INTERACTIVE VERIFICATION AND CONSTRUCTION OF COMMUNICATION PROTOCOLS IN *PROSPEC*

Simon S. Lam*, Ching-Hua Chow*, Mohamed G. Gouda
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
and
A. Udaya Shankar*
University of Maryland
College Park, Maryland 20742

ABSTRACT

The PROSPEC system has been developed on a SUN 2/120 workstation for interactive protocol verification and construction. With the model of communicating finite state machines, it is most convenient and efficient for a human designer to specify protocols graphically and also to verify (or 'debug') protocols by looking at displays of reachability graphs. PROSPEC has a graphical user interface. The graphical interface is not, however, the most important element of PROSPEC. The attractiveness of PROSPEC lies in the user's ability to access tools that implement techniques for managing the complexity of protocol specification and verification and for the modular construction of protocols. In this paper, we give an overview of PROSPEC and two of the techniques it implements (method of projections and multiphase protocol construction).

1. INTRODUCTION

A layered communications architecture facilitates the construction of networking software in a modular fashion. Nevertheless, each protocol layer consists of a set of complex parallel programs with multiple functions to perform. For example, a data link control protocol typically has three functions: connection management and one-way data transfers in opposite directions. During the past several years, we have been investigating various abstraction and composition techniques for reducing the analysis/construction of a multifunction protocol to the analysis/construction of smaller single-function protocols.

The *resolution* of a protocol system was proposed by Lam and Shankar as the basis for developing abstraction techniques to simplify the analysis and construction of multifunction protocols [1]. Roughly speaking, for a protocol system modeled as a network of processes that interact by message-passing, its resolution is measured by the number of distinct process states and distinct messages. To verify a particular logical property of a protocol, however, the *observable resolution* can be much lower than the actual resolution of the protocol system. They developed the method of projections for constructing *image protocols*. An image protocol is specified just like any real protocol but is smaller than the original protocol. It must be constructed in such a way that it will not mislead us into making any false statement about the behavior of the original protocol. Obviously, fewer logical properties are observable and verifiable in an image protocol than in the original protocol. Lam and Shankar presented a method for constructing a sequence of image protocols with increasing resolution until one is found with enough resolution for verifying the logical property in question. This approach was found to be very effective for the analysis of multifunction protocols that are not easily decomposable into different modules for implementing different functions, due to the use of shared variables and shared messages. Their method has been applied to verify a version of the HDLC protocol [2].

The construction of a multifunction protocol from a composition of single-function protocols is a much harder problem. There is no easy method that corresponds to an "inverse projection" operation. Many real-life protocols, however, can be observed to go through different *phases* performing a distinct function in each phase. Chow, Gouda, and Lam defined formally the concept of a phase. They presented a multiphase model for protocols and the following three-step methodology for constructing multiphase protocols:

- Identify the distinct functions of a multifunction protocol.
- Construct and verify a phase to perform each function. (A phase is a network of communicating finite state machines that has certain desirable properties, including proper termination, and freedom from deadlocks and unspecified receptions.)

*Work supported by National Science Foundation Grant No. ECS 83-04734.

- Connect individual phases together so that the resulting protocol is also a phase, and is thus guaranteed to possess the desirable properties of phases.

This last step in the methodology was the basic contribution in [3]. Chow, Gouda, and Lam also presented a sufficient condition for a multiphase protocol to have bounded communications given that its constituent phases have bounded communications. They illustrated their methodology with the construction of several nontrivial multiphase protocols, including a version of IBM's BSC protocol for data link control [3] and a high-level session control protocol [4].

The model of communicating finite state machines (CFSMs) has been used with success for the specification and verification of many communication protocols. With this model, it is most convenient and efficient for a human designer to specify FSMs 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 tools for protocol design and reachability analysis have been developed, such as the ones developed at IBM Research [5] and at Bell Laboratories [6].)

We have developed the programming environment PROSPEC for interactive protocol verification and construction with more than just the above objective. Following our efforts on the protocol projection method, the multiphase protocol construction method, and several other abstraction and composition techniques, we came to realize that a user-friendly graphical interface in itself is not adequate to make life easy for protocol designers. The inherent weakness of the CFSM model is its inability to deal with FSMs with many states and the resulting large (and possibly infinite) reachability graph. Even if a graph is finite it may be too large to display on a screen. Thus the abstraction and modular construction techniques that we have developed, in addition to reducing the analysis/construction of protocols from a large problem to a hierarchy of small problems, will also considerably enhance the effectiveness of an interactive graphical interface. At the same time, the interactive graphical interface enhances the effectiveness of the techniques.

PROSPEC has been developed on a SUN 2/120 workstation running 4.2 BSD 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 projections, multiphase protocol constructions, and some others. In this respect, two features of PROSPEC are significant. First, a standard internal representation of data facilitates the passing of FSMs 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 has to enter for interaction with PROSPEC [7,8].

