A Development Strategy of User Navigation Systems and GUI Applications

Jeongwon Baeg, Atsushi Hirahara and Yoshiaki Fukazawa

Dept. of Information and Computer Science,
School of Science and Engineering,
Waseda University
3-4-1, Okubo, Shinjuku-ku,
Tokyo 169, Japan
{baeg, hirahara, fukazawa}@fuka.info.waseda.ac.jp

Abstract

GUIs(Graphical User Interfaces) have been developed to make applications easier to use. However, effective methods to support the development process for GUI-based applications are highly required as the applications have become increasingly complex. As the supporting methods, we propose a representation technique to suitably describe the structure of an application using Petri nets. Our technique supports all development activities of GUI-based applications during the analysis, design and implementation stage.

On the other hand, concise explanations are indispensible to easily use complex applications and to help achieve user's intended actions. In this paper, we also present a method which is to navigate how to use applications for a user by changing messages depending on the present state of the application based on Petri nets.

1 Introduction

To provide easily usable application software, the studies on the use in GUIs (Graphical User Interfaces) have been continued. As a result, some methods to assist the development of GUI-based applications such as GUI builder tools and UIMS have been proposed [1][2]. By utilizing these methods, it has become possible to build applications with ever increasingly complex GUIs.

However, these conventional approaches lack the important concept in which software development should be supported all over the life cycle stages.

Hence, we propose a method considering the following aspects:

- Correct understanding of users' requirements: To clarify user's requirements at an early stage, a method is essential which supports interactive participation of the end-users. In this phase, characteristics of GUI applications must be considered, which means the user interacts with the application.
- Natural representation of GUI applications: The structure of the most GUI applications is event-driven. For that reason, design methods which are suitable for modeling event-driven systems are required.
- Verification of the results in the design process: A method to verify the results of the design process for an application is necessary because it is difficult to express dynamic behaviors of complex GUI applications.
- Support of the implementation process: To easily develop reliable programs, it is desirable to automatically produce some parts of programs from the results of the design process.

In order to fulfill the above four needs, we built a model based on Petri nets for the development of GUI-based applications.

It is well-known that a GUI-oriented model is useful to effectively design and implement complex GUI applications. For this reason, state-transition models [3], Petri net models [4][5], Message Sequence Chart(MSC) models [6], etc., have been proposed to model various aspects of GUI applications. Among these models, Petri nets have some superior characteristics: describability of event-based asynchronous

0730-3157/94 $04.00 © 1994 IEEE

dialog, representability of behavioral features of a system, and simulation capability. We have adopted the Petri net model and have added some extended representations in order to conveniently describe GUI applications.

In this paper, methods for representing GUI applications with the Petri net model and automatic program generation technique are mainly described.

To easily use applications with complex GUIs, appropriate explanation and guidance to help achieve user's intended actions are indispensable. A method to navigate how to use GUI applications for a user with Petri nets is also described.

2 Main System Features

2.1 Support of GUI Representation

Several Petri net representations to properly describe GUI applications are specially introduced to our system. 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.

2.2 Support of GUI Development

- *Requirements analysis:* In our method, at first, the structure of an application is represented with Petri nets for describing end-user's requirements. Then the dynamic behaviors of the Petri nets can be simulated.
- *Design:* Iterative refinement of already defined Petri nets is carried on in this phase. Also some design information is added to considering implementation of applications. Futhermore, the results in this phase are verified through continuing simulation.
- *Implementation:* Based on Petri nets obtained from the design process, the skeleton of program is automatically generated during this activity. Then the application functions in addition to the generated program are written and whole application programs are 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 Modeling with Petri Nets

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 the input conditions are satisfied. Events issued by the user are represented as transitions. For example, "pressing a button" is an event.
- *Markings represent the application states at a time.*

  Markings are changed by user's actions carried out in the application.

3.2 Extended Representations of Petri Nets

In our system, the representation capability of Petri nets are extended beyond those described in section 3.1. These extensions contain frequently appearing patterns found in many GUI applications, and can reduce the cost required to construct Petri nets.

