マルチメディア通信と分散処理 38 － 4
（１９８８．７．１４）

通信プロトコルの合成のためのソフトウェア環境

張 尭学　　高橋 薫　　白鳥 則郎　　野口 正一

東北大学電気通信研究所

本論文では、開発された通信プロトコルを合成するソフトウェア環境を論じる。本ソフトウェア環境は主に2つのプロセスからなる通信有限状態機械(FSM)で表現するプロトコルの合成をサッポートする。これらのプロセスは2つのFIFO、エラーフリー、有限容量を持つチャネルでつながる。本ソフトウェア環境はプロトコル設計のコストの減少と生産性の向上のために、開発された。そのため、本ソフトウェア環境はプロトコルを合成するとき、設計者にどのように入力するかと適切にアドバイスする。設計者がこれらのアドバイスに従って入力を指定したら、合成されたプロトコルには論理エラーがない。又、本ソフトウェア環境はユーザフレンドリーグラフィックインタフェースを持っている。この対話的なグラフィックインタフェースはラベル付きの有向グラフ表現、カラーとテキスト表現などの機能を持つのでさらに設計者を助ける。本ソフトウェア環境の応用例として、X.25の呼設定フェーズの合成を行う。

A Software Environment for Synthesizing Communication Protocols

Yao-Xue Zhang,　Kaoru Takahashi,　Norio Shiratori,　Shoichi Noguchi

Research Institute of Electrical Communication
TOHOKU UNIVERSITY

Abstract:　A software environment for synthesizing communication protocols has been developed to synthesize a protocol without logical errors. The software environment supports the Communication Finite State Machines (CFSM) model consisting of two processes. The aim of the software environment is to help the protocol designer easily design a protocol without logical errors such as unspecified receptions and state deadlocks, i.e., facilitate the design of protocol. To do this, the system provides the advices on how to give the input for the protocol designer in any current synthesizing stage. These advices protect the protocol designer from creating a protocol containing the logical errors. The software environment contains a user-friendly interactive graphical interface. This graphical interface provides a multiwindow environment and has the functions expressing two kinds of labeled directed graphs, colors and texts on the given windows. Consequently, the protocol designer without experience in protocol synthesis can easily design a protocol without logical errors. As an example, call-establish phase of X.25 has been synthesized by using the developed software environment.

1. INTRODUCTION

Communication protocols are very important in computer networks and distributed systems. The increasing complexity and variety of computer networks and distributed systems have increased the difficulty of designing, analysing and testing protocols. To eliminate errors or inconsistencies of protocols, a number of formal models and verification methods of protocols have been proposed and applied to verification and construction of protocols.

One useful method of representing and verifying protocols is Communicating Finite State Machines (CFSM). This method depicts the processes as being Finite-State Machines (FSM's) that send and receive messages among themselves. One process is described as one finite-state machine. The finite-state machines are connected by communication channels which are unidirectional, error-free, lossless and FIFO.

With the CFSM model, it is most convenient and efficient for a protocol designer to synthesis FSM's graphically. As the synthesized protocols do not contain logical errors (or have the required properties), executable code can be generated directly from the internal representation of the FSM's. Motivated by this objective, several graphical tools for protocol synthesis have been developed, such as the one introduced in [1].

We were motivated to develop the software environment for interactively synthesizing communication protocols by the following objective. When a protocol designer specifies FSM's, some logical errors often occur. These logical errors may be *unspecified receptions*, *state deadlocks*, *overspecifications*, *buffer overflows*, and etc.[1], [3], [5]. The proposed interactive synthesis tools, such as intruduced in [1], have successfully prevented the logical errors unspecified receptions and overspecifications, and can detect the logical errors state deadlocks and buffer overflows. These synthesis tools are used to design and analysis a protocol go hand in hand. The state deadlocks and buffer overflows can not be avoided or prevent in synthesizing protocols. Therefore, if there exists a synthesis environment which not only has a user-friendly graphical interface but also provides the advices or instructions for how to specify the FSM's which lead to avoidance of the logical errors, unspecified receptions, state deadlocks, overspecifications and buffer overflows, the protocol design activities will be enormously facilitated. Our environment for synthesizing communication protocols has been developed for the above objective. The synthesis algorithm embedded in the environment provides its advices for the protocol designer. At the same time, the interactive graphical interface of the environment facilitates the applications of this synthesis algorithm.