This paper is organized as follows. In Section 2, we give an overview of the method of projections. In Section 3, we give an overview of the multiphase protocol construction methodology. In Section 4, we show the structure of tools and menus available in PROSPEC and also illustrate the menus themselves. The current PROSPEC environment supports only the CFSM model and state exploration methods. (The method of projections, however, is *not* limited to the CFSM model. This is illustrated by its application in [2] where an HDLC protocol and its image protocols are specified using a programming language notation and verified using a deductive inference method.)

2. METHOD OF PROJECTIONS [1]

A protocol system consists of a network of protocol entities and channels. At any time, the *global state* of the system is specified by a joint description of the states of the entities and channels. Let G denote the set of all global states of the protocol system. The states of entities and channels (and hence the global state) may change due to the occurrence of certain events: entities sending messages, entities receiving messages, timeouts, channel errors, etc. These global state transitions define a directed graph on G. Given any initial global state g_0, the portion of the graph that is reachable from g_0 is referred to as the *reachability graph* R. R contains all available information on the logical properties of the protocol system.

Specifically, assertions of liveness properties are predicates on the set of paths in G. A liveness assertion is valid if it is satisfied by the paths in R. Assertions of safety properties are predicates on G. Let R_s denote the set of states reachable from g_0. A safety assertion is valid if it is satisfied by the states in R_s.

Verification of these properties may be carried out by a brute-force state exploration (in the case of a small finite R), or by proof techniques for parallel programs. The method of projections can be used in conjunction with either verification approach.

Consider a protocol with several distinguishable functions. We would like to ask questions regarding the logical behavior of the protocol system concerning these functions. Instead of asking such questions all at the same time, we may ask them with respect to one function of the protocol at a time. Our analysis approach avoids a characterization of R. Instead, we construct from the given protocol an image protocol for each of the functions that are of interest to us.

An *image protocol* is specified just like any real protocol. The states, messages and events of entities in an image protocol are obtained by *aggregating* groups of states, messages and events of the corresponding entities in the original protocol. Definitions needed for the construction of an image protocol are presented in [1].

Given an image protocol, suppose that a second image protocol is obtained by aggregating some of the entity states, messages and events of the first one. We say that the second image protocol has a lower *resolution* than the first image protocol. The original protocol can be thought of as an image protocol of itself, and obviously it has the highest resolution available.

Due to the aggregations, an image protocol is smaller than the original protocol and is typically easier to analyze. However, the reachability graph of an image protocol captures only part of the logical behavior of the original protocol. The following useful properties of image protocols are proved in [1].

First, any safety property that holds for an image protocol must also hold for the original protocol. Second, if an image protocol is constructed with sufficient resolution so that its events satisfy a well-formed property, then it is *faithful*: Any logical property, safety or liveness, of the image protocol holds *if, and only if,* it holds in the original protocol. The well-formedness of an image protocol is determined by checking protocol entities individually. The well-formed property is the weakest sufficient condition for faithfulness that can be stated *without any knowledge of R.*

Given a protocol and an assertion A_0 stating some desired logical behavior of the protocol, our objective is to construct the smallest image protocol that is of sufficient resolution to verify A_0. Towards this goal, we construct a sequence of image protocols of increasing resolution by a stepwise refinement process. The initial image protocol can be determined by the resolution needed to *describe* A_0. The stepwise refinement is terminated when an image protocol with sufficient resolution to *verify* A_0 is constructed. Two stepwise refinement algorithms are presented in [1] with termination conditions based upon the two image protocol properties mentioned above.

Given a multifunction protocol, a faithful image protocol can always be obtained for each function by adjusting its resolution. However, the successful construction of faithful image protocols that are much smaller than the original protocol depends upon whether the protocol has a good structure. Thus, one can think of a multifunction protocol as being *well-structured* if it possesses small faithful image protocols for its functions.

3. MULTIPHASE PROTOCOLS AND THEIR CONSTRUCTION [3]

A discipline for constructing multiphase communication protocols was developed for the CFSM model [3]. In this model, a machine M is a directed labelled graph with two types of edges, namely *sending* and *receiving edges.* A sending (or receiving) edge is labelled -f (or +f, respectively) for some *message* f in a finite set F of messages. A node in M whose outgoing edges are all sending (or all receiving) edges is called a *sending* (or *receiving*, respectively) *node.* A node in M whose outgoing edges include both sending and receiving edges is called a *mixed node*, and a node in M that has no outgoing edges is called a *final node.* One of the nodes in M is identified as its *initial node*, and each node in M is reachable by a directed path from the initial node.

Let M and N be two machines with the same set F of messages; the pair (M,N) is called a *network* of M and N.

A *state* of network (M,N) is a four-tuple [v,w,x,y], where v and w are nodes in M and N respectively, and x and y are strings over the messages in F. Informally, a state [v,w,x,y] means that the executions of M and N have reached nodes v and w respectively, while the input channels of M and N store the strings x and y respectively.

