An Adaptive User Navigation Mechanism and its Evaluation

Jeongwon Baeg
Dept. of Electrical Engineering
Waseda University
3-4-1 Okubo, Shinjuku-ku,
Tokyo 169, Japan
baeg@fuka.info.waseda.ac.jp

Atsushi Hirahara
Dept. of Software Engineering
Software Engineering Division, Canon Inc.
890-12 Kashimada, Saiwai-ku,
Kawasaki 211, Japan
hirahara@ccc.canon.co.jp

Yoshiaki Fukazawa
Dept. of Information & Computer Science
Waseda University
3-4-1 Okubo, Shinjuku-ku,
Tokyo 169, Japan
fukazawa@cfi.waseda.ac.jp

Abstract

Recently, there have been many on-line help systems that provide usage explanations for applications. However, it is difficult to find out necessary information in help contents and to understand how to obtain complete information. A navigation mechanism which can provide a user with essential information and guide the user so that he can achieve intended actions is indispensable In this paper, we propose an adaptive navigation system which does not display navigation information on decisions that a user has already made by reflecting the current state of an application. In this research, in order to describe the structure of an application, Petri nets which are suitable for modeling event-driven systems are used. The application system is constructed based on these Petri nets. By communicating between the application system and its navigation system through these Petri nets, user navigation system reflecting the execution state of the application can be realized.

1 Introduction

At present, there are many on-line help systems that provide usage explanations for applications, some of which are based on hypertext. On-line help systems have the following advantages over paper manuals:

- They can be referenced easily during execution of application software.
- Inquired information can be retrieved quickly.

On the other hand, recent on-line help systems also have some problems, such as:

- Both desired and undesired information are included in the explanation when a user requests help on an action.
- It is difficult to obtain the procedure of inquiry for finding complete information on how to use the application in order to carry out intended tasks.

At this point, we tried to review contents of several kinds of on-line help systems, and as a result we found out they could be subdivided as follows:

- a summary on an inquired function with its method of designation,
- explanations on optional functions with their methods of designation,
- designation of pre-conditions for executing a function in an application.

In this paper, we focus on the third contents in the above. Here, several terms are defined: "Navigation" refers to leading a user so that he can carry out his intended actions rapidly and correctly by referring to pre-conditions to be executed before an action is taken. This is different from the common meaning of navigation which is designation of actions to follow. Explanation for its navigation is called "navigation message." "Navigation system" means the system of displaying these navigation messages.

In this paper, we propose a mechanism for navigation system which can present suitable navigation messages with a user to solve the above problems. Specially we assumed that a user has enough information on the decisions that have already been made for the application. By this assumption, the application's current state can be understood to reflect user's understanding at that time. We built a model of user's understanding based on this assumption. Based on this model, our navigation system can produce navigation messages without unnecessary and redundant contents when a user requests. By considering the application's current state, this system can support adaptive navigation both in the case where users differ and in the case where the individual user's tasks,

0-8186-6960-8/94 $04.00 © 1994 IEEE

needs, and knowledge change. Adaptive user navigation can make complex systems more usable, present the user with what he needs and wants to see, as well as speed up and simplify interactions.

Our main target is to realize an adaptive navigation system for event-driven systems. In this paper, as a typical example of event-driven systems, GUI-based application system is used For developers of GUI-based applications, methods to support the development of GUI applications are required as it is very difficult to develop complex GUI applications.

A GUI-oriented model is useful to effectively analyze, design and implement complex GUI applications. State-transition models [1], Petri net models [2], Message Sequence Chart(MSC) models [3], etc., have been proposed to model various aspects of GUI applications. Among these models, Petri nets have the following superior characteristics:

- describability of event-based asynchronous dialog,
- representability of behavioral features of a system,
- simulation capability.

We have adopted the Petri net model for GUI-based application development in consideration of the above characteristics. In order to describe the components of GUI applications, some extended representations have been added.

Our supporting approach to the application development is different from other GUI builder tools such as Visual Basic and Sun Xview tool. They can not aim to support analysis and design activities at the development of application software, but rather to mainly support implementation activity. However, our method can support the whole process in the application software development including analysis, design and implementation process.