Currently, this environment supports only the CFSM model consisting of the two processes. The two communication channels connected the two processes are assumed to be bounded and the transitions of the messages are instantaneous. The protocols produced by the environment should be considered as a communication skeleton. To generate executable code, these protocols need to be augmented with data specification and internal operations. We are now developing a knowledge-based system which will contain this environment.

The paper is constructed as follows. In Section 2, we give the synthesis method for FSM's. In Section 3, we briefly overview our system. In Section 4, we discuss the features and applications of the proposed environment through synthesizing the call-establish phase of X.25. Section 5 is for desirable extensions of the environment and for concluding remarks.

2. INTERACTIVE SYNTHESIS METHOD OF COMMUNICATION PROTOCOLS

In this section, we give the interactive synthesis algorithm which has been implemented in our proposed environment. The outline of the interactive synthesis algorithm is shown in **Fig. 1**. The synthesis algorithm consists of four major components: (1) a set of rules for constructing Global State Transition Graph (called GSTG hereafter), (2) a set of rules for eliminating the logical errors and providing advices, (3) an algorithm for construction of GSTG and (4) an algorithm for production of the two processes, i.e., a protocol, from GSTG. The synthesis algorithm works as follows (See **Fig.1**). The protocol designer interactively inputs the messages to be sent at a new process state and the entry state of an occurring action (we call sending a message or receiving a message as an action). According to inputs from the protocol designer, the synthesis algorithm constructs GSTG by using the rules for constructing GSTG, the rules for eliminating logical errors and providing advices. The advices are interactively given as the feedback to the protocol designer. This GSTG is then decomposed into process P_1 and process P_2. Finally, these process P_1 and process P_2 are given as an output of the synthesis algorithm.

The interactive synthesis algorithm implemented in the proposed environment is a revision of the synthesis method [3] proposed earlier. We omit the detail descriptions of these components here and report them in another paper [1].

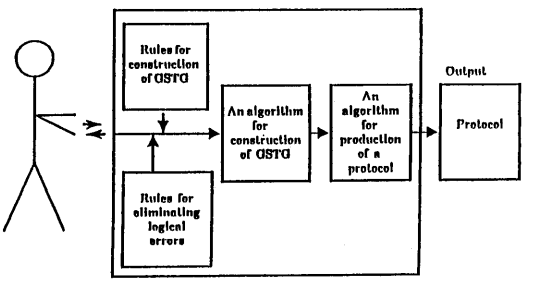

Fig.1 Outline of the interactive synthesis method

3. OVERVIEW OF THE SOFTWARE ENVIRONMENT

This programming environment mainly consists of two modules– a synthesis algorithm and an interactive graphical user interface. The synthesis algorithm implemented in our system has been outlined in Section 2. The functions provided

by the synthesis algorithm are all utilized through the interactive graphical user interface. The software environment has a modular structure. Its construction and the relations of the modules are shown in **Fig.2**.

The multiwindow mechanism and the colors utility of SUN have been used in the developed interactive graphical user interface. Five windows and four kinds of colors have been used. One of the windows, named *GSTG window*, has been used to express the GSTG. Two of the windows, named *protocol window*, have been used to express the two processes. The other two windows named *questionnaire window* have been used to express the system information and the interactions between the system and the protocol designer. The colors have been used to express the advices and the current synthesis status.

A set of functions which constitute the graphical interface translate the internal code into the labeled directed graphs, colors and texts expressed on the five windows.

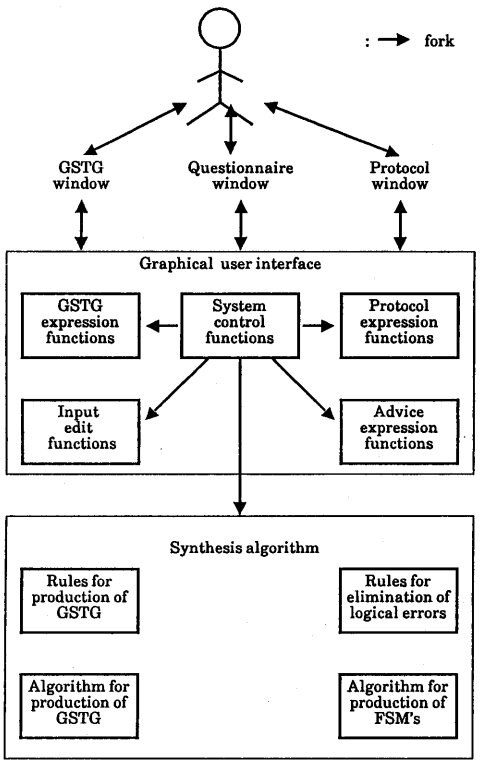

Fig.2 Construction of the software environment for synthesizing communication protocols

