PROSPEC: An Interactive Programming Environment for Designing and Verifying Communication Protocols

CHING-HUA CHOW, MEMBER, IEEE, AND SIMON S. LAM, FELLOW, IEEE

Abstract—**PROSPEC is a software environment for designing and verifying communication protocols. It integrates several tools that implement methods for protocol verification and construction (e.g., fair reachability analysis, multiphase construction and protocol projection). The system provides a unified graphical interface to facilitate the application of these methods and creates an interactive environment for specifying, verifying and designing communication protocols. PROSPEC was used successfully to design and verify versions of BSC, X.21, X.25, and Telnet document transfer protocols.**

Index Terms—**Analysis and verification, communication protocols, formal description techniques, formal specification, programming environment, protocol design, reachability analysis, software tools.**

I. Introduction

COMMUNICATION protocols play an important role in computer networks and distributed systems. The increasing complexity and variety of protocols have forced us to seek powerful methods and tools to facilitate their design. To avoid errors and ambiguities, numerous formal models and verification methods have been proposed and applied to the verification and construction of communication protocols.

Among these models, Communicating Finite State Machines (CFSM) has been shown to be very useful in the specification [7], analysis [5], [8], [18]-[20], [22], [29], [32], [34], [36], [37]-[39], and synthesis [6], [11], [17], [21] of communication protocols. The procedure for modeling and analyzing a communication protocol using this model typically proceeds as follows.

- First, the protocol is defined as a network of communicating finite state machines: each machine in the network has a finite number of states and state transitions (called nodes and edges respectively herein). Each state transition of a machine is accompanied by the sending of a message into a channel or the receiving of a message from a channel. Channels are assumed to be unidirectional (delivering messages from one machine to another), error-free, and FIFO. A reliable point-to-point communication link can be modeled as two unidirectional FIFO channels. An unreliable point-to-point communication link or a broadcast channel can be modeled as a communicating finite state machine which is connected to other machines by error-free FIFO channels. The network may have an arbitrary topology.
- Second, the network defined is analyzed to ensure that its communication satisfies certain correctness properties such as boundedness [38], freedom from deadlocks, and freedom from unspecified receptions [32], [31], [36], [38].

Examples of some realistic protocols that have been modeled and analyzed using this procedure include: the alternating-bit protocol [3], the Binary Synchronous protocol [12], and the call establishment/clearing procedures in X.21 [29], [36], and in X.25 [18], [30].

With the CFSM model, it is most convenient and efficient for a human designer to specify finite state machines (FSM's) graphically and also to verify (or "debug") his protocols by looking at graphical representations of reachability graphs. Ideally, when a protocol has been designed and found to have the desired logical behavior, executable code can be generated directly from the internal representation of the protocol's graphical specification. With this objective in mind, several graphical tools for protocol design and reachability analysis have been developed, such as the one developed at IBM Research [39] and at Bell Laboratories [1].

We were motivated to develop the programming environment PROSPEC for interactive protocol verification and construction by an additional objective. An inherent weakness of the CFSM model is the difficulty to manipulate FSM's with many states and very large (possibly infinite) reachability graphs. Even if a graph is finite, it may be *too large to display on a screen*. Thus a user-friendly graphical interface in itself is not adequate to make life easy for protocol designers. The approach of PROSPEC is to incorporate abstraction and modular construction techniques into the tools presented to protocol designers. In addition to reducing a protocol analysis/con-

Manuscript received February 15, 1987; revised October 1, 1987. This work was supported by the National Science Foundation under Grants ECS-8304734 and NCR-8613338.

C.-H. Chow was with the Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712. He is now with Bell Communications Research, 435 South Street, Morristown, NJ 07960.

S. S. Lam is with the Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712.

IEEE Log Number 8718692.

struction problem into several smaller problems, such abstraction and modular construction techniques considerably enhance the effectiveness of an interactive graphical interface. At the same time, the interactive graphical interface facilitates the application of these techniques.

PROSPEC has been developed on a SUN 2/120 workstation running UNIX®. In addition to a graphical user interface, the attractiveness of PROSPEC lies in the user's ability to do protocol design and verification interactively and with access to several helpful tools that implement the methods of projection, multiphase protocol construction, and some others. In this respect, two features of PROSPEC are significant. First, a standard internal representation of data facilitates the passing of FSM's and reachability graphs from one tool to another. Second, each tool of PROSPEC is associated with a window and to further increase the interaction speed, we employ the menu utility of SUN and group all commands provided by PROSPEC into menus. The menu-selection facility relieves a user from having to remember all the commands and reduces the number of key strokes he or she has to enter for interaction with PROSPEC [13], [26], [14].

Currently, PROSPEC supports only the CFSM model and can be used for the design and analysis of protocols that can be specified as CFSM's. The CFSM's generated by PROSPEC should be considered as a communication skeleton. To generate executable code, these CFSM's need to be augmented with data specification and internal operations. There are several other protocol design systems developed in recent years which use different models. Blumer and Sidhu designed a software system which uses a Pascal-like language to specify protocols. Their system provides software tools to perform reachability analysis, and is capable of semiautomatic generation of executable code from a specification [4]. SPANNER is a software system which allows a designer to use the selection/resolution model to formally specify a protocol, and then to analyze the protocol using either reachability analysis or simulation [2]. IC* is a development environment that uses the IC* language (similar to selection/resolution model) to specify protocols. It analyzes protocols by simulating the specifications. A protocol prototype can be automatically generated from its IC* language specification [16].

In Section II, we describe the protocol verification techniques and the design methodology which are implemented in PROSPEC. In Section III, a brief overview of the PROSPEC system is presented. In Section IV, the specification features of PROSPEC are discussed. In Section V, we demonstrate the protocol analysis capabilities of PROSPEC. Four protocols were analyzed: a version of the IBM BSC call setup protocol, a version of the Telnet document transfer protocol, and a version of the IBM BSC data transfer protocol were analyzed by the interactive reachability analysis tool; a call management protocol was analyzed by the fair reachability analysis program. In Section VI, we illustrate how to use the tool that implements the protocol projection method. In Section VII a modified version of the BSC protocol is used as an example to demonstrate the tool that implements the multiphase protocol construction methodology. In Section VIII, some desirable extensions of PROSPEC are outlined. Section IX is for concluding remarks.

®UNIX is a registered trademark of AT&T Bell Laboratories.

II. Protocol Verification and Construction Techniques

In this section we give an overview of several protocol verification techniques and a construction methodology which are implemented in PROSPEC.

Reachability analysis is the basic technique for verifying networks of CFSM's. Starting from a given initial state, all possible transitions are explored and the reachable states are generated. The transitions and the reachable states constitute a labeled directed graph, called the *reachability graph*, which contains all information about the logical behavior of a network and can be used to check whether the network satisfies given safety and liveness properties.

As the network of CFSM's becomes complex, the number of generated states grows very rapidly. This problem is called *state space explosion* and several techniques were proposed to solve the problem. Here we only mention the techniques that are implemented in PROSPEC.

Fair reachability analysis is an improved technique for verifying networks of two communicating finite state machines [31]. Each new state in this analysis is generated by letting both machines make a single transition. In the fair reachability graph generated, each arc is labeled with the two transitions made in the individual machines. The fair reachability graph is an abstract representation of the reachability graph. It contains fewer states and transitions than the reachability graph, and is capable of detecting deadlocks and unspecified receptions [31], [37], [20] as well as unboundedness [38], [20]. Since the fair reachability graph does not encode all possible paths can be executed in a network, it cannot be used to check some of the liveness properties of the network.