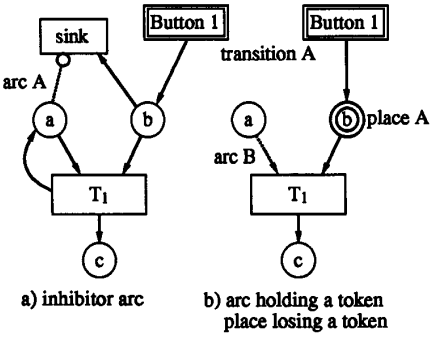


Figure 1: Extended arcs and places

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

When a button is pressed in a GUI function, such as to open a pop-up window, if the pop-up window has already been opened then the button press should be canceled. Also, before a pop-down action is designated, a pop-up window must be displayed. There are many such cases where one state enables the firing of one transition while at the same time disabling the firing of another. Therefore, we introduce an inhibitor arc which disables firing when an input place has a token ("arc **A**" in Figure 1 a)). Although a transition has been fired, a token is not moved through an inhibitor arc. Zero test (a test of whether a place has a token or not) can be achieved using this inhibitor arc[7][8].

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

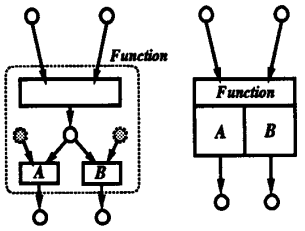
Several GUI components hold their own states. In cases where transitions are fired from these states, it is often convenient not to move tokens from an input place. For example, the state of a toggle button is kept, although a transition is fired. An arc through which tokens of each input place are not moved despite the firing of a transition is therefore introduced ("arc **B**" in Figure 1 b)). By utilizing this arc, the arc from "$\mathbf{T_1}$" of Figure 1 a) to place "**a**" can be omitted.

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

Many kinds of GUI components, like buttons, always enable the user's input. However, some GUI components begin to act only when other conditions are satisfied, e.g. a save button of an editor is valid only during editing. We introduce a place removing a token in the case where transitions can not fire in order to naturally represent such GUI components (place "**b**" in Figure 1 b)). By adopting this place, a transition "**sink**", as shown in Figure 1 a), together with its two input arcs can be omitted.

(4) *State transition with multiple outputs based on conditions.*

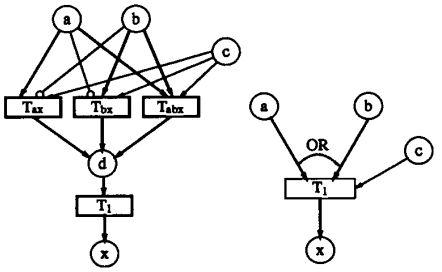
Generally, the state change caused by a procedure call is not uniquely determined. As an example, in a file opening procedure the function's output state consists of two possible alternatives: "The file has successfully opened" and "Opening the file has failed". Our system is extended to represent branches of outputs in a transition as shown in Figure 2. This extended representation is not always necessary if all possible states are specified in detail, however, the resulting Petri nets would become complex. This representation would be suitable for representing the branches of a pull-down menu resulting from a user's selection.



a) former representation b) extended representation

Figure 2: Extended Petri nets for multiple outputs

(5) *OR Representation.*



a) former representation b) extended representation

Figure 3: Representation of OR

In GUIs, there are various kinds of operations used to execute a function. Short-cut keys and confirmation procedures during file selection (pressing the OK button or double clicking) are their good examples. Generally, the firing condition for a transition is an AND type, i.e. all input places must have tokens (except in the case of an inhibitor arc, whose input place has no token). In Figure 3 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 de-

scription of a normal OR type condition becomes complicated. Figure 3 b) shows our simplified description of an OR type condition.

3.3 Stepwise Refinement of Petri Nets

Hierarchical editing to refine Petri nets of a system is possible with our Petri net editor. Coarse structures of a system are first described as high-level nets, and next a series of gradual refinement steps is followed to describe low-level ones while maintaining consistency between the inputs and outputs of high-level ones.