In the current version of the programming environment, five groups of functions have been developed:

1) The functions for system control provide thei nformation such as how to use the software environment, and the interface between the protocol designer and the other functions. After starting the suntools window environment, these functions are invoked by typing "sctool" in an shelltool window, and then the programming environment enters working status. A stop button has been used to interrupt or halt synthesis activities.
2) The functions for treating the inputs manipulate the inputs from the protocol designer and express them on the questionnaire window.
3) The functions for showing advices manage the advices from the synthesis algorithm and express these advices by using colors and texts utility of SUN.
4) The functions for expressing processes provide the expressions of the labeled directed graphs. One process has been positioned into one subwindow of the protocol window by these functions.
5) The functions for expressing GSTG locate GSTG to the GSTG window and express it in the labeled directed graph forms.

4. Application of the Software Environent

Our motivation of developing the programming environment is that we want to obtain a support system which not only synthesizes FSM's without logical errors, but also has a user-friendly interactive graphical interface which enormously facilitates the synthesis activities. The user (a protocol designer) has been considered as one who may have not the experience of protocol synthesis. Therefore, it is very helpful to express the labeled directed graphs of processes being produced, the advices for the protocol designer, the proofs of no logical errors and the current synthesizing status for the protocol designer in forms which are easy to be understood. With this idea in mind, we built the graphical interface for the protocol designer to create and manipulate the system informations, the advices, the designer's input, the reachability graphs, i.e., GSTG (which is the proof of no logical errors), and the FSM's represented in the labeled directed graphs. In the following we first describe the features of synthesis provided by the proposed programming environment and then show how to use the system through synthesizing call-establish phase of X.25.

A. Features of using the software environment

Preparation for using the system: It is necessary to informally specify the protocol to be synthesized in the natural language or other description forms. The informal descriptions should include the functions provided by the protocol to be synthesized. Moreover, the messages and states of the protocol should be expressed in integers. The software environment does not accept an input which is not given as an integer except for the protocol's name.

Operations for system information, advice, synthesizing status and input: The system information has been positioned in the subwindow of the questionnaire window (See the left-lower part of **Fig.3**). The system information is changed with progress of synthesis in four phases- initial phase, sending phase, receiving phase and end phase. They can be found in