Protocol projection is a method presented by Lam and Shankar [25] for constructing image protocols. An image protocol is obtained by aggregating the process states, messages, and events of the original protocol. An image protocol is specified like any real protocol, and is constructed in such a way that invariant and liveness properties of the image protocol are also properties of the original protocol. Thus, this method can be used to reduce the analysis of a multifunction protocol to the analyses of several smaller single-function protocols. An application of this method to verify a version of the HDLC protocol is presented in [33]. Image protocols have also been applied to specify protocol converters and have been found to be very useful for reasoning about the correctness of a protocol conversion [27].

Multiphase protocol construction is a method that addresses the state space explosion problem from the pro-

tocol construction point of view [12]. The construction of a multifunction protocol from a composition of single-function protocols is in general a difficult problem. However, many real-life protocols can be observed to go through different phases of behavior. In particular, these protocols go through their phases one at a time with a distinct function performed in each phase. For protocols characterized by this model of multiphase behavior, the following three-step methodology for constructing a multifunction protocol is proposed:

- Divide the protocol's functionality into separate functions.
- For each function, construct and verify a phase to perform this function. A phase is a network of communicating finite state machines that satisfies certain desirable general properties (including proper termination, and freedom from deadlocks and unspecified receptions) [12].
- Connect individual phases to form the required protocol. The resulting protocol is guaranteed by the construction method to be a phase and thus satisfies the same general properties of proper termination, and freedom from deadlocks and unspecified receptions.

Since the resulting protocols do not have to be verified from scratch, the multiphase protocol construction methodology also reduces the analysis of a complex multifunction protocol to the analyses of several smaller protocols.

III. Overview of the PROSPEC System

PROSPEC has a modular structure. Each important function of the system is realized by a tool (with the exception of the fair reachability analysis program). The user can invoke each tool independently. The hierarchy of tools within PROSPEC is shown in Fig. 1. PROSPEC also provides a tool called pdtool to facilitate invocation of the other tools and the fair reachability analysis program. The current version (1.1) consists of six tools:

1) The protocol design tool (pdtool) provides an interface between the user and the other tools. After starting the Suntools window environment, pdtool is invoked by typing "pdtool" in any Shelltool window.

2) The protocol editing tool (petool) provides commands for the user to specify the topology of a protocol system.

3) The machine editing tool (metool) is for specifying labeled directed graphs of which CFSM's constitute a special case.

4) The state exploration tool (setool) is an interactive tool for exploring all reachable states of a protocol (given a finite reachability graph). The reachability graph can be generated one portion at a time under the user's direction. The tool highlights problem states in the graph. The setool also includes all the functions of the metool so that when errors are discovered, the user can proceed immediately to modify the machines of the protocol.

5) The protocol projection tool (pptool) implements the method of projections [25]. The user can aggregate machine states and messages according to some resolution

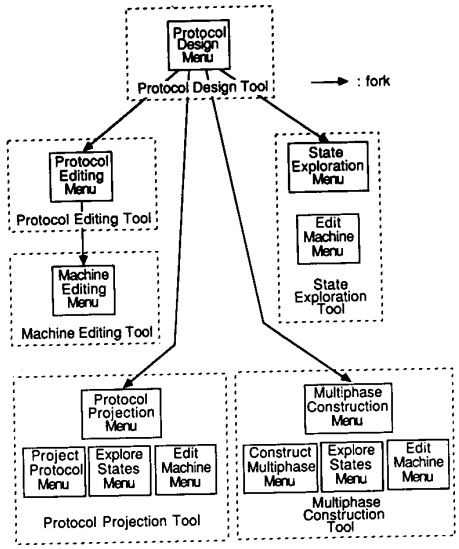

Fig. 1. Structure of the PROSPEC system.

he specifies. The tool constructs the corresponding image protocol. The image protocol events can be checked to see if they are well-formed. If not, the resolution of the aggregation may be reduced and a different image protocol constructed. The reachability graph of the image protocol can be generated within this tool.

6) The multiphase construction tool (mctool) implements the multiphase protocol construction methodology [12]. Each protocol, together with an exit set, can be specified and checked to see if it is a phase. The tool provides functions to connect phases together to form a larger protocol (also a phase) and to disconnect them. The internal details of machines in the resulting protocol are generated mechanically and the protocol is guaranteed by the methodology to terminate properly and to be free from deadlocks and unspecified receptions.

IV. Using PROSPEC to Specify Protocols

In the CFSM model, the topology, the machines, and the reachability graphs of a protocol are all labeled directed graphs. Only their semantic interpretations are different. With this idea in mind, we built a unified graphical interface for the user to create and manipulate labeled directed graphs. The same graphical interface is used to create and manipulate protocol topologies, machines, reachability graphs, and for input of the user's design/analysis decisions to the system. In the following we describe the specification features provided by PROSPEC.

Operations for Graph Manipulation: For the graphical interface of PROSPEC to be easy to use, it has to provide operations to manipulate the labeled directed graphs at several levels of granularity, i.e., node and edge symbols, subgraphs, and the graph as a whole. The set of operations also has to be complete in the sense that at each granularity level, the user should be able to carry out the operations of insertion, deletion, and modification. Fig. 2 shows the menu of metool and a DTE machine of the

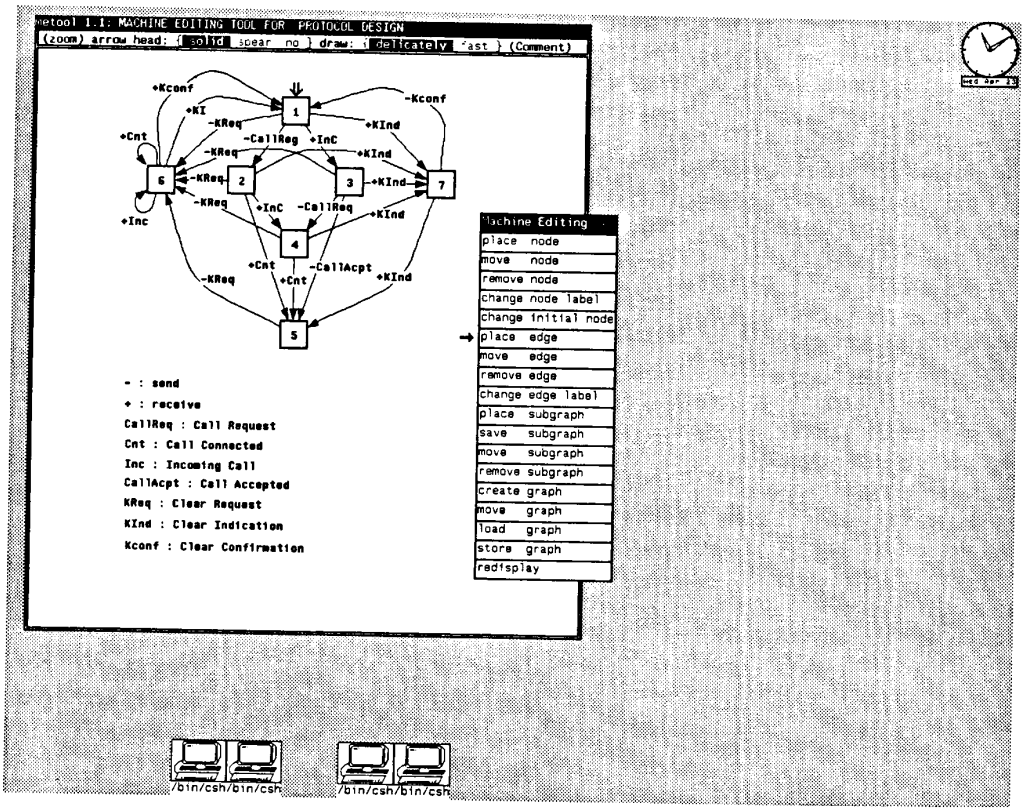

Fig. 2. An illustration of the machine editing tool.