Our system is constructed with the procedure as follows: To start with, Petri nets to represent an application with GUI are described. Next, from these Petri nets, program fragments which notify the navigation system of the state transition from an application system are automatically generated. Finally, the whole application programs are completed by adding the developer's hand coding. The navigation system can produce navigation messages according to the marking of these Petri nets in which application's states are reflected by being notified of their transitions.

This paper focuses on the navigation system can support adaptive navigation mechanism based on GUI-based applications, rather development support.

2 Main Features

2.1 Adaptive Navigation Messages

When a user requests navigation on an action, information on which the user already knew can be regarded as unnecessary and redundant explanation. Our navigation system can present explanation to fit the application's current state and to be excluded redundancy with the user.

For example, let us assume that for the playback function of a sound application system, "file selection" and "pressing a button" are necessary pre-conditions in order for "play" function to be executed. When a user inquires navigation about "play" function after only carrying out "file selection," our navigation system adaptively presents only the navigation message about "pressing a button" with the user. In this situation, navigation message on "file selection" is considered to be redundant

2.2 Approach of Reflecting Application States

In order to realize our adaptive navigation system, it is necessary that application's current states is always notified of the navigation system. By this notification, navigation system can vary navigation messages based on the application's current state.

For this purpose, at the implementation time, a mechanism to detect state transitions of an application is incorporated in the application program. Navigation system has Petri nets whose markings reflect the application's state at all times. Whenever the state transition of the application is occurred, the markings of Petri nets in the navigation system is altered and according to those markings navigation messages are generated.

2.3 Support of GUI Application Representation

Several Petri net representations to properly describe GUI applications are specially introduced in this paper. Generally many kinds of windows, menus, and buttons are used in GUI applications, and a suitable model is needed to describe those dialog components in the development process. Using our extended representations of Petri nets, some characteristics and constraints which GUI components have can be described easily. For example, after executing a function, i.e. being selected a menu or a button, its results may

have multiple branches. Our extended representation can be described for that case.

2.4 Support of GUI Application Development

The whole process in application software development by using Petri net modeling can be supported by our method.

At the analysis time, the structure of an application including its states and functions is represented with Petri nets for describing end-user's requirements. The dynamic behaviors of an application based on these Petri nets can be simulated to clarify any relevant problems in the analysis phase.

Iterative refinement of already defined Petri nets is carried on at the design time. The results of the design are verified through continuing simulation.

Based on Petri nets obtained from the design process, the skeleton of program is automatically generated at the implementation time. In addition to this generated program, the application functions are written and the whole application program is completed. Automatic generation of program fragments of an application from Petri nets contributes to enhancement of program productivity and a decrease in its development costs.

3 System Structure with Petri Net Modeling

3.1 Petri Net Notation

Components of Petri nets are represented as follows:

- *Places (conditions) represent states of an application.*

  A place is defined as an application's state. Input places are preconditions required to execute a function, and output places are postconditions which describe the states resulting from the execution of the function.

- *Transitions (actions) represent functions of an application or its interactive components.*

  A transition is defined as an application's function which is fired (executed) if its pre-conditions are satisfied. Events issued by the user are also represented as transitions. For example, "pressing a button" is an event.

- *An arc between a place and a transition defines cause and effect relation.*

- *Markings represent application's states at a time.*

  Markings are changed by actions executed in the application.

Figure 1 illustrates a Petri net a part of the recording function of a sound application system. In this example, "**Record**" function is executed after "**Set file name**" and "**Set record time**" functions being executed. By the marking of this Petri net the application's current state is "*Record panel is pop-uped*" and "*File name is set.*"

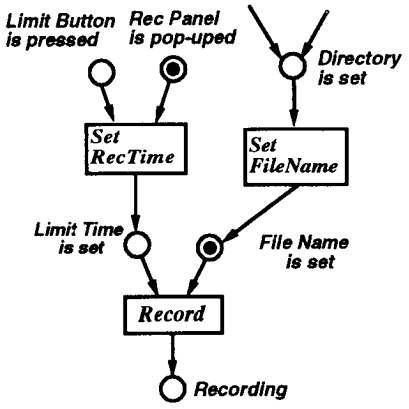


Figure 1: Example of a Petri net

3.2 Navigation Message Derivation Mechanism

The markings of the Petri nets in our navigation system reflect the application's state at all times. Whenever the transition of application's states is occurred by a user, the marking of the corresponding Petri net is altered. When a user requests navigation, the navigation system selects the appropriate navigation messages from the current markings of Petri nets and presents them with the user.