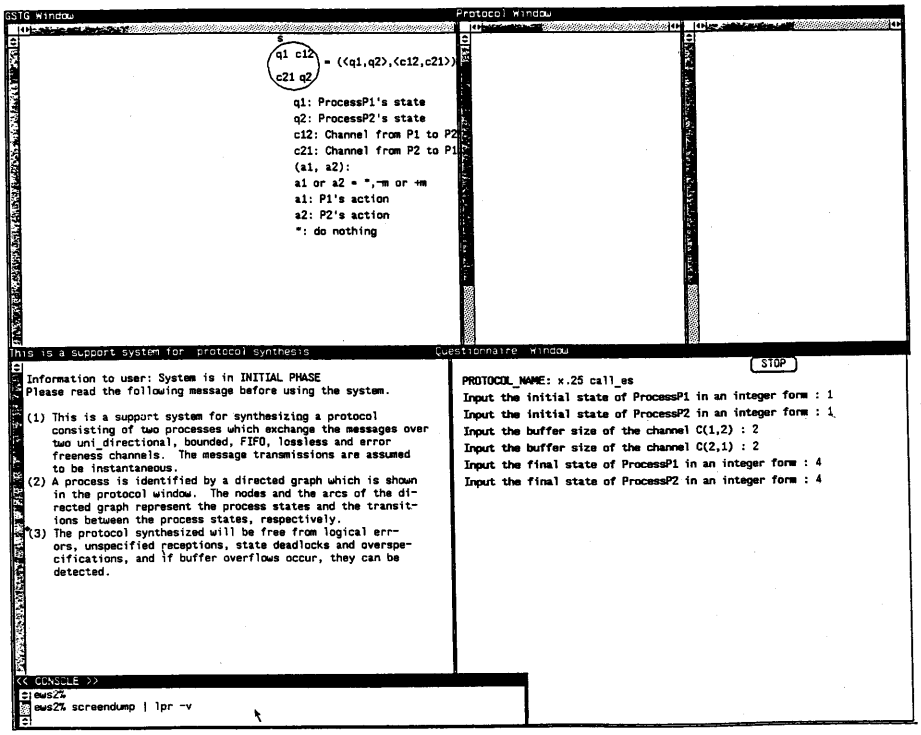

Fig.3 Application to the synthesis of call-establish phase of X.25 - (1)

the left-lower parts of **Fig.3-Fig.8**. In each phase, the information for how to use the system under the current synthesizing stage is shown for the protocol designer. As sending or receiving a message is proceeded repeatedly, the sending phase and the receiving phase are also repeatedly shown to the protocol designer. For a designer who have used the environment, these information may not be necessary.

Advices for the designer are expressed in two forms. One is the color form. The produced process states which can not be assigned as the entry state of an occurring action are all colored red, and the process state where a message must be sent is colored green. The other one is the text form. The produced process states are divided into two groups- the candidates and the rejections of the entry state of an occurring action. These two groups of the states are bath expressed on the questionnaire window in the text form. The advices on how to specify sending a message are also expressed on the same window in the text form. **Fig.4-Fig.7** contain the example of the text expression of the advices which follow the expression of the synthesizing status in the right part of the questionnaire window. However, the hardcopy with colors can not be taken because of the limitation of our printer.

Expression of the current synthesizing status has been treated as the same as the expression of the advices. Progressing global state (or current global state) is colored blue. The number of the progressing global state, the first message in the incoming channel and the current state of each process are all expressed in the questionnaire window in a text form. **Fig.4-Fig.7** contain the example of the text expression of the current synthesizing status which follow the requisitions from the system.

The needed input is asked by the system. The protocol designer answers the questions followed the prepared informal description of the protocol and the advices from the system. For example, **Fig.3** shows the answers to the questions about the protocol name, initial process states, buffer sizes and final process states which follows every colon in the right part of the questionnaire window.

Expressions of labeled directed graphs: Two kinds of labeled directed graphs can be automatically produced when the input has been given. One of both is FSM's, i.e., the processes. Examples about the expressions of the processes are shown in the right-upper parts of **Fig.4-Fig.8**. While the other one is the GSTG. the examples of expressions of the GSTG are shown in the left-upper parts of **Fig.4-Fig.8**. The states, the arcs and the labels of these directed graphs are all positioned by the functions used for expressing processes and GSTG. We found that it is more difficult to position them in the synthesis case than in the analysis case because the position of the progressing global state and the inputs from the designer can not be predicated.

For these two kinds of directed graph, the FSM's are what the protocol designer want to obtain, but the GSTG is the one which only plays in the proofs that there is no logical errors in