X.25 call management protocol [18] created using metool. The menu lists all the operations provided by metool. The metool provides commands to store a subgraph or the whole graph as a file and to retrieve them later on.

Workstation Features: PROSPEC uses the menu-selection facility and mouse pointing device provided by the SUN workstation. Except for entering names and labels, all interactions are done using the mouse pointing device. For example, the selection of an operation is done by first popping up the menu and the clicking the middle button of the mouse when the cursor is on top of the corresponding menu item. The position of a new node is specified by moving the mouse cursor to the desired position and pressing the left mouse button. This sequence of actions is called "pointing." The shape of a new edge is defined by pointing first at the source node, then at the successive intermediate points of the edge, and finally at the destination node. The adjustment of a node position is done by first pointing at the node and then at the new position. In all, instead of typing a tedious sequence of text commands, PROSPEC edits labeled directed graphs in a natural and efficient manner.

Size of Graph: The tools of PROSPEC use dynamic memory allocation to request the memory needed for a newly created component. Therefore the size of the graph is only limited by the maximum memory space that can be allocated to a tool by the system. In a virtual memory system, the size of a graph can be very large.

Zooming Capability: PROSPEC provides zoom in/out commands to allow the user to select different sizes for displaying graphs and to focus on a specific area of a graph.

Example 1—DTE Machine of X.21: Fig. 3 shows the DTE machine of a version of the X.21 protocol [28] specified using the metool. Because the structures of DTE and DCE are quite similar, the DCE machine can be created by modifying the edge labels of the DTE machine from sending (− sign) to receiving (+ sign), and vice versa, and by deleting/inserting a few edges. With the help of PROSPEC's editing features, the specification of the X.21 protocol was quite easy. This version of the X.21 protocol was then verified using the setool and was found to have some unspecified reception states. It turned out that these errors were due to some typographical errors in the message labels of the DCE machine in [28].

V. Using PROSPEC to Analyze Protocols

PROSPEC implements both the reachability analysis and fair reachability analysis methods. The reachability analysis function can be invoked from setool, pptool, or mctool. The fair reachability analysis function can only be invoked from pdtool. Since the reachability analysis method implemented by PROSPEC has a built-in interactive capability, we call it *interactive reachability analysis*. Next we discuss the features of each analysis function and show how to use them.

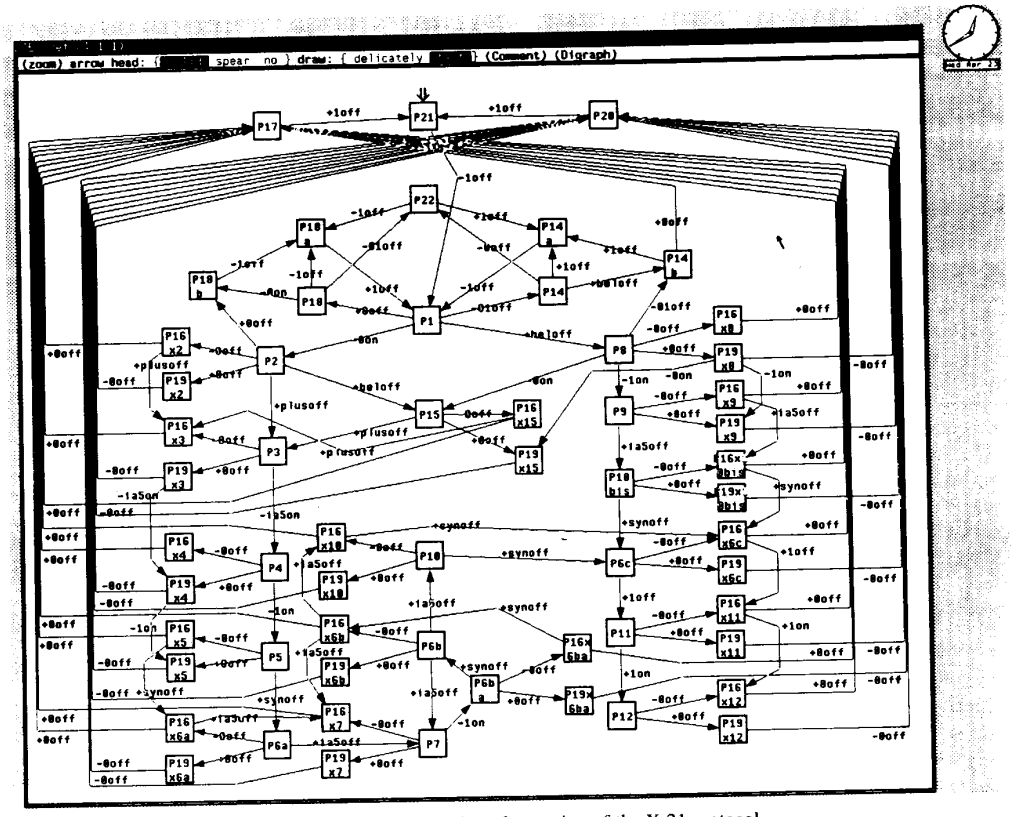

Fig. 3. The DTE machine of a version of the X.21 protocol.

A. Interactive Reachability Analysis

In PROSPEC reachability graphs are displayed graphically. This feature provides the user with important visual information about the structure of a reachability graph, which is very hard to capture if reachability graphs are displayed in text form. For example, it is easier for a user to check paths in a graphical display. During the verification process, it also indicates whether the state exploration process is expanding or converging.

Since the tools have a built-in interactive capability, the reachability graph can be generated and displayed portion by portion as directed by the user. Thus, the user has control over the direction of the state exploration process. We refer to this capability as *focus exploration*.

To facilitate a reachability analysis, PROSPEC highlights all "problem" states in the reachability graph, e.g., deadlock states, unspecified reception states, proper termination states, and states that are yet to be explored. The path from the initial state to a specific state can be shown by clicking at the specific state. PROSPEC can also show the execution of a protocol along a path by highlighting the current executing edges or nodes of the machines at each execution step. These functions greatly enhance the debugging capability of PROSPEC.

In PROSPEC, both the reachability analysis function and the machine editing function can be accessed from the setool. Therefore, after errors have been found during a reachability analysis, the machines can be modified very quickly and then analyzed again.

Example 2—A Version of the IBM BSC Call Setup Protocol: Fig. 4 shows a version of the IBM BSC call setup protocol [12] created by PROSPEC. The primary machine is at the upper left corner of the window. The secondary machine is at the upper right corner of the window. By selecting the "RG from the initial state" menu item the user can generate the reachability graph of this protocol which is then moved to the lower portion of the window by using the "move RG" menu item. (Fig. 5 explains the details of the state symbol.) In Fig. 4, the character string "P1" in arc labels identifies the transitions made by the primary machine, while the "P2" string identifies those made by the secondary machine. Because of the limited space allocated for displaying a state, abbreviation characters are used to represent message names. A name resolution algorithm is implemented to give each message a unique abbreviation character. The message table is displayed at the upper right corner of the window. In Fig. 4 two proper terminating states, states 18 and 21, are highlighted.