3.4 Petri Net Editor

We developed a graphical editor to construct Petri nets. A developer writes Petri net graphs which represent the structure of an application using this editor.

This editor has several functions:

- graphical editing of Petri nets,
- simulation of constructed Petri nets,
- automatic generation of application program fragments.

An example of Petri nets constructed using the Petri net editor is illustrated in Figure 4. This net represents the record function, one part in all functions of a sound application.

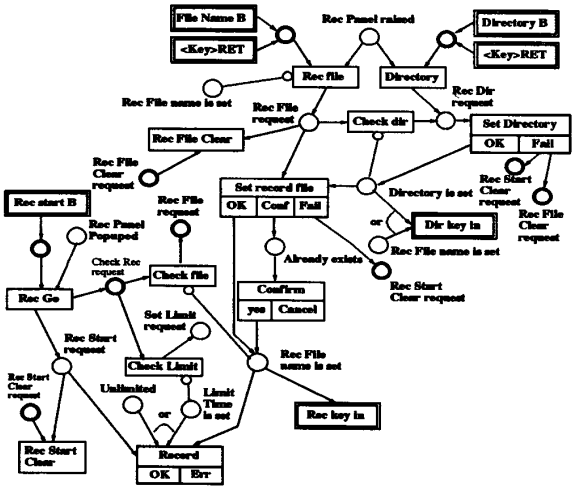


Figure 4: A Petri net for the record function

4 Application Development Support

4.1 Automatic Program Generation

When a procedure(transition) is focused on, states that can be reached from the procedure, i.e. procedures that may execute next can be detected by checking its output places. Therefore, the mechanism which causes the state to change can be generated. Figure 5 illustrates an example of automatically generated program fragments.

Function calls such as "ChangeState(...)" in Figure 5 are state transition procedures. Developers write the application's body in *"Manufacturing part"* Though the developer must write the procedures to be called from *"Manufacturing part"*, he needs not to write procedures concerning the state transitions in these procedures, as these will be generated automatically. In this way, the skeleton of an application can be generated automatically from the output places of a transition.

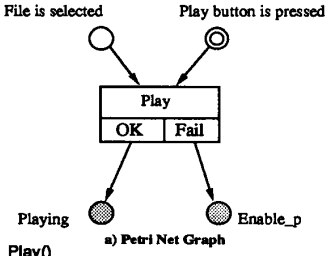


Figure 5: A Petri net and its generated code

4.2 Control of Processing

Transitions can be fired, and registered procedures called, when each input place gains a token, i.e. every input state satisfies their constraints. In our system, the State Manager (Petri net engine) controls the application's process flow. The State Manager monitors state transitions and fires a transition if every input place of that transition has at least one token. Also, called procedures request a state change in order

for the State Manager to transit tokens to the output places for firing the next transition. The State Manager includes IF-THEN style expressions that are automatically translated from every input place of each transition of the given Petri net, and repeatedly evaluates these expressions during execution.

Figure 6 illustrates the internal structure of an application constructed by applying our method. The parts surrounded by solid lines are automatically generated, and the parts drawn with dotted lines are application body components to be completed by hand coding.

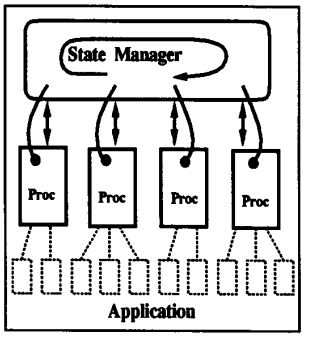


Figure 6: Internal structure of an application

5 Navigation Mechanism

We also propose a navigation mechanism to cope with the problems of recent on-line help systems that display even redundant information which a user has already known.

In this paper, "navigation" means a designation of pre-conditions of the action when a user carries out an action in an application. Specifically, we simply assume that a user has enough knowledge on the actions that have already been carried out. When a user requests navigation, information on functions which have not yet been executed among pre-conditions for executing a function at the present application's state is displayed to the user. This information is called a navigation message.