For example, let us assume that a user wants to read navigation messages for function "**Record** " in Figure 1. This navigation system checks whether the input places of function "**Record**", i.e. "*Limit time is set*" and "*File name is set*", have a token or not. A token in place "*File name is set*" means that function "**Set file name**" has already executed. Function "**Set record time**" has not yet executed because place "*Limit time is set*" has no token. Therefore, only the

message "change the state of "*Limit time is set* " using function "**Set record time**" is presented with the user. In this situation, a navigation message for function "**Set file name**" is considered to be redundant and is not presented. If the user doesn't know how to execute function "**Set record time**" the same process is repeated for investigating the previous states. Thus, our system displays only necessary information by selecting arcs which should be traced back from the current state.

In the case where "**File name is set**" has no token in Figure 1, two paths from function "**Record** " can be traced. The user can trace back through either arc, and when enough information has been obtained, can return to the branch point and select another arc by requesting navigation on function "**Record** "again.

3.3 System Structure

The structure for realizing our proposal is illustrated in Figure 2. The upper part of Figure 2 consists of two systems: the GUI-based application and the navigation system which we suppose. The lower part illustrates the Petri net editor.

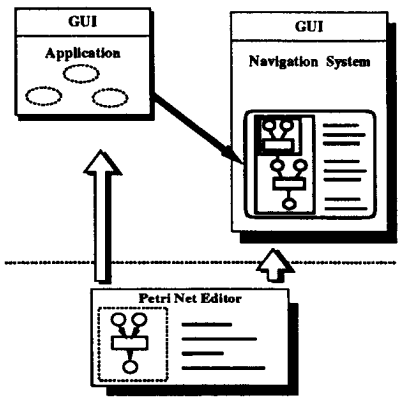


Figure 2: Software structure

Interactive GUI components, application functions and application states are represented in Petri nets. State transitions of the application system by user's actions are informed of the navigation system. Navigation system generates requested navigation messages in accordance with the marking of Petri nets modified by an application system.

This navigation system has a graphical user interface independent of that of the application system. The interface includes a browser on which a user can

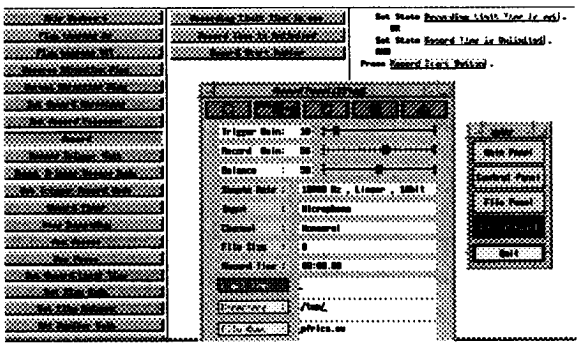


Figure 3: GUI of navigation system

designate function names and a window in which navigation messages are displayed. The GUI of the navigation system is illustrated in Figure 3. The left hand side of this figure displays the browser. Two windows in the middle of this figure illustrate GUI of a sound application system. The right hand side of this figure displays navigation messages presented according to user's requests.

4 Application Development Support

4.1 Extended Representations of Petri Nets

In addition to Petri nets described in section 3.1, representations of Petri nets are extended in our system. These extensions contain frequently appearing patterns found in many GUI applications, and can decrease the cost required to construct Petri nets.

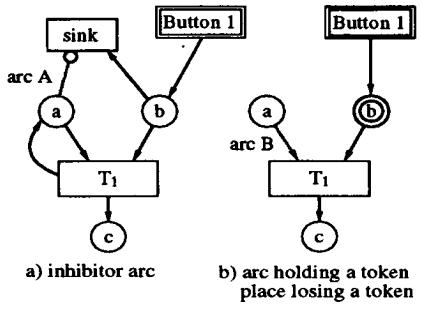


Figure 4: Extended arcs and places

(1) *Inhibitor arc inhibiting a transition's firing.*

When a user presses a button to to open a pop-up window, if the pop-up window has already been opened, button pressing action has to be canceled. Also, before pop-down action is designated, the corresponding window must be pop-uped, i.e. the state of being pop-uped disables button pressing action and enables pop-down action.