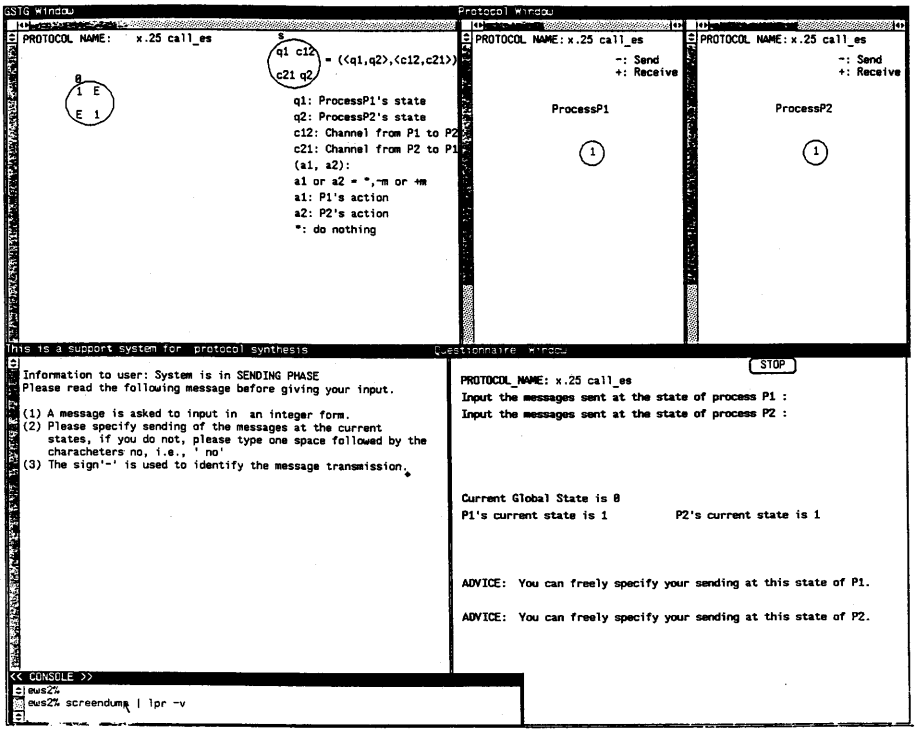

Fig.4 Application to the synthesis of call-establish phase of X.25 - (2)

the synthesized FSM's. Therefore, the protocol designer does not need to care this graph very much. A protocol designer who doubts the correctness of the obtained FSM's may look at the GSTG and get his proofs.

B. Example

Here is an example using the proposed programming environment to synthesize the call-establish phase of X.25[7]. The application of this example will further help the readers understand the software environment.

First, we assume that the informal description of this protocol has been given, and integers and messages have the following relations: 1, call request; 2, incoming call; 3, call connected; and 4, call accepted.

Then, we type "sctool" in a shelltool window of SUN, the computer screen shows the five windows described Section 3.

The system first inquires the protocol name, the initial process states, the final process states and the buffer size of each incoming channel. **Fig.3** displays the screen view just after we gave the inputs. Here, we has answered "1" and "1" as the initial states, "4" and "4" as the final states of process P_1 and process P_2, respectively.

Once the answers have been given, the corresponding nodes are produced in the protocol window and the GSTG window (See **Fig.4**). At the same time, the questionnaire window enters the sending phase, and the synthesis algorithm selects the progressing global state underlying the rule 1 given in Section 2. In **Fig.4**, global state numbered "0" is selected as the progressing global state (current global state). Moreover, the number of the selected global state, the current states and the first messages in the incoming channel of each process, included in the progressing global state, are all displayed on the questionnaire window following the requisitions from the system.

In the sending phase, if a current process state is the new state, then the system inquires the messages to be sent at the current process state. At the same time, the synthesis algorithm provides the advices on how to specify sending messages at the current process states. There two kinds of advices- send messages arbitrarily, or must send messages at the current process states. In **Fig.4**, the system advises that sending messages at the current states of process P_1 and process P_2 can be specified arbitrarily. With the advices from the system, we give the messages to be sent at the current process states according to the functions of the protocol. The messages "1" and "2" are input as the messages sent at the initial states of processes P_1 and P_2, respectively.

When the messages sent at the current process states have been given, the questionnaire window enters the state phase and the protocol designer is asked to assign the entry process states of the occurring actions following the advices from the system. The system provides three kinds of advices- assign the entry state arbitrarily, do not assign the given process states as the entry state or assign a final state as the entry state. In this time, the advices from the system are that the entry states of the two occurring actions can be assigned arbitrarily. Hence, we assign the entry state of the occurring