Example 3—A Version of the Telnet Document Transfer Protocol: Fig. 6 shows the analysis result of a version of the Telnet document transfer protocol [10]. The sender and receiver machines are shown at the upper left corner and the reachability graph is shown to their right. 43 reachable states are generated. There are no deadlock

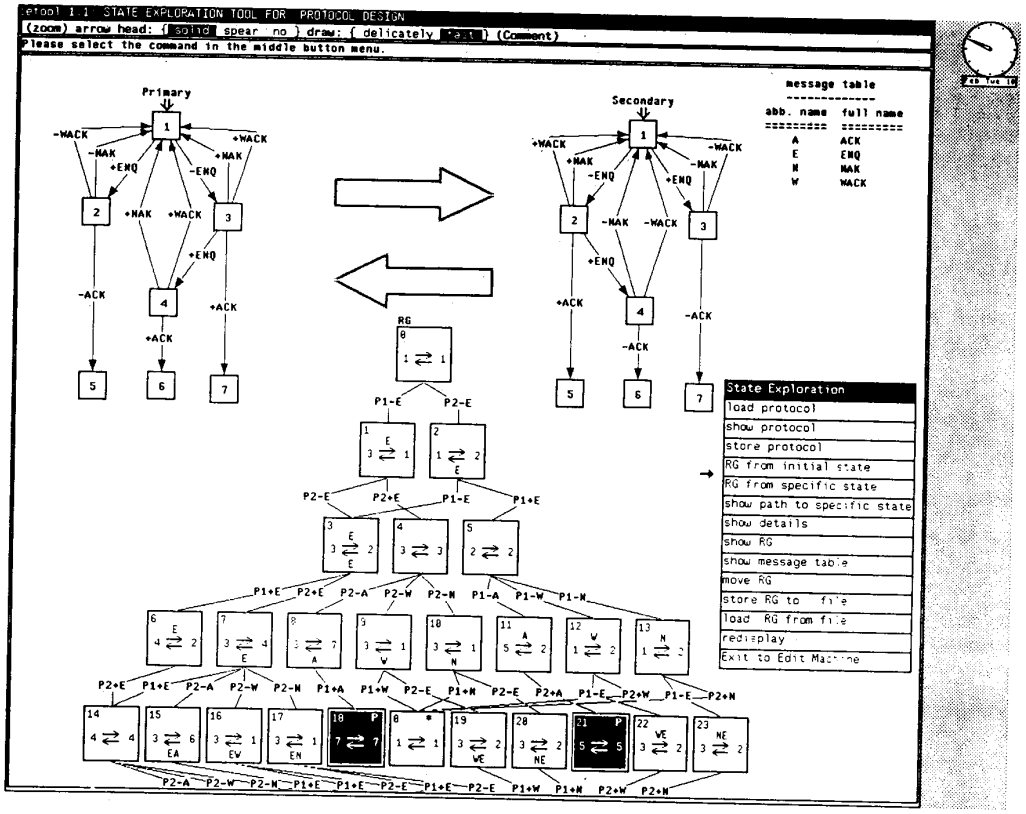

Fig. 4. An illustration of the state exploration tool.

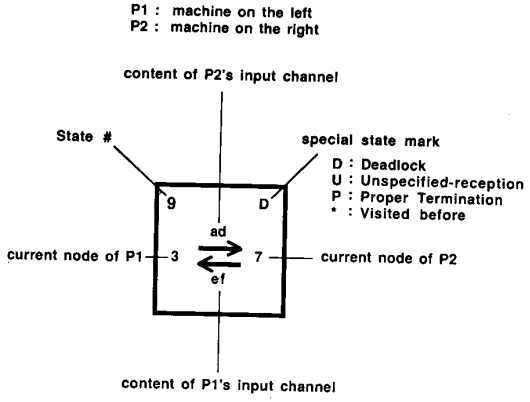

Fig. 5. Details of the state symbol.

states or unspecified reception states. The highlighted state, state 11, shows that the input channel of the receiver needs to have a buffer which can accommodate five messages. The reachability graph also indicates that the other channel needs to have a buffer which can store two messages.

Example 4—A Version of the IBM BSC Data Transfer Protocol: Fig. 7 shows a path which leads to a deadlock state in a version of the IBM BSC data transfer protocol [12]. The machines are displayed with the nodes corresponding to the highlighted global state highlighted. The user can click a state (or arc) along the deadlock path, and the highlighted nodes (or edges) in the machines and channels provide an animation effect which helps the user gain insight on what is wrong with the protocol implementation.

B. Fair Reachability Analysis

In fair reachability analysis, by letting both machines progress with the same speed, the size of a fair reachability graph is smaller than that of the corresponding reachability graph. In some cases, a fair reachability graph has a finite number of states while the corresponding reachability graph has infinitely many states.

In the current version of PROSPEC, the fair reachability analysis program is invoked from pdtool, and the generated fair reachability graphs are displayed in text form. Since the displaying window is in **vi** editing mode, the user can use **vi** editing commands to scan the text and search for any problem states.

Example 5—A Call Management Protocol: Fig. 8 shows a call management protocol and its fair reachability graph generated by the fair reachability analysis program. There are only six reachable states in the fair reachability graph. Note that the number of states in the reachability graph is infinite, since with asynchronous operation, machine P1 at node 4 can keep on sending messages to machine P2.

The analysis shows that there is an unspecified reception state. In state 1, machine P2 is in node 3, a receiving

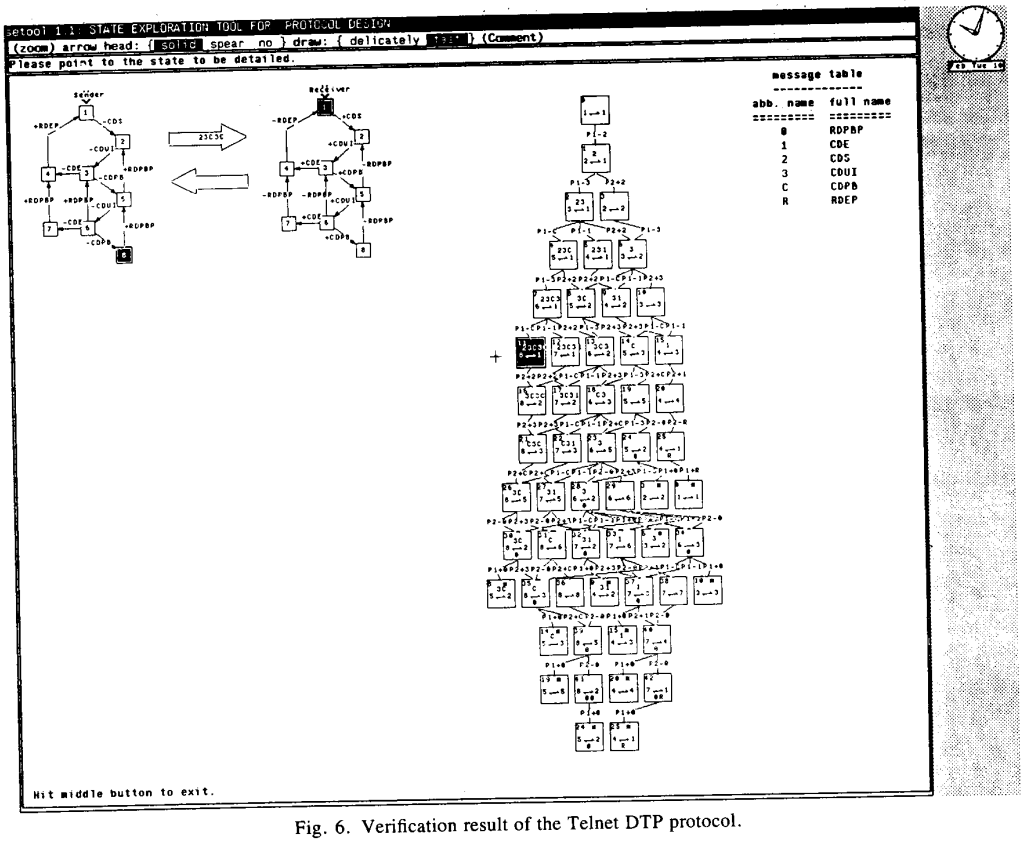

Fig. 6. Verification result of the Telnet DTP protocol.