There are many such cases where one state enables the firing of one transition while disabling the firing of another one. Therefore, we introduce an inhibitor arc which disables firing when an input place has a token.

"Arc **A**" in Figure 4a) is an inhibitor arc. When place "**a**" in Figure 4a) has a token, any token cannot be moved through "arc **A**." As a result, a token in place "**a**" disables firing of "**sink**" and enables firing of "$\mathbf{T_1}$." Zero test (a test of whether a place has a token or not) can be achieved using this inhibitor arc[5][6].

(2) *Arcs through which tokens are not moved despite of the firing of a transition.*

Several GUI components hold their own states. In the case where transitions are fired from these states, it is often convenient not to move tokens from an input place. For example, the state of a toggle button is kept, although a transition is fired. An arc is introduced through which tokens of each input place are not moved despite the firing of a transition("arc **B**" in Figure 4 b)). By using this arc, the arc from "$\mathbf{T_1}$" of Figure 4 a) to place "**a**" can be omitted.

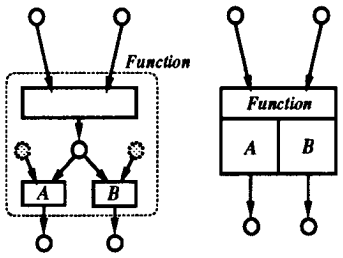
(3) *A place removing a token in the case where transitions can not be fired.*

Many GUI components such as buttons are always available to the user. However, some GUI components are available only when other conditions are satisfied, e.g. a save button of an editor is accessible only in editing state.

In order to represent such GUI components naturally, a place from which a token can be removed in the case where no transition is to be fired is introduced (place "**b**" in Figure 4 b)). By adopting this place, a transition "**sink**" with its two input arcs, as shown in Figure 4a), can be omitted.

(4) *State transition with multiple outputs.*

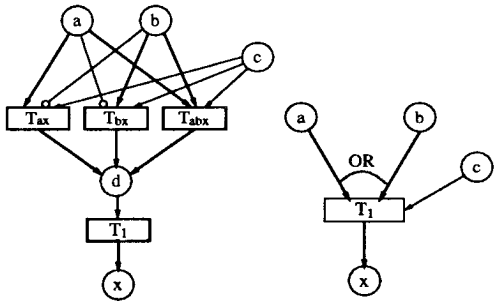
Generally, the state transition caused by a procedure call is not uniquely determined. As an example, in a file opening procedure, the function's output state consists of two possible alternatives: "The file has successfully opened." and "Opening the file has failed." Extended representation is introduced to describe branches of outputs in a transition as shown in Figure 5. This extended representation is available because specifying in detain all possible state makes the description of Petri nets be complicated. Also this representation would be especially suitable for representing the branches of a pull-down menu resulted from a user's selection.



a) former representation b) extended representation

Figure 5: Extended Petri nets for multiple outputs

(5) *OR Representation.*



a) former representation b) extended representation

Figure 6: Representation of OR

In GUIs, there are various kinds of operations used to execute a function. Short-cut keys and confirmation function at the file selection (pressing the OK button or double clicking) are their good examples. Generally, the firing condition for a transition is AND type, i.e. all input places must have tokens (except for an inhibitor arc, whose input place has no token). In Figure 6 a), transition "$\mathbf{T_1}$" is enabled if either or both of

places "**a**" and "**b**" have a token and place "**c**" gains a token. In this situation, the description of normal OR type condition becomes complicated. Figure 6 b) shows simplified description of OR type condition.

4.2 Petri Net Editor

A graphical editor is provided to construct Petri nets, as shown in Figure 7. A developer edits navigation messages and Petri nets to represent an application using this editor. This editor was developed using X Window System Ver. X11 R5.

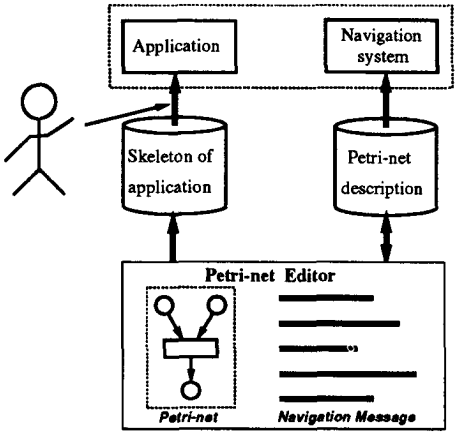


Figure 7: Output of the Petri net editor

This editor has several functions:

- graphical editing of Petri nets,
- editing of navigation messages,
- simulation of constructed Petri nets,
- automatic generation of application program fragments,
- generation of Petri net descriptions used by the navigation system.