Let M and N be two machines. The network (M,N) is called *safe* if, and only if, its communication terminates properly and is free from deadlocks and unspecified receptions.

Let (M,N) be a safe network, and let v and w be two final nodes in machines M and N respectively. The node pair (v,w) is called an *exit node pair* of (M,N) if, and only if, the state [v,w,E,E] of (M,N) is reachable.

The *exit set* of a safe network (M,N) is the set of all exit node pairs of (M,N).

A safe network (M,N) is called a *phase* if, and only if, every final node in M or N appears in exactly one exit node pair in the exit set of (M,N).

In what follows, we discuss a discipline to connect a number of phases together to construct a multiphase network that is also a phase (thus guaranteeing that its communication terminates properly and is free from deadlocks and unspecified receptions). Phases are connected by joining the exit node pairs of one phase to the initial node pair of another phase, or the same phase. The technique is discussed in detail next.

Let $p_1=(M_1,N_1)$ and $p_2=(M_2,N_2)$ be two phases, with exit sets S_1 and S_2 respectively, and let C be a subset of of S_1. We define a *composite network* of p_1, C, and p_2, denoted by $<p_1,C,p_2>$, to be the network (M,N) where

i. M is the machine constructed (from M_1, C, and M_2) by joining all the final nodes of M_1 in C to the initial node of M_2. The initial node of M_1 becomes the initial node of M.

ii. N is the machine constructed (from N_1, C, and N_2) by joining all the final nodes of N_1 in C to the initial node of N_2. The initial node of N_1 becomes the initial node of N.

The two phases $p_1=(M_1,N_1)$ and $p_2=(M_2,N_2)$ are called the *constituent phases* of the composite network $<p_1,C,p_2>$. In this case, machines M_1 and M_2 are called the *constituent machines* of M, and machines N_1 and N_2 are called the *constituent machines* of N. It is proved in [3] that the composite network $<p_1,C,p_2>$ is also a phase whose exit set is $(S_1 \cup S_2 - C)$.

So far we have discussed how to connect one phase to another. Next, we discuss how to connect a phase to itself.

Let $p_1=(M_1,N_1)$ be a phase whose exit set is S_1, and let C be a subset of S_1. The *composite network* of p_1 and C, denoted $<p_1,C>$, is a network (M,N) where

i. M is the machine constructed (from M_1 and C) by joining all the final nodes of M_1 in C to the initial node of M_1. The initial node of M_1 becomes the initial node of M.

ii. N is the machine constructed (from N_1 and C) by joining all the final nodes of N_1 in C to the initial node of N_1. The initial node of N_1 becomes the initial node of N.

Phase $p_1=(M_1,N_1)$ is called the *constituent phase* of the composite network $<p_1,C>=(M,N)$. In this case, machines M_1 and N_1 are called the *constituent machines* of M and N respectively. It is proved in [3] that the composite network $<p_1,C>$ is also a phase whose exit set is S_1 - C.

4. THE PROSPEC ENVIRONMENT

PROSPEC is constructed in a modular fashion. Each important function of the system is realized by a *tool*. The hierarchy of tools within PROSPEC is shown in Figure 1. The user can invoke each tool independently. PROSPEC also provides a tool called pdtool to facilitate invocation of the other tools. Each *fork* arrow in Figure 1 indicates that a tool can be invoked; a new window for the tool is created upon such invocation. The current version (1.1) of PROSPEC 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 Shell Tool window. Figure 2 shows the menu of pdtool.

2. The protocol editing tool (petool) provides commands for the user to specify the topology of a protocol system. Figure 3 shows the menus of petool.

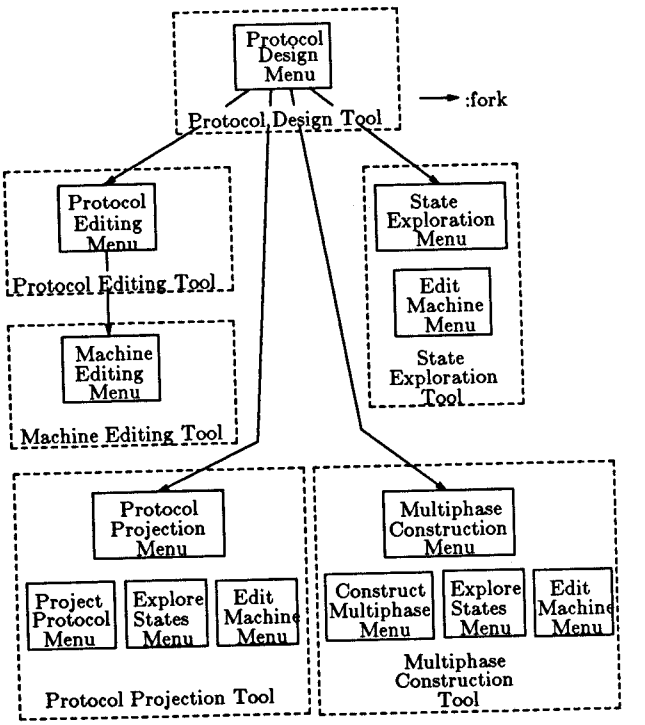

Figure 1. Structure of the PROSPEC System

3. The machine editing tool (metool) is capable of specifying labeled directed graphs of which CFSMs constitute a special case. Figure 4 shows the menus of metool.