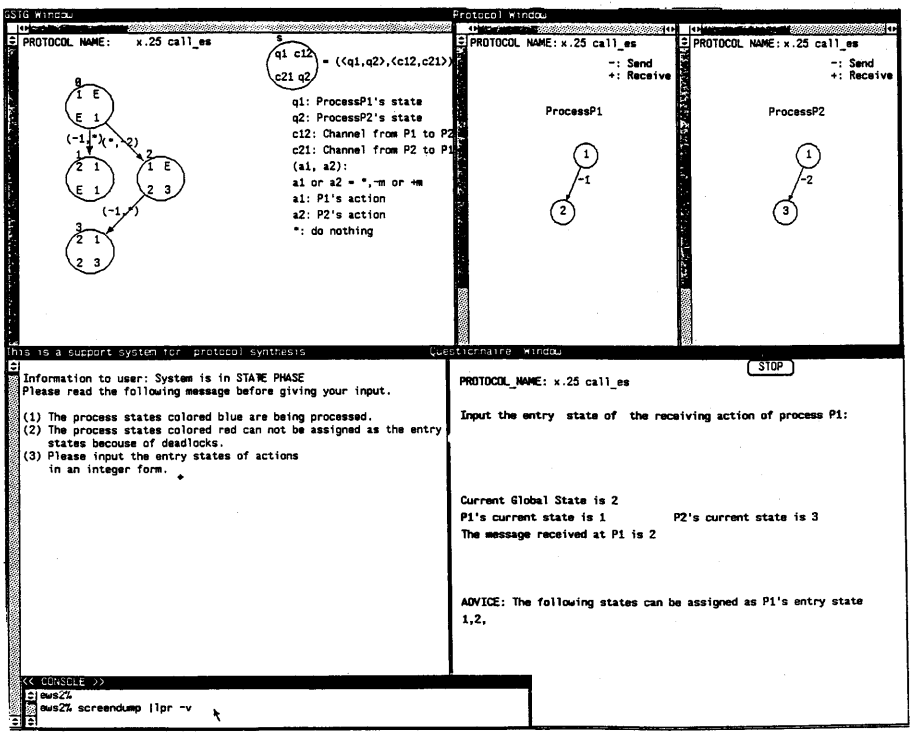

Fig.5 Application to the synthesis of call-establish phase of X.25 - (3)

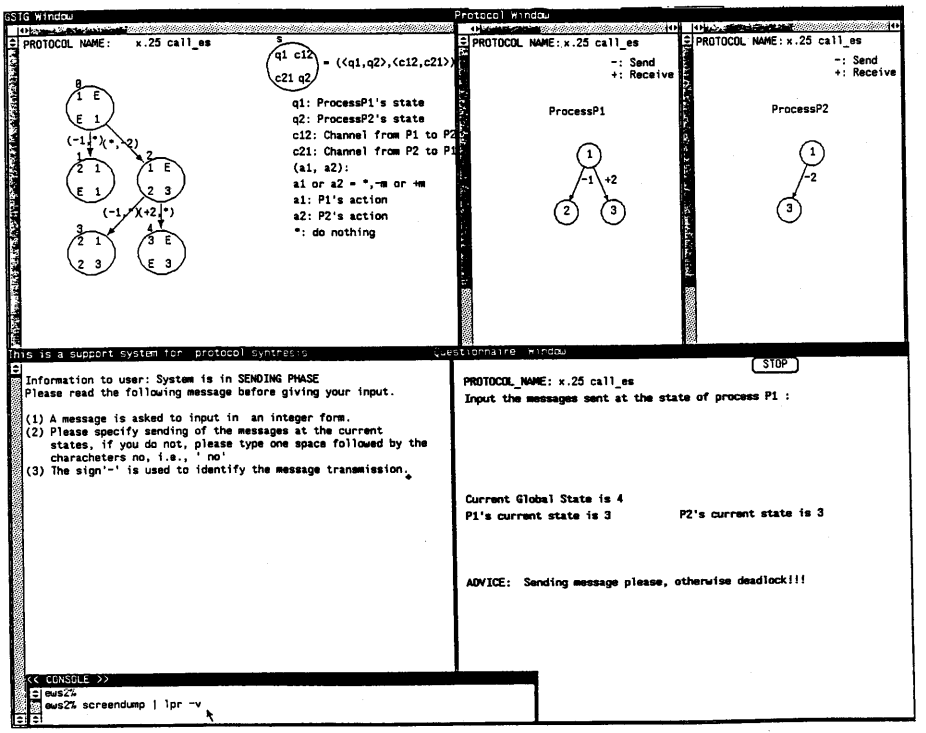

Fig.6 Application to the synthesis of call-establish phase of X.25 - (4)

