculate average delay, paths can be explored which emanate from a global state with particular properties (the reception of a positive acknowledgement in the data-link-control protocol for which performance is shown in Figure 5) and terminate in some global state (the identical state in the case of the protocol of Figure 5). The average delay may then be calculated as

$$\text{Average delay} = \sum_{i=1}^{N} (P_i)(D_i),$$

where D_i is the delay along a path and P_i is the probability of being on that path.

The calculation is terminated when the sum of the probabilities of unterminated paths drops below a specified threshold.

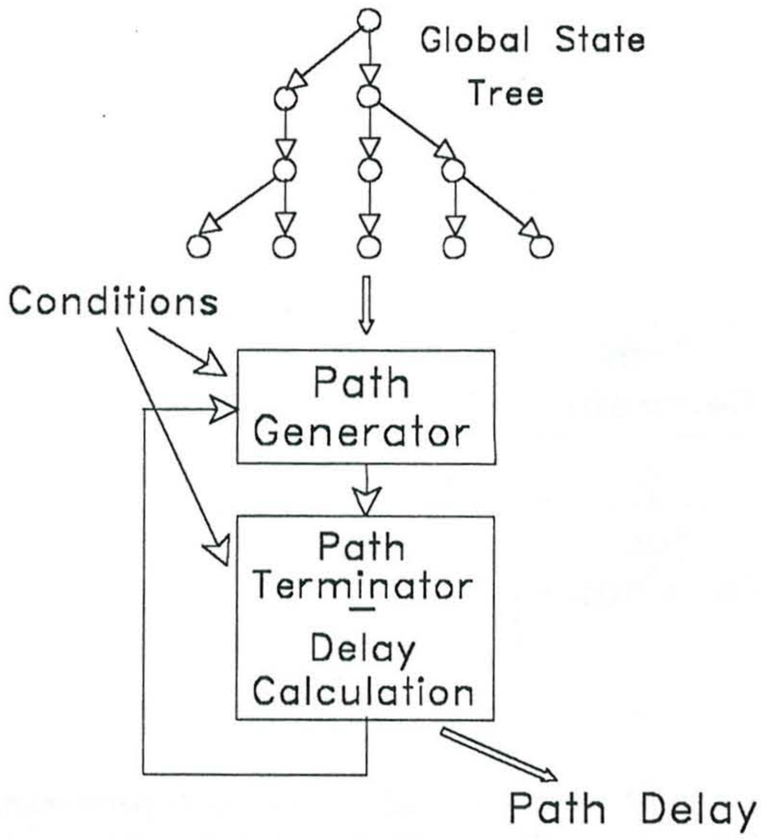

Figure 9. Predicting protocol performance

None of the techniques mentioned here -- at least in rudimentary form -- represents an enormous programming task. All could be used as software design exercises. The inclusion of such tasks in a curriculum would not only provide meaningful exercises but could teach much about computer communication protocols as well.

V. FUTURE DEVELOPMENTS

In our tour of the protocol engineering system of Figure 1, the reader may have been misled into thinking that all the problems have been solved. This is not the case; there is need for continuing research in all of the areas mentioned. It should also be stated that several of the examples given are taken from work which was close at hand to the author; there are alternative approaches, some of them with advantages over the methods presented here, depending on the circumstances. Pointers to the literature covering some of these alternatives are given in Section VI.

Let us now review the areas discussed to see some of the further research needed.

V.1) Validation

We are unable to validate "complex" protocols. The most prevalent technique is exhaustive exploration of a multidimensional state space. A "complex" protocol leads to an impractically large number of states when validation is attempted. Many protocols are complex and the need for an appropriate approach is urgent.

A related question deals with partitioning: "Under what conditions is it satisfactory to validate the component parts of a protocol, having a guarantee that a validation of the complete protocol is unnecessary?" Further exploration of the calculus underlying LOTOS will likely shed some light here.

Often it is necessary to include the specification of one or more time-outs in a protocol. A difficult problem is validation of a protocol to determine whether or not these values are consistent with each other and with the overall protocol specification. The problem is further complicated when delays are specified for the time it takes for messages to flow between processes. Some progress has just been made here (Bolognesi84) but more is needed.

V.2) Verification

When exhaustive exploration is used as a basis for verification, the problem of handling complex protocols again arises. Alternatively there are algebraic techniques but these tend to require heavy human intervention. There is a need both to refine these techniques and to automate them to a greater extent.

V.3) Performance prediction

Work here, in the sense that performance analysis programs are driven direct from the protocol's formal specification, has just begun. One interface which promises to be fruitful is that between traditional queueing theory with its ability to take waiting customers into account and the contention-free analysis discussed above.