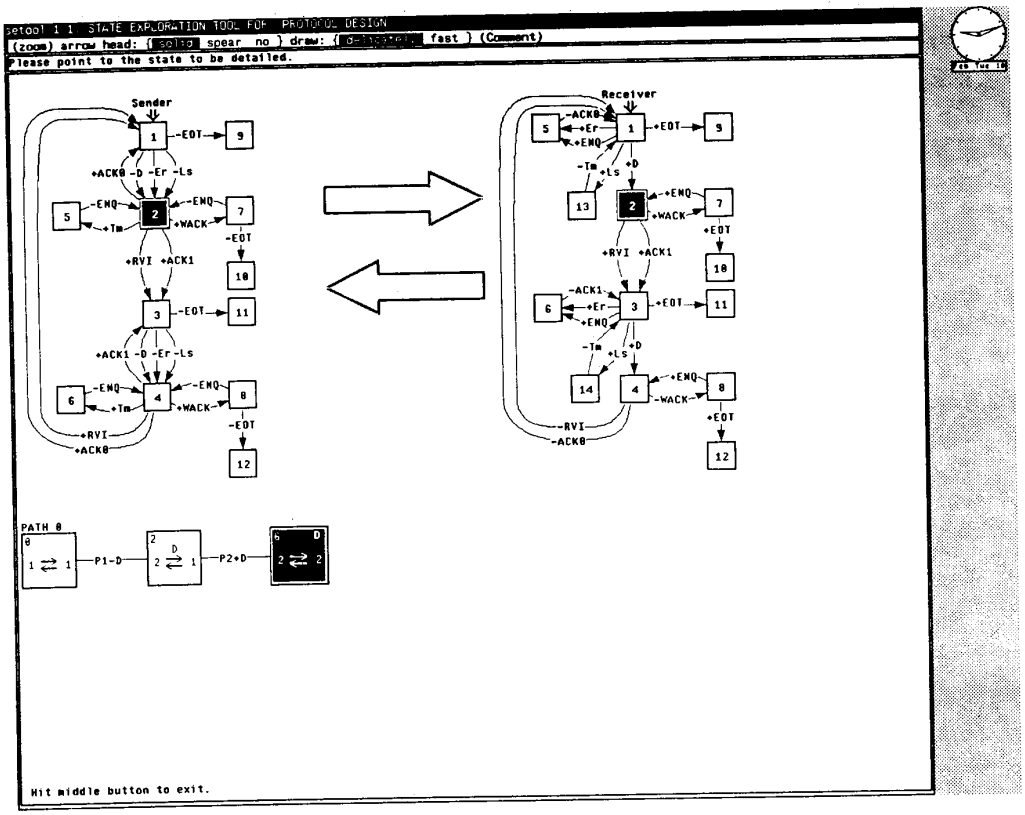

Fig. 7. A deadlock path and details of a deadlock state.

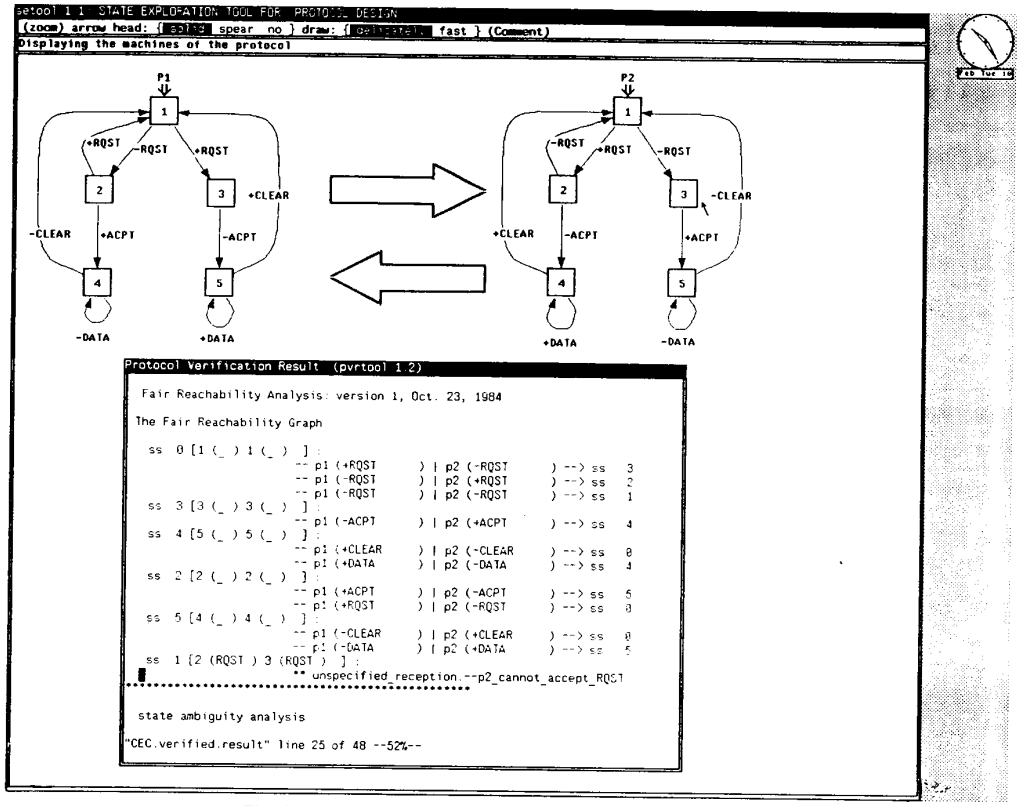

Fig. 8. An illustration of the fair reachability analysis program.

node, and cannot receive message "RQST". That error results from the misplacement and mislabeling of an edge in machine P2. The edge from node 2 to node 1 labeled "−RQST" should be deleted, while an edge labeled "+RQST" should be added from node 3 to node 1.

Here we take advantage of the multiwindow feature of the SUN system: displaying the graphical representation of the protocol using setool in one window while displaying the fair reachability analysis results in the other window. Similarly, we can use this feature to compare different protocols.

VI. Protocol Projection

In PROSPEC, protocol projection functions are invoked from pptool. The analysis of a protocol via protocol projections follows these steps:

1) Based upon the desired safety and liveness properties to be checked, the user specifies a partitioning of the state space of each machine and asks PROSPEC to construct an image protocol.

2) The user asks PROSPEC to check well-formedness of the image protocol events (well-formedness is needed for liveness properties only). If there is a machine with events that are not well-formed, the user will specify a new partitioning of the state space of the machine (with a higher resolution [25]) and ask PROSPEC to construct a new image protocol.

3) The user asks PROSPEC to generate the image protocol's reachability graph and the user checks to see if the image protocol satisfies the desired safety and liveness properties. Both safety and liveness assertions, however, are restricted to formulas containing references to image machine states, image messages, and image events; each reference to x' in such an assertion is interpreted for the original protocol as *some x whose image is x'*, where x denotes a machine state, a message, or an event. (The reader is referred to [25], [27] for a more detailed explanation.) Because the reachability graph of the image protocol is smaller than that of the original protocol, the time needed to analyze the image protocol is less. However, the image protocol might not have been specified with adequate resolution for verifying the desired safety and liveness properties, in which case the user will have to try a different image protocol.