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 according to the user's decisions. The tool highlights all problem states in the graph. Figure 4 shows the menus of setool. Note that all functions of the metool are included, 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 [1]. The user can aggregate machine states and messages according to some resolution. The tool constructs the corresponding image protocol. The image protocol can be checked for faithfulness. The reachability graph of the image protocol can be generated within this tool. The machines can also be edited within this tool. Figure 6 shows the menus of pptool. Since the Edit Machine menu is shown in Figure 5, it is not repeated in Figure 6.

6. The multiphase construction tool (mctool) implements the multiphase protocol construction methodology [3]. 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. Figure 7 shows the menus of mctool. Note that the Explore States menu and Edit Machine menu are also included in this tool.

Figure 8 shows two additional menus available in the subwindow at the top of the screen. The Zoom Scale menu allows the user to select different sizes for displaying machines and graphs on the screen.

Figures 9-11 illustrate several typical screen displays during the course of interacting with PROSPEC. Figure 9 shows two communicating FSMs with a partial display of the reachability graph. The machines and the graph can be moved around on the screen. Their sizes can be changed by pulling down the Zoom Scale menu. Most interactions with PROSPEC are done by the user with the SUN workstation's mouse. The menus are hidden until the user holds down one of the buttons on the mouse. Figures 10 and 11 show displays when the mctool was used to construct a multiphase protocol. Figure 10 shows the primary machine of the BSC protocol constructed by connecting phases together [3]; to do so the user needs to see only the initial nodes and exit nodes of individual phases. Figure 11 shows the internal structure of the primary machine after it has been constructed.

A more detailed description of PROSPEC can be found in [7,8]. We anticipate adding some more tools to PROSPEC in the future.

REFERENCES

[1] S. S. Lam and A. U. Shankar, "Protocol Verification via Projections," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 4, July 1984, pp. 325-342.

[2] A. U. Shankar and S. S. Lam, "An HDLC Protocol Specification and Its Verification Using Image Protocols," *ACM Transactions on Computer Systems*, Vol. 1, No. 4, November 1983, pp. 331-368.

[3] C.-H. Chow, M. G. Gouda, and S. S. Lam, "A Discipline for Constructing Multiphase Communication Protocols," *ACM Transactions on Computer Systems*, Vol. 3, No. 4, November 1985, pp. 315-343.

[4] C.-H. Chow, M. G. Gouda, and S. S. Lam, "An Exercise in Constructing Multiphase Communication Protocols," *Proc. ACM SIGCOMM '84 Symposium* (June 1984), ACM, New York, pp. 493-503.

[5] P. Zafiropulo, et al., "Towards Analyzing and Synthesizing Protocols," *IEEE Transactions on Communications*, Vol. COM-28, April 1980, pp. 651-661.

[6] S. Aggarwal and R. P. Kurshan, "Automated Implementation from Formal Specification," *Protocol Specification, Testing, and Verification IV*, Y. Yemini et al. (editors), North-Holland, 1985.

[7] C.-H. Chow and S. S. Lam, "Prospec 1.1: User's Guide," Technical Report, Department of Computer Sciences, University of Texas at Austin (in preparation).

[8] C.-H. Chow, "A Discipline for Verification and Modular Construction of Communication Protocols," Ph.D. Dissertation, Dept. of Computer Sciences, University of Texas at Austin, December 1985.