V.4) Automatic implementation

While there have been a few experimental efforts here, experience is relatively limited, despite the obvious large return. Using the latest techniques in compiler design, a number of test cases could be made comparing the memory and execution-time efficiencies of automatically and manually produced implementations.

V.5) Automated testing

How long should a test be run? How much test coverage is obtained at the cost of what execution time? Is it possible to design particularly effective test sequences? What is the point of diminishing returns in testing? There are many open issues here. Experience with the first automated test systems is just becoming available. This experience will help to determine future directions.

VI. POINTERS FOR FURTHER READING

The December 1983 issue of the Proceedings of the IEEE (Folts83) is devoted to the OSI Model and related topics. A special issue of the IEEE Transactions on Communication in April 1980, since published with additions in book form, is a collection of papers on computer-networking architecture and protocols (Green80). Carl Sunshine also edited a collection of papers, including some of the early classics in protocol definition and analysis (Sunshine81).

A number of the topics emphasized in this paper was covered in a session at the ICC83 in Boston (Bauerfeld83, Diaz83, Nash83, Piatkowski83, Rayner83, Rudin83c, Sunshine83, and Tenney83). Finally, IFIP Working Group 6.1 has sponsored a series of annual workshops devoted to the specification, verification, and testing of protocols which contain state-of-the-art contributions (Rayner81, Sunshine82, Rudin83a, Yemini84, Diaz85). The next of these workshops is scheduled for Montreal in June, 1986.

VII. CONCLUSION

Formal protocol specification is a concept which has firmly taken root and is here to stay. There is likely no "best" formal specification technique. Further work is required to understand where each of the existing techniques has its greatest advantage. There is scope for much more work on the tools which may be driven from a machine-readable formal specification. The success of the tools which already exist promises handsome rewards for this future work.

REFERENCES

Bauerfeld83- W. L. Bauerfeld, "Protocol performance prediction," Proc. International Conference on Communications, IEEE, Boston, Mass., June 20-23, 1983, pp. 1311 - 1315.

Bolognesi84- T. Bolognesi and H. Rudin, "On the analysis of time-constrained protocols by network flow algorithms," Proc. Workshop on Protocol Specification, Testing, and Verification, IV, Skytop, Pennsylvania, June 1984, (North-Holland, Amsterdam, 1985), pp. 491 - 513.

Brinksma84- E. Brinksma, "A specification of the OSI transport service in LOTOS," Proceedings of the Workshop on Protocol Specification, Testing, and Verification, IV, Sky Top, Pennsylvania, June 1984, (North-Holland, Amsterdam, 1985), pp. 227 -251.

CCITT76- C.C.I.T.T., Orange Books, Sixth Plenary Assembly, Vol. VIII.2, Geneva, 1976.

CCITT81- C.C.I.T.T., Yellow Books, Seventh Plenary Assembly, Vol. VIII.2, Geneva, 1981.

Cork83- R. M. S. Cork, "The testing of protocols in SNA products", Proceedings of the Workshop on Protocol Specification, Testing, and Verification, III, Ruschlikon, Switzerland, May 1983, (North-Holland, Amsterdam, 1983) pp. 455 - 463.

Diaz82- M. Diaz, "Modeling and analysis of communication and cooperation protocols using Petri-net based models," Computer Networks, Vol. 6, 1982, pp. 419 - 441.

Diaz83- M. Diaz, "Status of using Petri nets for protocols," Proceedings of the International Conference on Communications, Boston, Mass., 20 - 23 June, 1983, pp. 1301 - 1305.

Diaz85- M. Diaz, editor, Proceedings of the Fifth International Workshop on Protocol Specification, Testing, and Verification, Moissac-Toulouse, June, 1985, (North Holland, Amsterdam, 1985).

Dickson83- G. J. Dickson and P. J. Chazal, "Status of CCITT description techniques and application to protocol specification," Special issue of the Proceedings of the IEEE on OSI, Vol. 71, No. 12, December, 1983, pp. 1346 - 1355.

Folts83- H. C. Folts and R. desJardins, Eds., Special issue of the Proceedings of the IEEE on OSI, Vol. 71, No. 12, December, 1983, pp. 1331 - 1452.

Green80- P. E. Green, Editor, IEEE Trans. Commun., April, 1980, vol. COM-28, no. 4, pp. 409 - 677. Later republished as Computer Network Architectures and Protocols (Plenum, New York, 1982).

Kritzinger84b- P. Kritzinger, "A performance model of the OSI communication architecture", IBM Zurich Research Laboratory Research Report RZ 1346, Ruschlikon, Switzerland, Dec. 11, 1984.