Example 6—A Protocol Projection Example: Here we use the protocol example in [25] to illustrate the use of pptool. In Fig. 9, the upper part of the window displays a protocol and its state space partition which has been specified by the user. Each partition is represented by a dashed-box. The image protocol is shown in the lower portion of the screen together with image message tables. The well-formedness of the image protocol events is checked by selecting the "check well-formedness" menu item. It turned out that this image protocol is well-formed. To verify the image protocol, the "Exit to Explore State" menu item is selected to generate the reachability graph. Fig. 10 shows the reachability graph of the image protocol. There is a deadlock state, state 4, and it is high-

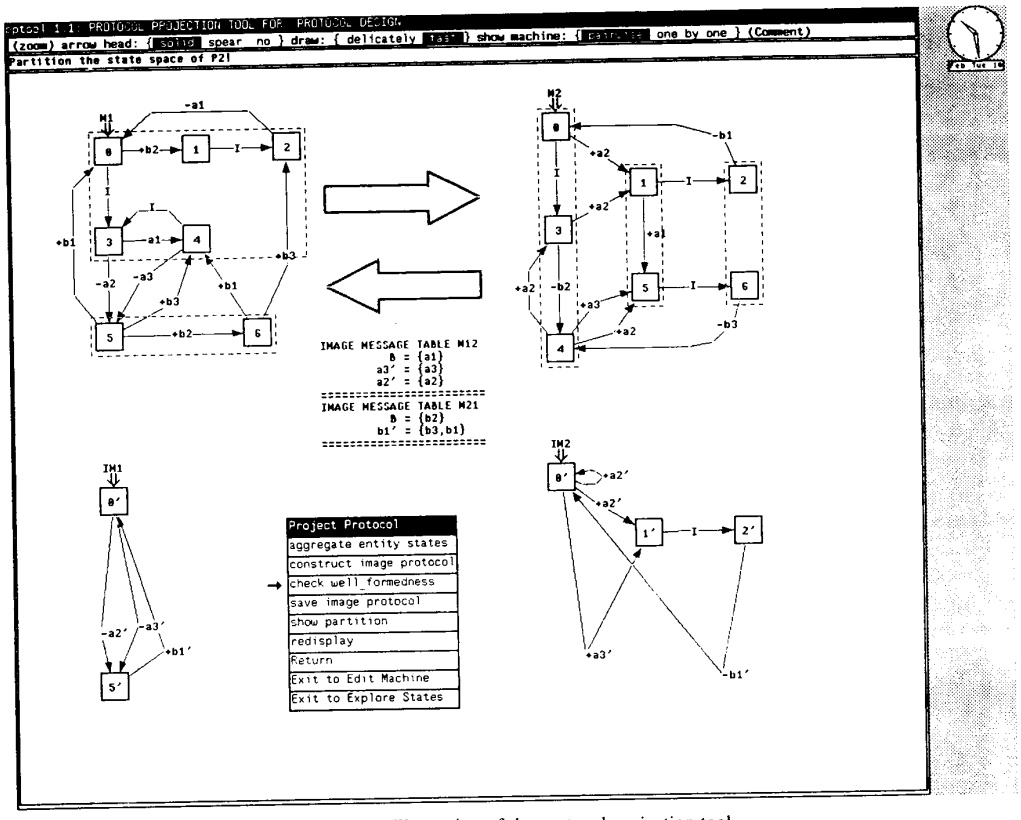

Fig. 9. An illustration of the protocol projection tool.

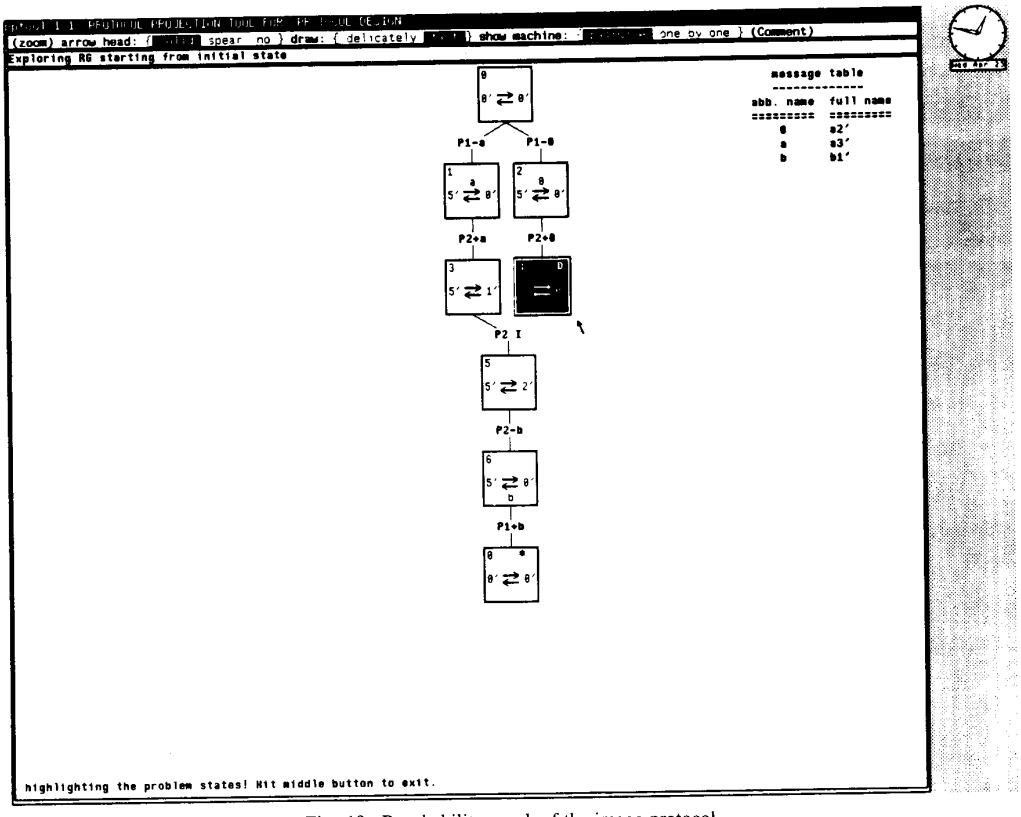

Fig. 10. Reachability graph of the image protocol.

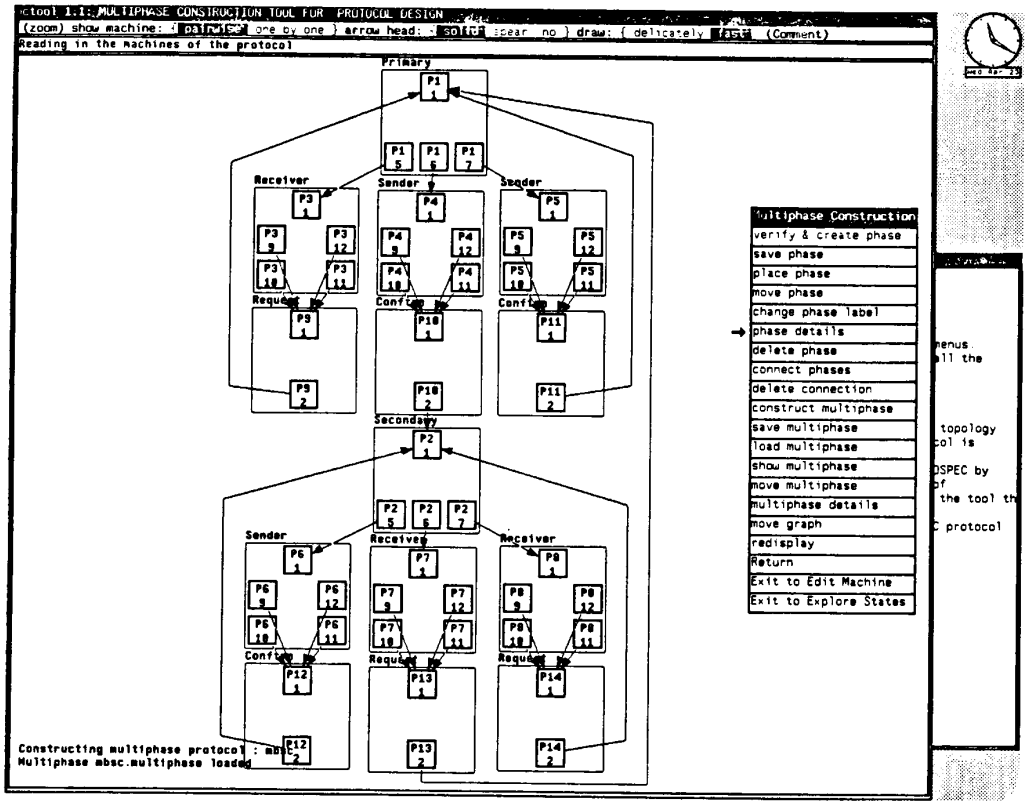

Fig. 11. An illustration of the multiphase construction tool.

lighted. A deadlock state in the image protocol implies that the original protocol can reach the set of states that have state 4 as their image. And once inside this set of states, there is no enabled event to take the protocol out of it.