Figure 7 illustrates the output of this Petri net editor and the flow of information between the various parts. The automatic generation for application program fragments and control mechanism of application programs are in detail described in section 5.

As an example, our method was applied to a sound application system. Figure 8 illustrates the Petri nets representing the playback function. These Petri nets were constructed using this Petri net editor. An example of the Petri net editor's GUI is illustrated in Figure 9.

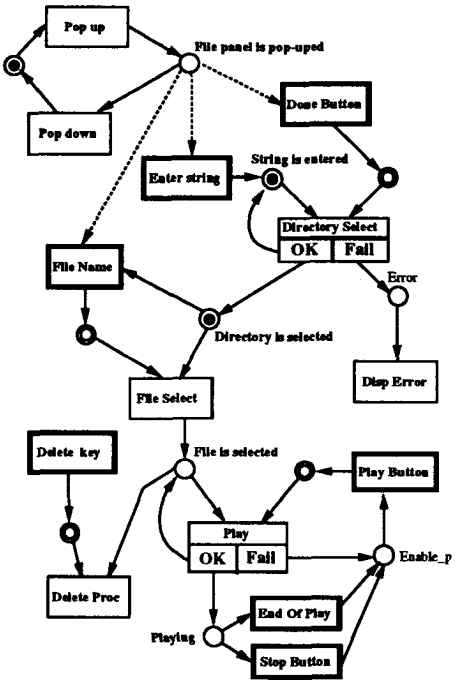


Figure 8: Petri nets for the playback function.

4.2.1 Stepwise Refinement of Petri Nets

When application systems are described with Petri nets, it is sufficient to determine the necessary input places (preconditions) and output places (post-conditions) for each function(transition). However, it is very difficult to describe a large-scale system with one Petri net graph. Therefore, a top-down technique, similar to the concept of stepwise program development, is applied in order to decompose Petri nets from high-level descriptions into more detailed structures.

This Petri net editor makes possible hierarchical editing to refine Petri nets, as shown in Figure 10. Coarse structures of a system are first described as high-level nets, and next a series of gradual refinement steps is followed to describe lower level of nets while both the inputs and outputs of higher level of nets are consistent with those of lower level.

Figure 10 illustrates an example of the stepwise refinement process for Petri nets. While Figure 10 a) represents only the overall "**Load**" function, in Figure 10 b) this function is refined so that the "**Confirm**" function is also realized. The input and output places of b) are made to be consistent with those of a), as shown this figure.