Linn84- J. Linn, "An evaluation of the ICST test architecture after testing Class-4 Transport", Proceedings of the Workshop on Protocol Specification, Testing, and Verification, IV, Skytop, Pennsylvania, June 1984, (North-Holland, Amsterdam, 1985), pp. 611 - 621.

Mills84- K. L. Mills, "Testing OSI protocols: NBS advances the state of the art", Data Communications, Vol. 13, no. 3, June, 1984, pp. 277 - 285.

Milner80- R. Milner, "A Calculus of Communicating Systems", Lecture Notes in Computer Science, Springer Verlag, Berlin, 1980.

Molloy82- M. K. Molloy, "Performance analysis using stochastic Petri nets," IEEE Trans. Computers, Vol. C-31, no. 9, pp. 913-917, Sept. 1982.

Nash83- S. Nash, "Automated implementation of SNA communication protocols," Proceedings of the International Conference on Communications, Boston, Mass., 20 - 23 June, 1983, pp. 1316 - 1322.

Piatkowski75- T. Piatkowski, "Finite-state architecture," IBM Technical Report TR-29.0133, Systems Development Division (now Systems Communications Division), Research Triangle Park, North Carolina, August, 1975.

Piatkowski81- T. Piatkowski, "An engineering discipline for distributed protocol systems," Protocol Testing - Towards Proof? (an INWG/NPL Workshop) National Physical Laboratory, Teddington, U. K., 27-29 May 1981, pp. 177 - 215.

Piatkowski83- T. Piatkowski, "Protocol engineering", Proceedings of the International Conference on Communications, Boston, Mass., 20 - 23 June, 1983, pp. 1328 -1332.

Pozefsky82- Pozefsky, D. P. and F. D. Smith, "A meta- implementation for Systems Network Architecture", IEEE Trans. Commun., Vol. COM-30, no. 6, pp. 1348-1355, June, 1982.

Rayner81- D. Rayner and R. W. S. Hale, editors, Protocol Testing - Towards Proof? (an INWG/NPL Workshop), Vol. 1: Specification and Validation and Vol. 2: Testing and Certification, National Physical Laboratory, Teddington, U. K., 27-29 May 1981.

Rayner83- D. Rayner, "Progress in testing protocol implementations", Proc. International Conference on Communications, Boston, Mass., 20 - 23 June, 1983, pp. 1323 - 1327.

Rudin82b- H. Rudin, "Validation of a token-ring protocol," in Proc. Intl. Symposium on Local Computer Networks, Florence, April, 1982, (North-Holland, Amsterdam, 1982) pp. 373 - 387.

Rudin82c- H. Rudin, "Automated protocol validation: some practical examples", Proc. Sixth International Conference on Computer Communication, London, September 1982, pp. 919-924.

Rudin83a- H. Rudin and C. H. West, editors, Proceedings of the Third International Workshop on Protocol Specification, Testing, and Verification, Ruschlikon, 31 May - 2 June, 1983, (North Holland, Amsterdam, 1983).

Rudin83c- H. Rudin, "The ICC83 Communications Protocol Session: an overview," Proc. International Conference on Communications, Boston, Mass., 20 - 23 June, 1983, pp. 1291 - 1295.

Rudin84- H. Rudin, "An improved algorithm for estimating protocol performance", Proc. Workshop on Protocol Specification, Testing, and Verification, IV, Skytop, Pennsylvania, June 1984, (North-Holland, Amsterdam, 1985), pp. 515 - 525.

Smith83- F. D. Smith and C. H. West, "Technologies for network architecture and implementation," IBM J. Res. Develop., Vol 27, No. 1, January 1983, pp. 68 - 78.

Sunshine81- C.A. Sunshine, Ed., Communication Protocol Modeling, (Artech House, Dedham, Mass., 1981).

Sunshine82- C. A. Sunshine, editor, Proceedings of the Second International Workshop on Protocol Specification, Testing, and Verification, Idylwild, May 17 - 20, 1982, (North Holland, Amsterdam, 1982).

Sunshine83- C. A. Sunshine, "Experience with automated protocol verification," Proceedings of the International Conference on Communications, Boston, Mass., 20 - 23 June, 1983, pp. 1306 - 1310.

Tenney83- R. Tenney, "Status of the ISO ad hoc Subgroup B on Formal Description Techniques", Proceedings of the International Conference on Communications, Boston, Mass., 20 - 23 June, 1983, pp. 1296 - 1300.