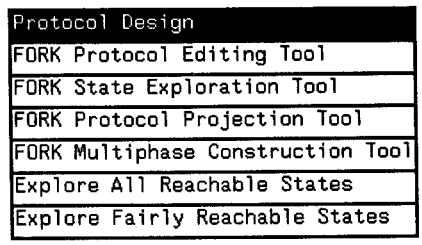

Figure 2. Menu of the Protocol Design Tool

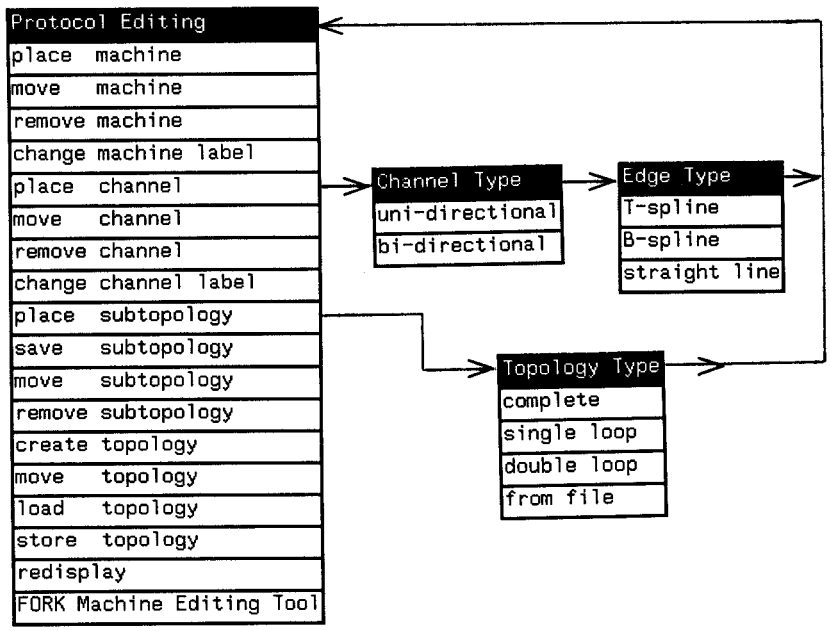

Figure 3. Menus of the Protocol Editing Tool

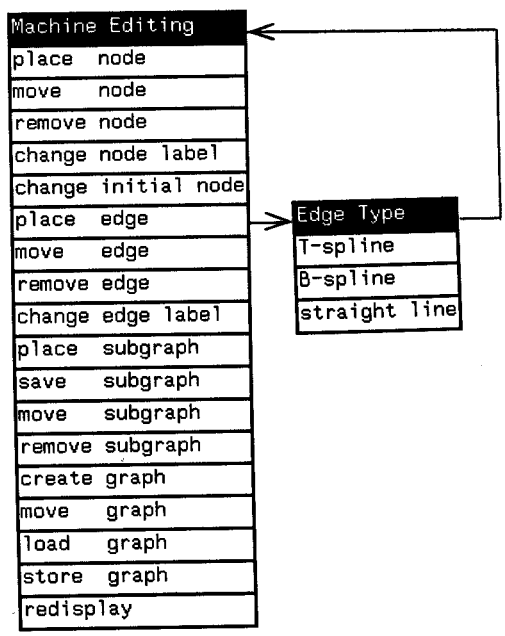