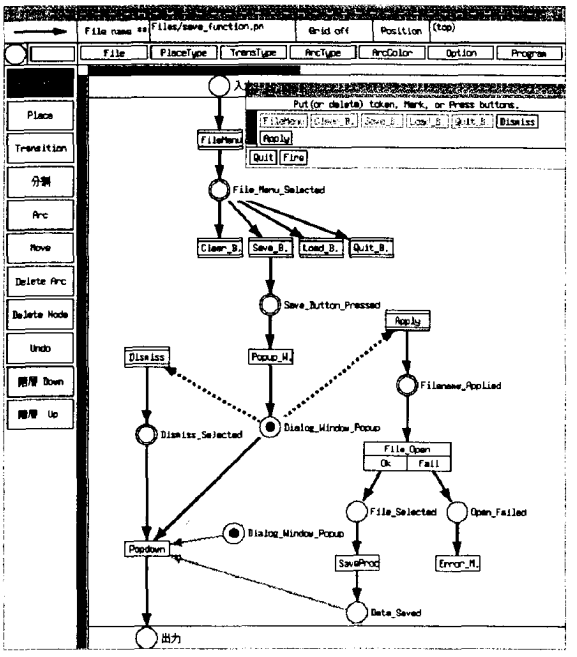


Figure 9: GUI of the Petri net editor

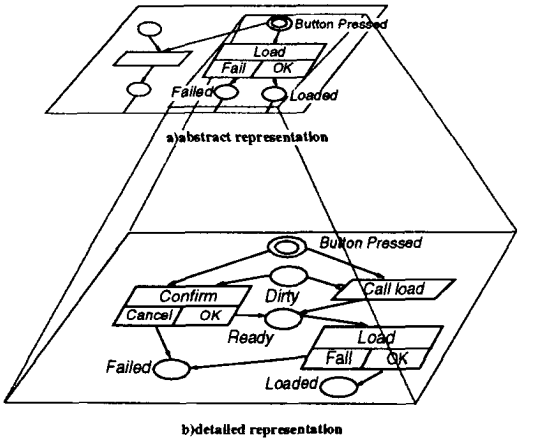


Figure 10: Hierarchy of Petri nets

5 Structure of an Application

5.1 Control Mechanism in Application Programs

By checking pre-conditions before firing and post-conditions after firing of Petri nets, some control mechanism of an application can be incorporated at the implementation time. For that support, its program fragments can be automatically generated.

Figure 11 illustrates the internal structure of an application constructed by our method. Each procedure, "proc" in Figure 11, corresponds to each transition of a Petri net. Program fragments which cause the state to change can be automatically generated in each procedure.

In this figure, the parts drawn with dotted lines are application bodies to be completed by hand coding.

The State Manger (Petri net engine) controls procedures to be called subsequently by checking whether each input place of a transition has a token or not. It is generated automatically from the given Petri net.

5.2 Automatic Generation of Procedures

Figure 12 illustrates an example of automatically generated program fragments. Functions such as "ChangeState(...)" in Figure 12 are state transition procedures. It is not necessary for developers to write program fragments concerning the state transitions. This makes developers free to write complicated control codes. Developers write the application's body in "*Manufacturing part.*"

5.3 Automatic Generation of State Manager

When each input place of a transition gains a to-

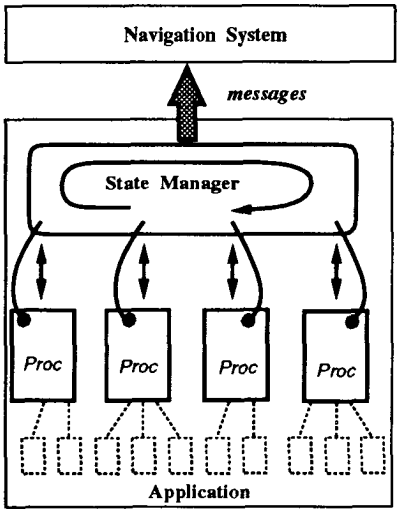
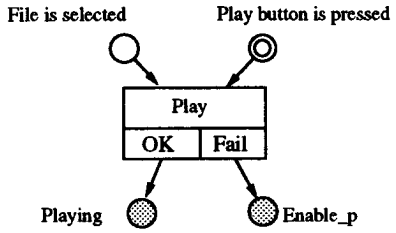


Figure 11: Structure of an application and its automatically generated parts.



a) Petri Net Graph

```
Play()
{
    int state;
    /* Manufacturing part */
    if(state==OK){      /* State OK */
      ChangeState(Playing);
    }else{              /* State Fail */
      ChangeState(Enable_p);
    }
}
```

b) generated procedure

Figure 12: A Petri net and its generated code

```
if (State(File_is_loaded) && State(Play_Button)){
    Trans_SET[trans_num++]=Set_Play_proc;
}
if (State(Play_Break) || State(Play_Stop_Button)){
    Trans_SET[trans_num++]=Play_Break_proc;
}
if (State(Play_Open_Fail)){
    Trans_SET[trans_num++]=Play_Open_Error_proc;
}
```

Figure 13: Example of a part of the State Manager

ken, i.e. pre-conditions of each function are satisfied, the transition can be fired, i.e. registered procedure in application program is called. In our system, the State Manager controls the flow of application's processes. The State Manager monitors state transitions and make a transition fire if every input place of that transition has at least one token.

The State Manager includes if-then style expressions that are automatically translated from every input place of each transition, and repeatly monitors these conditions during execution. Figure 13 illustrates an example of automatically generated State Manager.