Vissers83- C. A. Vissers, R. L. Tenney, and G. V. Bochmann, "Formal Description Techniques", Special issue on OSI, Proc. IEEE, Vol. 71, No. 12, December, 1983, pp. 1356 - 1364.

West78b- C.H. West, "General technique for communications protocol validation," IBM J. Res. Develop., Vol. 22, July 1978, pp. 393-404.

Yemini84- Y. Yemini, editor, Proceedings of the Fourth International Workshop on Protocol Specification, Testing, and Verification, Skytop, June, 1984, (North Holland, Amsterdam, 1985).

Zafiropulo80- P. Zafiropulo, C.H. West, H. Rudin, D.D. Cowan, and D. Brand, "Towards analyzing and synthesizing protocols," IEEE Trans. Commun., Vol. COM-28, April, 1980, pp. 651-660. Also reprinted in P. E. Green, ed., Computer Network Architectures and Protocols (Plenum, New York, 1982), pp. 645-669.

Zimmermann80- H. Zimmermann, "OSI Reference Model-- the ISO model of architecture for Open Systems Interconnection," IEEE Trans. Commun., Vol. COM-28, April, 1980, pp. 425-432. Also reprinted in P. E. Green, ed., Computer Network Architectures and Protocols, (Plenum, New York, 1982), pp. 33 - 54.

DISCUSSION

Lecture 1:

Professor Randell commented that there appears to be a disadvantage in the complete mechanisation of the process. Either the specification or the implementation might contain errors. If both the specification and the implementation were provided by the designer, an element of redundancy can be included for error checking. Clearly this is not possible with a machine generated implementation.

Professor Milner asked what time delay is there between the emission of a value and its subsequent reception using the asynchronous finite state machines?

Dr. Rudin replied that they have moved away from asynchronous finite state machines and now consider a queue.

Professor Pyle stated that surely there is more to protocols than can be modelled using FSM's? For example, how would the presentation layer be modelled? There is a need for an analysis of protocols to identify these areas which cannot be modelled using FSM's.

Dr. Rudin's response was that clearly one cannot detect those areas which are not susceptible to FSM modelling. They are not sure what sort of errors to look for.

Dr. Rayner stated that in attempting to describe non-FSM entities one comes across the data typing problem. As yet he is not aware of anyone who has done work on that sort of validation of data typing.

Dr. Vissers remarked that in real protocols there are probabalistic expressions, time and performance. Any modelling technique needs to permit these expressions.

Mr. Llewellyn pointed out that this gives rise to a state explosion and concurrency problems. The temptation is to use a new extended FSM but this loses all the beauty of FSM's.

Dr. Rudin claimed that despite this, the technique may still be used effectively. However, algebraic techniques may prove sufficient in those areas where FSM's cannot be used.

DISCUSSION

Lecture 2:

Professor Randell asked what percentage of errors are caught by this technique, to which Dr. Rudin replied that no data is available.

Dr. Rayner asked whether the protocol development system that had been described is a proposal for implementation or a working system.

Dr. Rudin stated that it is possible to buy a development system and to perform the work described.

Dr. Cerf remarked that an important aspect of any protocol for the user is its cost - how much does it cost to send a message. In addition to analysing the layers it would be a good idea to examine the cost. The metric is the amount of data sent per packet.

Dr. Rudin replied that they are currently looking at the cost in terms of instructions executed per layer.

Dr. Rudin was asked whether they had tried the scheme of determining how good their technique is at finding errors by inserting deliberate errors into a protocol.

Dr. Rudin replied that that is a technique which they had not considered. In fact there are always enough errors in the protocols they have used so far!

Dr. Larcombe pointed out that the technique assumes that one can distinguish the actual errors from those introduced.

Mr. Cowlishaw stated that they have tried the technique on data links and are satisfied that it works.

Professor Randell mentioned one could use a recoverable Petri net which is extended by error links and nodes.

Dr. Rudin stated that they have not yet specified a protocol using a Petri net.

Dr. Burkhardt claimed a Petri net specification for the transport service works completely satisfactorily.

Professor Tiernari asked Dr. Rudin what use has been made of his technique.

Dr. Rudin replied it is to be used on SNA, and that they are also taking existing 'error free' protocols and using them. In this way they hope to find errors and thereby convince product managers of the usefulness of this technique.

Dr. Vissers pointed out that the SNA specification was initially informal. It was implemented before it was validated. He asked whether Dr. Rudin had any evidence that protocol designers are willing to adopt these specification techniques?

Dr. Rudin replied that there will always be an intuitive stage that leads onto a formal specification. When disputes arise over the prose, existing products need to be modified. There is always a hostility to the discovery of errors.