Figure 4. Menus of the Machine Editing Tool

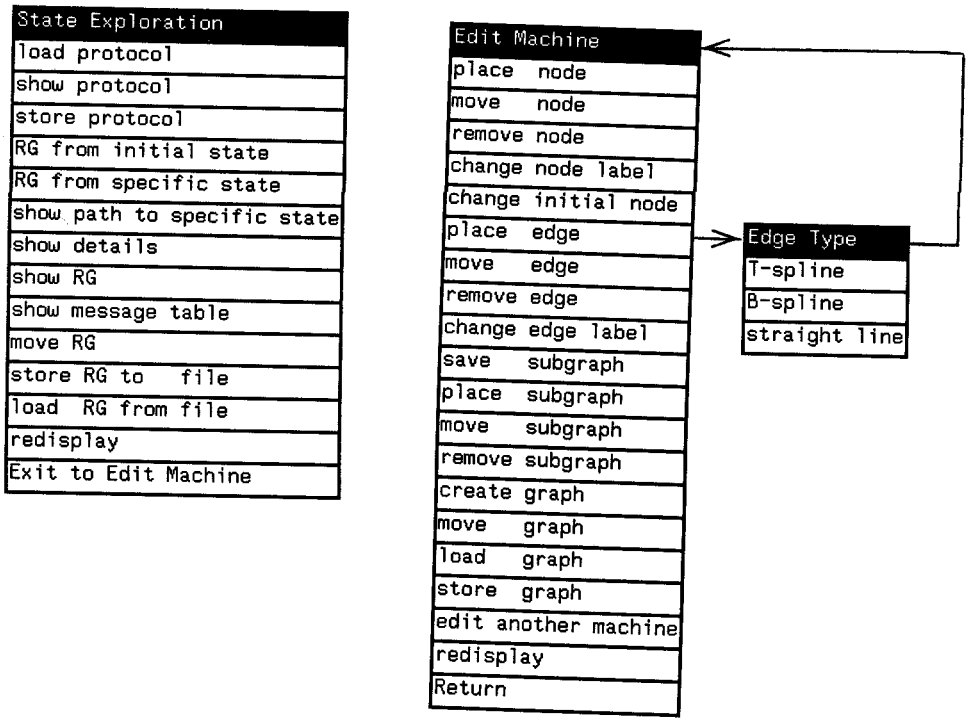

Figure 5. Menus of the State Exploration Tool

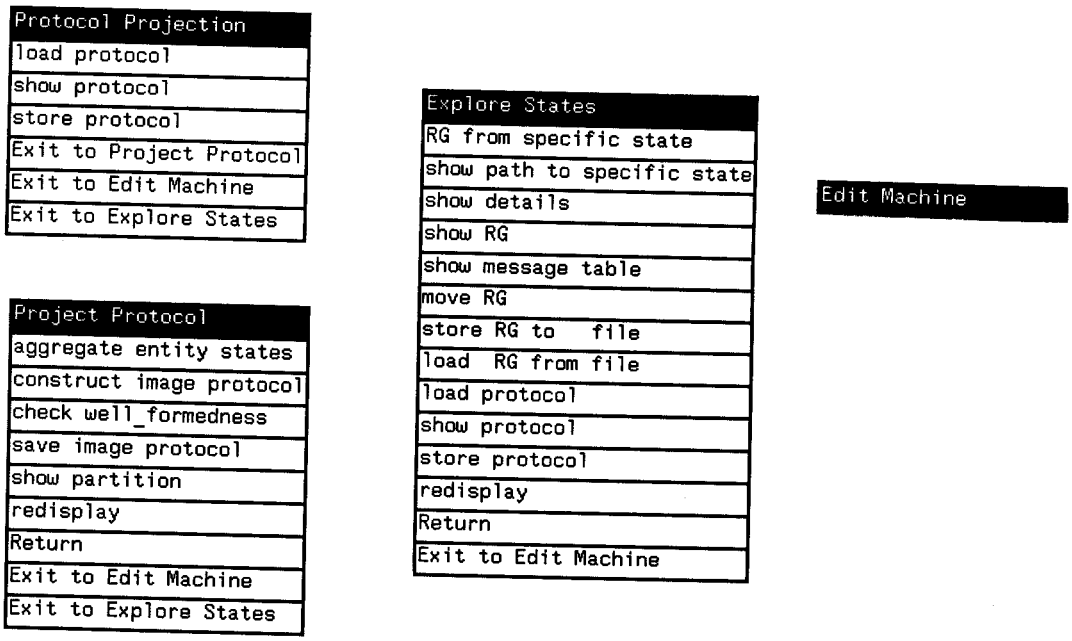

Figure 6. Menus of the Protocol Projection Tool

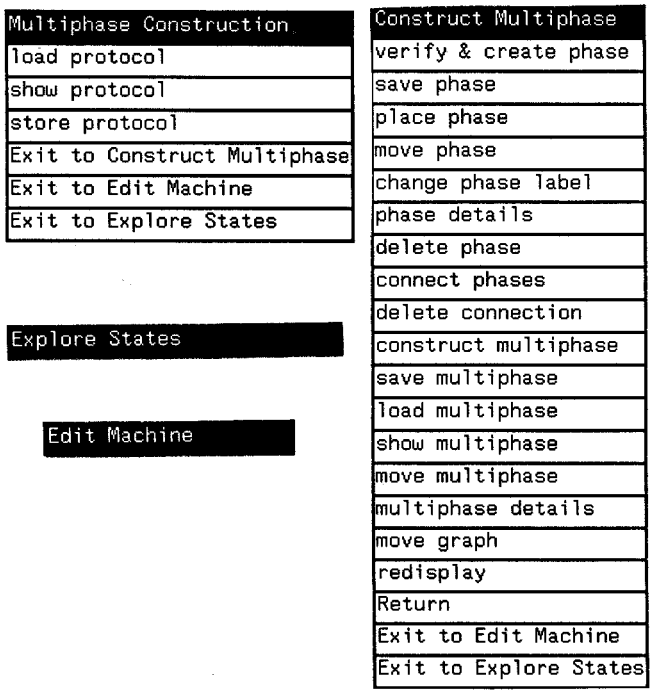