6 Evaluation

6.1 Evaluation of the Development Activities

To evaluate our system, several examples were used. Our system and these test cases were implemented under SunOS 4.1.3 + X11R5 on SUN SPARCstation 10. The navigation system developed for evaluation consists of about 2340 lines of C and X11R5 code. The Petri net editor consists of 7500 lines. *Case A* is the whole sound application system as described in the section 4.2. *Case B* is similar to *case A*, but includes many more complex functions. *Case B* is a new version of a system which had already been developed. This trial is a feasibility study for software re-engineering. *Case C* is a Petri net editor developed as a part of our project. In *case D*, the Xarchie tool which is an X11 browser interface to the Archie Internet Information System has been described. These cases were developed by a different individual.

Table 1: Effectiveness of extended Petri nets.

cases	places	transitions	arcs
case A	74 (17)	57 (12)	142 (11)
case B	130 (67)	202 (99)	423 (73)
case C	125 (17)	93 (34)	223 (38)
case D	63 (19)	46 (11)	117 (20)

Table 2: Scale of trial systems.

cases	total (lines)	generated parts(lines)	
		code skeleton	State Manager
case A	4500	500	300
case B	13000	1600	950
case C	8900	1050	720

Table 1 illustrates the number of places, transitions and arcs for each case. The numbers in parentheses represent extended Petri net features described in section4.1 From these results, extended elements were used for about 30% of the places, about 39% of transitions, and about 16% of arcs. These ratios show that extended features are very effectively used. However, the description style of Petri nets varies with each developer, so it is necessary to provide guidelines for describing Petri nets.

Table 2 illustrates the total number of lines for *case A*, *case B* and *case C* (not including parts of the navigation system), and the line numbers for automatically generated procedures and the State Manager. All were implemented in C, and automatically generated code is included in the totals. The original version of *case B* was expressed in about 11000 lines. After reconstructing *case B*, the application body was reduced to about 10450 lines (13000 lines - 1600 lines - 950 lines). Here, 550 lines in the application body was decreased (11000 lines - 10450 lines). These decreased statements resulted from the fact that the control structure of procedures are included in the State Manager. Execution time by this reconstruction be-

Table 3: Variations of message numbers.

functions	*initial state*	*state A*	*state B*
playback	8	5	1
recording	7	5	3
functions	*initial state*	*state B*	*state C*
next selection	9	3	1
fast-forward	9	3	1

came slightly long, however, there was no problem for practical use of the system

Our method requires some additional cost for the construction of Petri nets. However, several merits such as the achievement of rapid interaction by adaptive navigation, support of development activities from the requirements analysis phase, and the generation of high quality programs are also obtained.

6.2 Evaluation of Adaptive Navigation

To evaluate adaptivity for users, we enumerated the number of navigation messages in several states of some functions such as playback, recording, selection of the next music, and fast-forward of *case B* in section 6.1. The results are illustrated in Table 3. The numbers in Table 3 represent the number of messages displayed by our navigation system. *Initial state* means the state of not being acted on by the user at all. *State A* refers to the time when a directory was selected, *state B* refers to the time when a sound file was selected, and *state C* is the playing state.

The number of messages at the initial state is considered to be equivalent to that of a navigation system. which does not vary messages based on application's current state. The average number of messages in *states A, B* and *C* amounts to about 33% of the messages displayed in the initial state. This reduction proves that our method is very effective.

7 Conclusion

A navigation mechanism which can generate the navigation messages controlled by application's states was described in this paper. The process for constructing GUI-based application and its navigation system using Petri nets was also illustrated. Our method can support the development of user navigation system, as well as user-friendly GUI-based system.

As a future work, we are planning to build a system to navigate more effectively using histories of user's previous actions. More researches should be done on the adaptive help systems.

References

[1] P.D.Wellner, "Statemaster, A UIMS based on Statecharts for Prototyping and Target Implementation," *In Proc. CHI'89*, pp. 177-182, ACM, 1989.

[2] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, 1981.

[3] Working Party X/3, "Draft Recommendation Z.120-Message Sequence Chart(MSC)," CCITT, Mar 1992.

[4] M. Ajmone Marsan, et al, "TOPNET: A Tool Based on Petri Nets for the Simulation of Communication Networks, Tool Description," *In Proc. PNPM '91, also in IEEE Journal on SAC*, Vol. 8, No. 9, pp. 1735-1747, 1990.

[5] K. Jensen and G. Rosenberg (eds.), *High-Level Petri Nets, Theory and Application*, Springer-Verlag, 1991.

[6] W. Reisig, *Petri Nets(An Introduction)*, Springer-Verlag, 1985.