Note that while the reachability graph of the image protocol has only seven states, that of the original protocol has an infinite number of states. The sending loop between node 3 and node 4 of Machine M1 can send out an infinite number of messages. The edges labeled with "I" represent internal transitions.

VII. Multiphase Protocol Construction Methodology

The multiphase protocol construction methodology is implemented by mctool. Here we use a version of the IBM BSC data link protocol [12] as an example to illustrate the use of mctool. In the BSC protocol, there are three basic phases: call setup phase, data transfer phase, and call clear phase. An execution of the protocol will go through these three phases sequentially. In the call setup phase, when both machines want to send data, the primary machine will have the higher priority to send. In the following exercise, we construct a modified version of the BSC protocol where scheduling between the two machines is fair. We achieve this goal by alternating the priority of the machines.

Example 7—A Modified Version of the BSC Protocol: To construct the modified BSC protocol, the three basic phases together with their exit nodes are first specified and verified. Then the primary machine of the modified BSC protocol is constructed by connecting *copies* of basic phases together. Fig. 11 shows the main menu of mctool and the primary machine of the modified BSC protocol constructed.

Only the initial node and the final nodes of the primary machine in each copy of a phase are displayed. The user can select the "phase detail" menu item to examine the details of a phase. Note that the methodology allows us to neglect the internal structures of phases and to concentrate on the interconnection of phases.

Having constructed the multiphase protocol, the user can select the "multiphase detail" menu item to generate the details of the machines. Fig. 12 shows the internal structure of the primary machine of the modified BSC protocol.

VIII. Extensions to PROSPEC

To make PROSPEC even more useful, several extensions are desirable:

1) *Integrate other verification techniques.* There are other verification techniques that can further enhance the capabilities of PROSPEC. For example, the closed cover technique [22] and the model checker [15] are two possible candidates.

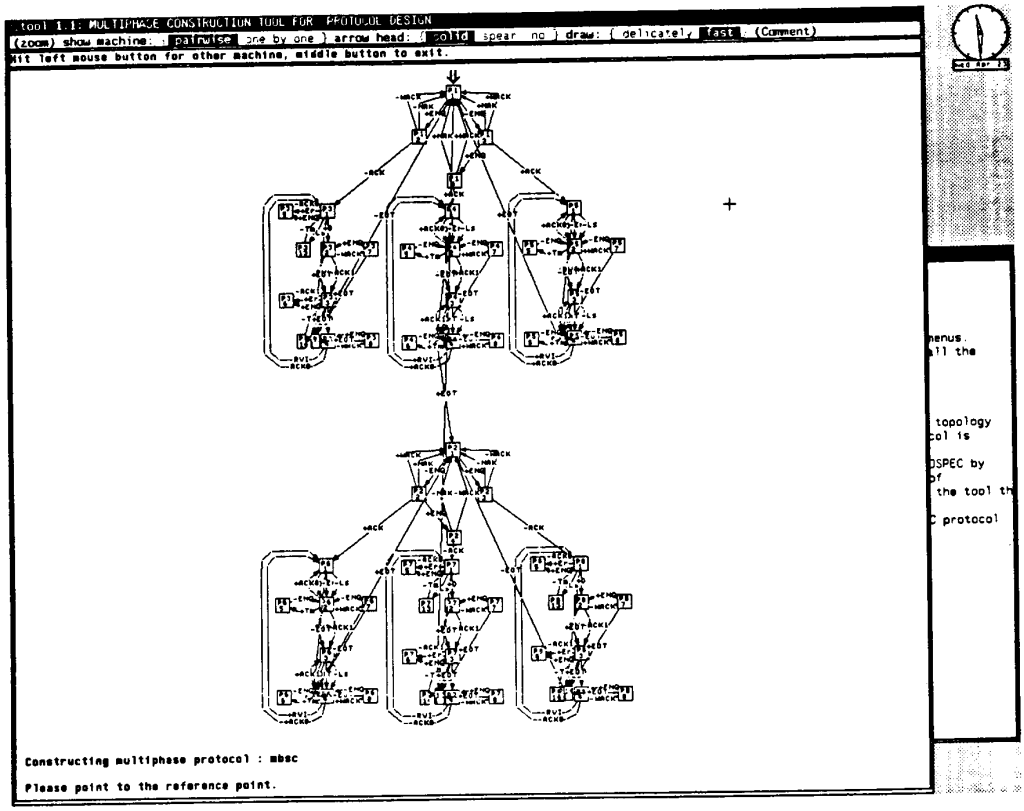

Fig. 12. The primary machine of the modified BSC protocol.

2) *Implement tools that generate executable code.* The protocol generated by PROSPEC can be considered a communication skeleton. Each protocol machine can be further augmented with data structures and internal operations. The resulting representation can then be used to mechanically generate executable code.
3) *Implement tools for protocol simulation.* After a protocol is constructed and verified by PROSPEC, it is desirable to have a tool for specifying the network environment in which the protocol is to be executed, and for carrying out a simulation of the performance of the protocol.

The tools of PROSPEC have been modified to do other jobs. The petool has been modified and used as a graphical interface to specify the network topology for an interactive network design tool [23]. Currently, Chow is modifying PROSPEC to design and analyze finite state tables that are used to implement services of an experimental telephone network.

IX. Conclusions

We have shown how to use PROSPEC to specify and verify communication protocols. Seven protocol examples were presented here to illustrate the features of PROSPEC. The DTE machines of X.25 and X.21 protocols were specified using metool to demonstrate the specification capability. A version of the IBM BSC call setup protocol, a version of the Telnet document transfer protocol, and a version of the IBM BSC data transfer protocol were analyzed using setool to demonstrate interactive reachability analysis and the graphical interface. A call management protocol was used to illustrate the use of the fair reachability analysis program. A protocol adapted from [25] was used to demonstrate pptool which implements the protocol projection method. A modified version of the IBM BSC protocol was constructed using mctool which implements the multiphase protocol construction methodology.

We found PROSPEC to be useful for specifying and verifying a variety of communication protocols. The unified graphical interface and interactive capability of PROSPEC make it easy to use. They also speed up the protocol design process. The availability of tools that implement abstraction and modular construction techniques, in addition to tools that implement interactive reachability analysis, further increases the effectiveness of the graphical user interface of PROSPEC.