Figure 7. Menus of the Multiphase Construction Tool

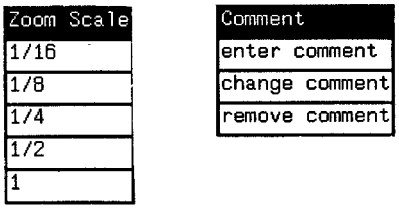

Figure 8. Menus in Subwindow at top of screen

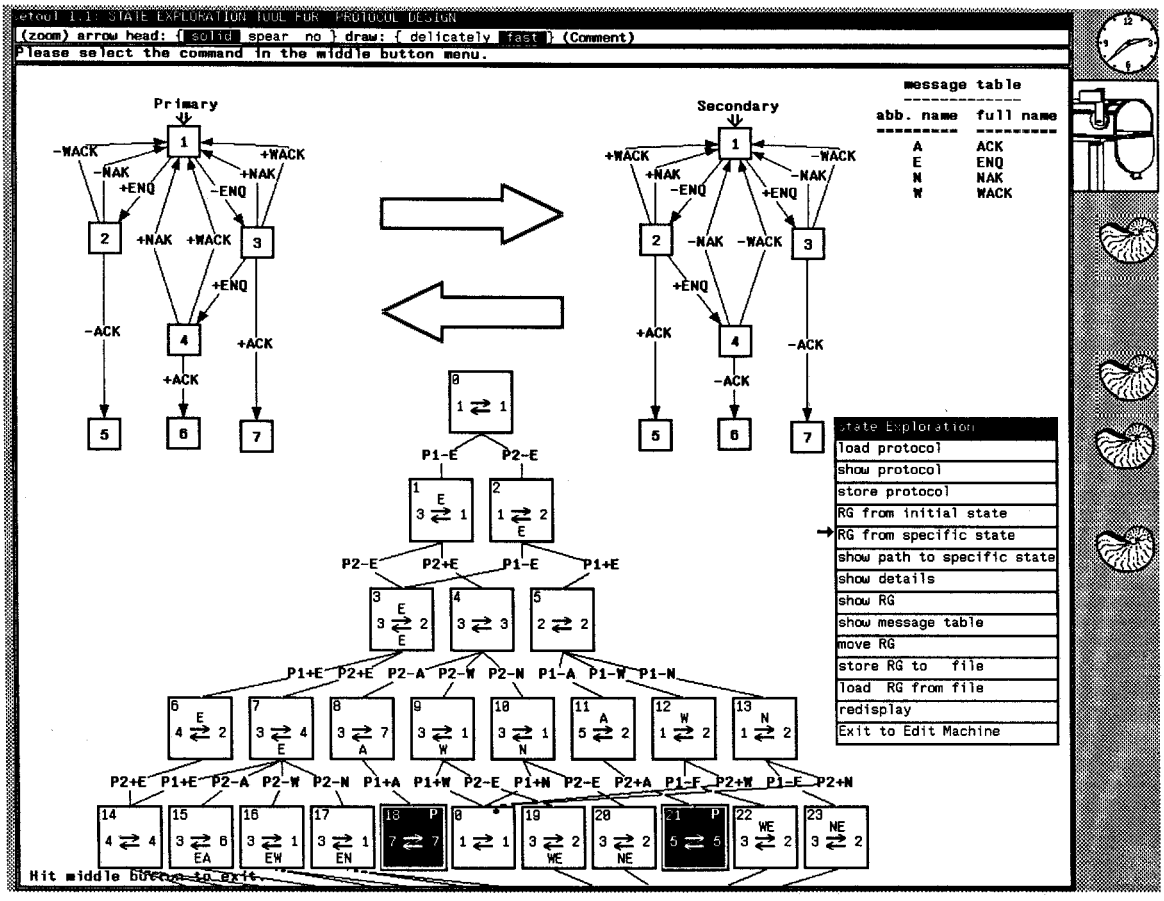

Figure 9. An illustration of the State Exploration tool