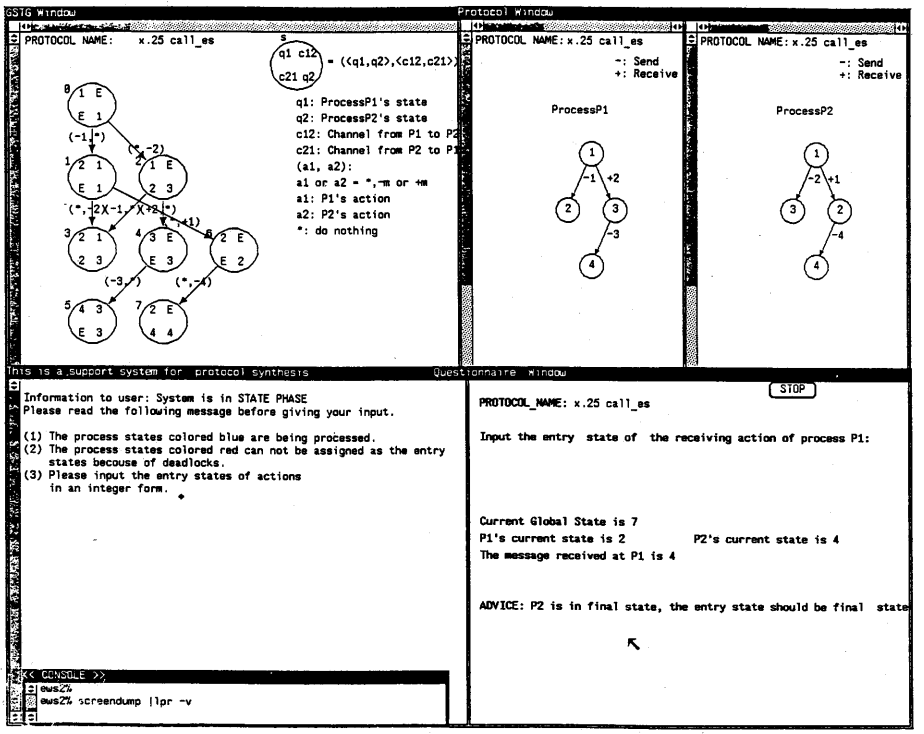

Fig.7 Application to the synthesis of call-establish phase of X.25 - (5)

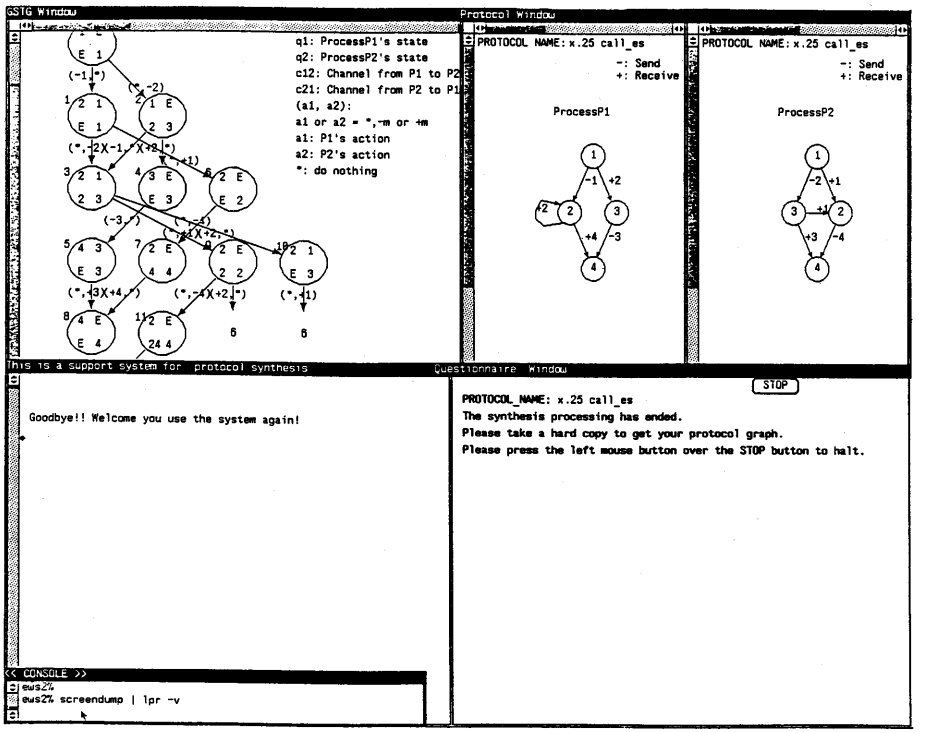

Fig.8 Application to the synthesis of call-establish phase of X.25 - (6)

action of process P_1 as "2" and the entry state of the occurring action of process P_2 as "3".

When the entry states of the occurring actions have been input, the corresponding GSTG and processes are displayed on the GSTG window and the protocol window, respectively. In this time, the two incoming channel are both empty so that no message can be received at the two current process states.