Navigation message derivation mechanism is explained using an example. Let us assume that a user wants to read navigation messages for function C in Figure 7. Our system checks whether each input places of function C, i.e. "d" and "e", contains a token or not. A token in place "e" means that function B has already executed. In Figure 7, because place "d" has no token, function A has not yet executed. Therefore, the user is given the message "change the state of "d" using function A". In this example, a navigation message for function B is not displayed because it would be redundant in this situation. If the user doesn't know how to operate function A, the same process is repeated for the previous states. Thus, our system displays only the necessary information by selecting arcs which should be traced back from the present state.

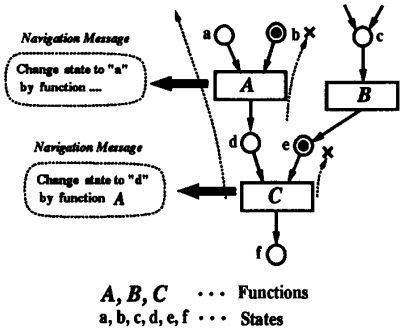


Figure 7: Navigation message derivation mechanism

In order to realize this mechanism, the following approach is taken. To start with, Petri nets representing the structure of an application, including the GUI, are written with our Petri net editor. Next, navigation messages based on these Petri nets are edited with that editor. Navigation system also has the same structure of Petri nets described before. Then, from these Petri nets, our system automatically produces program fragments which notify the navigation system of its application's state transitions, so that the change of an application's state occurred by user's actions is reflected. Our navigation system generates navigation messages according to the markings of these Petri nets.

6 Evaluation

To evaluate our system, several examples were used. Our system and these test cases were implemented under SunOS 4.1.3 + X11R5 on a SUN SPARC station 10. *Case A* is the whole application described in the section 3.4, and *case B* is similar to *case A*, but includes several more complex functions. *Case B* is

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 the 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

the new version of a system which had been already developed, and therefore also represents a feasibility study for software re-engineering. *Case C* is a Petri net editor developed as part of our project. In *case D*, the Xarchie tool which is an X11 browser interface to the Archie Internet Information system has described. Each of these cases was developed by different individuals.

Table 1 illustrates the number of places, transitions and arcs for each case. The numbers in parentheses represent extended Petri net features described in section 3. 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 effective. 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*, and the line numbers for automatically generated procedures and the State Manager. All were implemented in C, and code that was automatically generated is included in the totals. The original version of *case B* was expressed in about 11000 lines. After reconstruction with our system, the application body was reduced to about 10450 lines (13000 lines - 1600 lines - 950 lines) and the number of statements decreased somewhat as code related to process flow were moved to the State Manager. The resulting increase in execution time was minimal and shown no problem for practical use of the system.

Our method introduces some additional cost for the construction of Petri nets. However, several merits such as the support of development activities from the requirements analysis phase, and enhancement of program productivity are also obtained.

7 Conclusion

We have illustrated the process for constructing GUI-based applications using Petri nets.

Our system can support software development activities, via simulation in the Petri net editor, from the requirements analysis phase. However, when these requirements are changed, it is necessary to modify original Petri nets in our system. Also if the skeleton of an automatically generated application program is modified, it is very difficult to modify the corresponding hand-written code. For this purpose, a method will be needed that minimizes parts of the application program that must be modified owing to changes in the requirements.

As a part of our future work, we are planning to build a system to more effectively navigate using histories of user's previous actions. Also, we need more research to generate concise information for other help information except pre-conditions to execute a function.

References

[1] J.C.Armstrong Jr, "Six GUI Builders Face Off," *SunWorld*, December, 1992.

[2] M.Green, "A Survey of Three Dialog Models," *ACM TOG*, Vol.5, No.3, 1986.

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

[4] C.Janssen, A.Weisbecker, J.Ziegler., " Generating User Interface from Data Models and Dialogue Net Specifications," *In Proc. INTERCHI '93*, pp. 418-423, ACM, 1993.

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

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

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

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