References

[1] S. Aggarwal and R. P. Kurshan, "Automated Implementation from Formal Specification," in *Protocol Specification, Testing, and Verification IV*, Y. Yemini *et al.*, Eds. Amsterdam, The Netherlands: North-Holland, 1985.
[2] S. Aggarwal, D. Barbará, and K. Z. Meth, "SPANNER: A tool for

the specification, analysis, and evaluation of protocols," *IEEE Trans. Software Eng.*, vol. SE-13, no. 12, pp. 1218-1237, Dec. 1987.
[3] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A note on reliable full-duplex transmission over half-duplex links," *Commun. ACM*, vol. 12, pp. 260-261, May 1969.
[4] T. P. Blumer and D. P. Sidhu, "Mechanical verification and automatic implementation of communication protocols," *IEEE Trans. Software Eng.*, vol. SE-12, no. 8, pp. 827-843, Aug. 1986.
[5] G. v. Bochmann, "Finite state description of communication protocols," *Comput. Networks*, vol. 2, no. 4/5, pp. 361-372, Oct. 1978.
[6] G. v. Bochmann and C. Sunshine, "Formal methods in communication protocol design," *IEEE Trans. Commun.*, vol. COM-28, pp. 624-631, Apr. 1980.
[7] G. v. Bochmann *et al.*, "Experience with formal specifications using an extended state transition model," *IEEE Trans. Commun.*, vol. COM-30, no. 12, pp. 2506-2513, Dec. 1982.
[8] D. Brand and P. Zafiropulo, "On communicating finite-state machines," *J. ACM*, vol. 30, no. 2, pp. 323-342, Apr. 1983.
[9] T. Y. Choi and R. E. Miller, "A decomposition method for the analysis and design of finite state protocol," in *Proc. Eighth Data Commun. Symp.*, pp. 167-176, Oct. 1983.
[10] T. Y. Choi, "Formal techniques for the specification, verification and construction of communication protocols," *IEEE Commun. Mag.*, vol. 23, no. 10, pp. 46-52, Oct. 1985.
[11] C. H. Chow, M. G. Gouda, and S. S. Lam, "An exercise in constructing multi-phase communication protocols," in *Proc. ACM SIGCOMM'84 Symp.*, pp. 493-500, June 1984.
[12] ——, "A discipline for constructing multiphase communication protocols," *ACM Trans. Comput. Syst.*, vol. 3, no. 4, pp. 315-343, Nov. 1985.
[13] C.-H. Chow, "A discipline for verification and modular construction of communication protocols," Ph.D. dissertation, Dep. Comput. Sci., Univ. Texas at Austin, Dec. 1985.
[14] ——, "Using PROSPEC to design and verify communication protocols," in *Proc. GLOBCOM '86*, Dec. 1986.
[15] E. M. Clark, E. A. Emerson, and A. P. Sistla, "Automated verification of finite state concurrent system using temporal logic," in *Proc. 10th ACM POPL Conf.*, 1983.
[16] D. M. Cohen and E. J. Isganitis, "Automatic generation of a prototype of a new protocol from its specification," in *Proceeding of GLOBECOM '86*, Dec. 1986.
[17] M. G. Gouda, "An example for constructing communicating machines by stepwise refinement," in *Proc. 3rd IFIP Workshop Protocol Specification Testing, and Verification*, H. Rudin and C. H. West, Eds. Amsterdam, The Netherlands: North-Holland, 1983, pp. 63-74.
[18] M. G. Gouda and Y. T. Yu, "Protocol validation by maximal progress state exploration," *IEEE Trans. Commun.*, vol. COM-32, no. 1, pp. 94-97, Jan. 1984.
[19] M. G. Gouda and C. K. Chang, "Proving liveness for networks of communicating finite state machine," *ACM Trans. Program. Lang. Syst.*, vol. 8, no. 1, pp. 154-182, Jan. 1986.
[20] M. G. Gouda, C. H. Chow, and S. S. Lam, "On the decidability of livelock detection in networks of communicating finite state machines," in *Proc. 4th Int. Workshop Protocol Specification, Testing and Verification*, June 1984.
[21] M. G. Gouda and Y. T. Yu, "Synthesis of communicating machines with guaranteed progress," *IEEE Trans. Commun.*, vol. COM-32, no. 7, pp. 779-788, July 1984.
[22] M. G. Gouda, "Closed covers: To verify progress for communicating finite state machines," *IEEE Trans. Software Eng.*, vol. SE-10, no. 6, pp. 846-855, Nov. 1984.
[23] C.-T. Hsieh, "Models and algorithms for the design of store-and-forward communication networks," Ph.D. dissertation, Dep. Comput. Sci., Univ. Texas at Austin, 1987.
[24] S. S. Lam, "Data link control procedures," in *Computer Communications, Vol. 1, Principles*, W. Chou, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1983, ch. 3, pp. 81-113.
[25] S. S. Lam and A. U. Shankar, "Protocol verification via projections," *IEEE Trans. Software Eng.*, vol. SE-10, no. 4, pp. 325-342, July 1984.
[26] S. S. Lam, C.-H. Chow, M. G. Gouda, and A. U. Shankar, "Interactive verification and construction of communication protocols in PROSPEC," in *Proc. IEEE INFOCOM'86*, 1986, pp. 67-75.
[27] S. S. Lam, "Protocol conversion," Technical Report TR-87-05, Dep. Comput. Sci., Univ. Texas at Austin, Feb. 1987; see also *IEEE Trans. Software Eng.*, this issue, pp. 353-362.
[28] C. V. Ramamoorthy, S. T. Dong, and Y. Usuda, "An implementation of an automated protocol synthesizer (APS) and its application to the X.21 protocol," *IEEE Trans. Software Eng.*, vol. SE-11, no. 9, pp. 886-908, Sept. 1985.
[29] R. R. Razouk and G. Estrin, "Modeling and verification of communication protocols in SARA: The X.21 interface," *IEEE Trans. Comput.*, vol. C-29, no. 12, pp. 1038-1052, Dec. 1980.
[30] R. Razouk, "Modeling X.25 using the graph model of behavior," in *Proc. 2nd Int. Workshop Protocol Specification, Testing and Verification*, May 1982.
[31] J. Rubin and C. H. West, "An improved protocol validation technique," *Comput. Networks*, Apr. 1982.
[32] L. E. Rosier and M. G. Gouda, "Deciding progress for a class of communicating finite state machines," in *Proc. Conf. Information Sciences and Systems*, Princeton Univ., 1984.
[33] A. U. Shankar and S. S. Lam, "An HDLC protocol specification and its verification using image protocols," *ACM Trans. Comput. Syst.*, vol. 1, no. 4, pp. 331-368, Nov. 1983.
[34] M. Sherman and H. Rudin, "Using automated validation techniques to detect lockups in packet-switched networks," *IEEE Trans. Commun.*, vol. COM-30, no. 7, pp. 1762-1767, July 1982.
[35] C. Sunshine, "Formal techniques for protocol specification and verification," *Computer*, vol. 12, no. 9, pp. 20-27, Sept. 1979.
[36] C. H. West and P. Zafiropulo, "Automated validation of a communications protocol: The CCITT X.21 recommendations," *IBM J. Res. Develop.*, vol. 22, pp. 60-71, Jan. 1978.
[37] Y. T. Yu and M. G. Gouda, "Deadlock detection for a class of communicating finite state machines," *IEEE Trans. Commun.*, vol. COM-30, no. 12, pp. 2514-2518, Dec. 1982.
[38] ——, "Unboundedness detection for a class of communicating finite-state machines," *Inform. Processing Lett.*, vol. 17, pp. 235-240, Dec. 1983.
[39] P. Zafiropulo *et al.*, "Towards analyzing and synthesizing protocols," *IEEE Trans. Commun.*, vol. COM-28, no. 4, pp. 651-661, Apr. 1980.

Ching-Hua Chow (S'80-M'86) received the B.S. degree in electrical engineering from National Taiwan University in 1977 and the M.A. and Ph.D. degrees in computer science from the University of Texas at Austin in 1982 and 1985, respectively.

Since 1985, he has been a Member of Technical Staff in the Network Systems and Services Research Laboratory at Bell Communications Research in Morristown, NJ, where his work involves specification, verification, and implementation of network services and communication protocols in an intelligent network testbed called the Modular Integrated Communications Environment (MICE). He is currently investigating the design of a mixed language distributed service programming environment, multimedia communication system, and the protocol conversion issues between ISDN and non-ISDN networks. His research interests include computer networks and protocols, distributed systems, and programming environments.

Simon S. Lam (S'69-M'74-SM'80-F'85), for a photograph and biography, see this issue, p. 362.