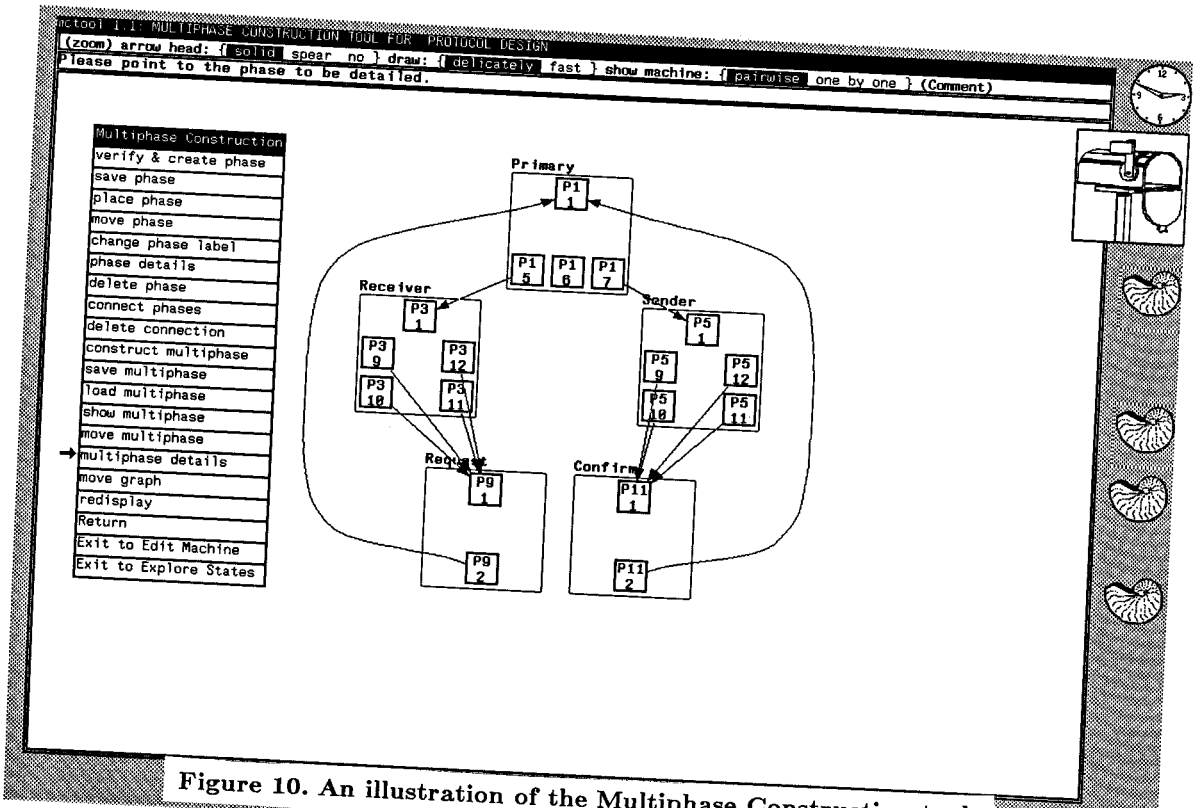

Figure 10. An illustration of the Multiphase Construction tool

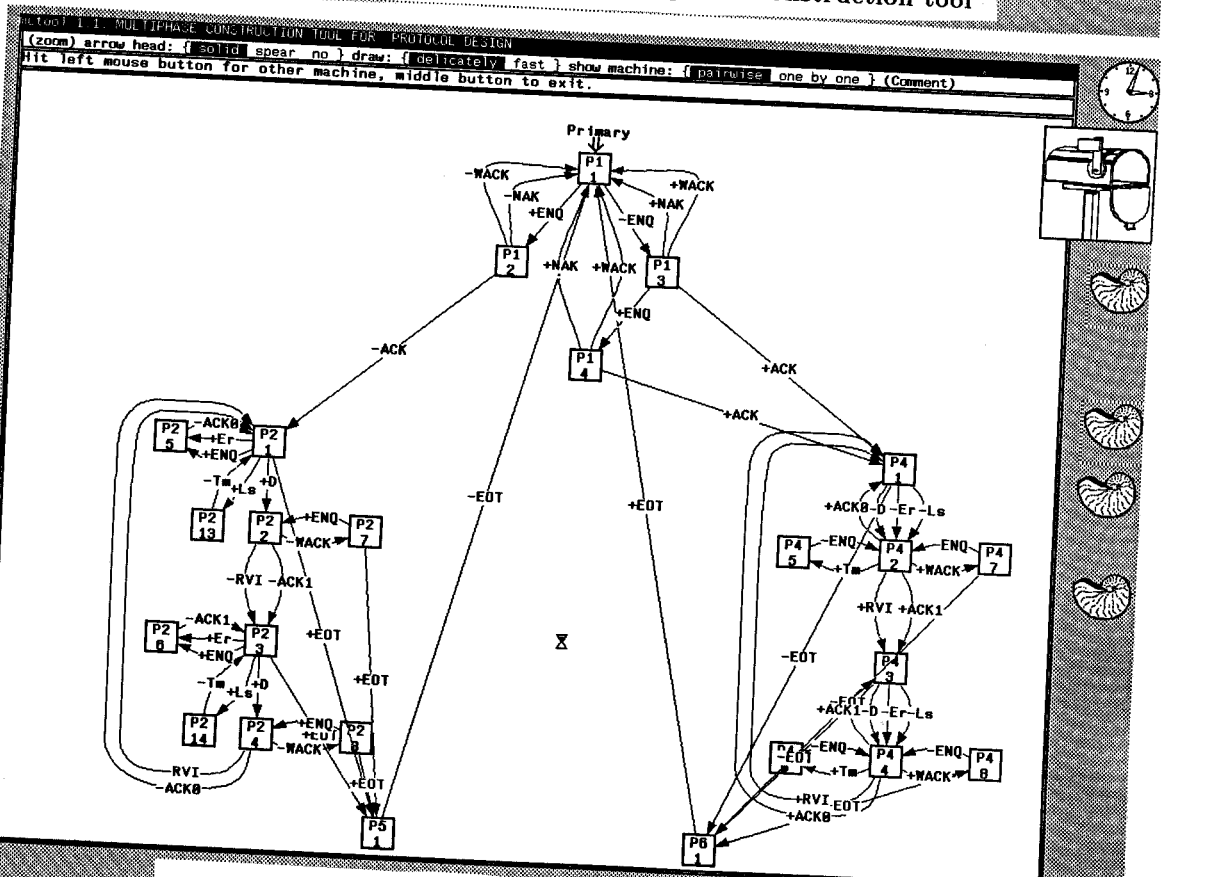

Figure 11. An illustration of the Multiphase Construction tool