When the activities of sending messages and receiving messages has been finished, the system repeatedly selects the progressing global state, and the questionnaire window returns to the sending phase. The selected progressing global state is one numbered "2" and the current process states of two processes are "1" and "3", respectively. Since the current state "3" of process P_2 is a new state, the system asks us to specify sending messages at the current state "3". However, the advice given by the system tells us that sending messages at this state can be done arbitrarily, so that we type a space followed by "no". To tell the system that we do not specify sending messages at this state. Consequently, the system enters state phase and asks us to input the entry state of the occurring action of process P_1 (See **Fig.5**). As shown in the right part of the questionnaire window of **Fig.5**, the message can be received at state "1" of process P_1 is "2", and the entry state can be assigned as the produced states "1", "2" or any new process state (we assume that in any case a new state is suitable to be assigned as an entry state). Following these advices and the informal description of the protocol, we assign a new state "3" as the entry state. Continuously, the corresponding GSTG and processes are displayed on the GSTG window and the protocol window as shown in the upper part of **Fig.6**. Repeatedly, the synthesis activities progress as above until the conditions for ending the synthesis have been satisfied. An example about the second of advice on how to specify sending a message is given in **Fig.6**, where the system warns that if no message to be sent then deadlocks will occur. Another example about the third kind of advice on how to assign the entry state of an occurring action is given in **Fig.7**. In **Fig.7**, the system advises us that the entry state of the receiving action of process P_1 should be the final state because the process P_2 has reached the final state. Finally, the results of this example are shown in **Fig.8**. In **Fig.8**, the right-upper part displays the synthesized processes, the left-upper part shows the GSTG which is the proof of no logical errors in the synthesized protocol, and the beneath is the end messages for the protocol designer. GSTG window, protocol window and questionnaire window can be individually opened or closed.

5. CONCLUSIONS

We have developed a software environment for protocol synthesis. The purpose of developing this software environment is to increase the productivity of protocol design. Using this software environment, the protocol designer without experience in the protocol synthesis can design protocols without logical errors. The proposed software environment for protocol synthesis has been implemented in SUN 3 workstation running on UNIX.

The advantages of the developed software environment are as follows. First, it automatically provides the advices on how to give the input for the protocol designer. These advices protect the protocol designer from designing a protocol containing the logical errors. Second, the software environment has a user-friendly graphical interface which provides the functions of expressions of two kinds of labeled directed graphs and the interactions between the protocol designer and the system by using the multiwindow mechanism, colors and texts utility of SUN workstation. Accordingly, the developed software environment facilitates the synthesis activities of FSM's.

The further research directions can be summarized as follows:

1) Extend the software environment to treat the problem for synthesizing protocols which consists of more than two processes.

2) Extend the software environment to support the design of protocols which are not modeled in CFSM.

3) Extend the software environment to synthesize protocols without the logical errors which have not been introduced in this paper.

ACKNOWLEDGMENT

The authors are thankful to N. Miyake, U. Fujita and Z.X.Cheng for their great help while implementing this version, and to A. Hamid for his useful discussion, which improved this paper.

REFERENCES

[1] N.Shiratori, Y-X. Zhang, K. Takahashi and S. Noguchi, " A Software Environment for Synthesizing Communication Protocols," in contributing.

[2] P. Zafiropulo et al., "Towards analyzing and synthesizing protocols, " IEEE Trans. Commun., vol. COM-28, no.4, pp. 651-661, Apr. 1980.

[3] Y-X. Zhang, K. Takahashi, N. Shiratori and S. Noguchi, " An interactive protocol synthesis algorithm using global transition graph," IEEE Tans. Software. Eng., vol.14, no. 3,pp. 394-404, Mar. 1988.

[4] ___ ," A knowledge -based system for protocol synthesis (KSPS), " IEEE Journal on Slected Areas in Communi., June 1988.

[5] N.Shiratori et al., " EXPA: Validation method of a communication protocol based on the perturbation analysis", Trans. IPS of Japan, Vol.26, No.3, pp.446-453, 1985 (in Japanese).

[6] N.Shiratori et al., " NESDEL : protocol oriented specification and description language and its applications ", Trans. IPS of Japan, Vol. 26, No.6, pp.1136-1144, Nov. 1985 (in Japanese).

[7] A. Tannenbaum, "Computer networks", Prentice-hall, 1981.