Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I
IMECS 2008, 19-21 March, 2008, Hong Kong

Method of Generating Operation Map from Source Programs for Operation Learning

Hajime Iwata * , Daisuke Kuroiwa *, Junko Shirogane †, Yoshiaki Fukazawa *

Abstract— **Many complicated software packages make the operation difficult for end users. To acquire the skills for software operation, supporting end users to learn operating methods by themselves is effective. When in learning operating methods, it is important for end users to check the overview of the operating methods and their current step in the sequence of operations. On the basis of the overview and their current step, end users can consider how to operate the software to achieve their goal. For this purpose, we consider that "operation maps" are effective. Operation maps show the sequences of window switching in the software. Also, in operation maps, the window on currently focused is emphasized. This window can be considered as the current step of an end user. To realize operation maps easily, we propose a method of generating operation maps from the source programs of the software. In this paper, we describe how operation maps can be generated by extracting window information, such as the titles and widget types of windows, from source programs.**

Keywords: Graphical User Interfaces, Learning Support, Operation Maps, Window Switching

1 Introduction

Many types of software have been developed with a wide range of functions. As a result, the operating methods used in software tend to be complicated; thus, it is sometimes difficult for end users to learn how to operate software. Therefore, it is important to provide support for learning operating methods.

There are various support methods for learning operation methods, such as paper manuals, help systems, navigation systems [1] and tutorial systems [2] [3]. These methods are suitable for end users who want to learn the details of each operating step, because these support methods show operating methods step by step. Also, these methods are effective when end users follow the instructions without any errors or mistakes. However, in learning operation methods, it is also important for end users to understand the overview of the entire operation and for them to be supported when they make errors and mistakes. These above-mentioned methods are not suitable for helping end users to understand overviews of operating methods or for helping end users to recover from mistakes and errors.

In these cases, it is important for end users to be able to learn how to operate the software by understanding overviews of the operating methods by finding their current step in the overview and by considering how to achieve their goal. To support these aims, it is effective to show a map of window switchings and to emphasize the end users' current step on the map. In our method, this map is called an "operation map" and the system that shows the operation map and the end users' current step is called an "operation map system".

Using an operation map system, end users can see whole sequences window switching involved in software operations. The operation map system is considered to be effective for supporting end users as they learn operation methods and recover from errors and mistakes. Thus, by considering how to operate the software and how to recover from errors and mistakes, the end users can acquire much greater skill at operating the software.

However, developing operation maps and an operation map system are a heavy burden for software developers, and high costs are required. Thus, we propose a method for generating operation maps and an operation map system automatically. Using our method, it becomes easy for developers to develop operation maps and an operation map system, which are generated from the source programs of the target software. In our method, our target program language is Java and the source programs are translated into JavaML [4] format using a JavaML converter, a tool for translating Java source programs into JavaML format, which is the XML format that Java source programs are translated into. Using the source programs in JavaML format, our system generates operation maps and an operation map system. The operation map system generated by our system is written in AspectJ [5] programs, and is weaved into the target software. AspectJ is one of the tools used to realize aspect-oriented programming (AOP). Using AspectJ, it is pos-

*Department of Information and Computer Science, Waseda University, Okubo 3-4-1, Shinjuku-ku, Tokyo 169-8555, Japan; Tel/Fax: +81-3-5286-3345; Email: {hajime_i,daikuro,fukazawa}@fuka.info.waseda.ac.jp

†Tokyo Woman's Christian University, 2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan; Tel/Fax: +81-3-5382-6763; Email: junko@lab.twcu.ac.jp

sible to weave the operation map system into the target software without modifying the source programs.

2 Features of our method

2.1 Overview of operation learning

Operation maps show the sequences of all windows during the necessary operations in the target software. By consulting an operation map, end users can find the current step they are at in the operation sequence and consider how to operate the software to achieve their goal.

During learning, it is said that it is very important for learners to consider and find the solutions of problems by themselves. In our method, using the operation maps, end users search for the final window that they wish to operate to achieve their purpose. This final window corresponds to the step for achieving end users' goal of the operation maps. Then, end users consider the appropriate path of operations from the current step to the final window by themselves.

Also, when end users make mistakes in the operations, or when errors occur, they can recover from the mistakes or errors by referring to the operation map, which is difficult to do using manuals. In these cases, operation maps can support error recovery.

Details of the operation maps and an operation map system are as follow.

2.2 Operation maps

Operation maps include all window switchings during the operation of the target software. Window sequences of the entire software are represented in operation maps. That is, window maps represent overviews of operations in the target software.

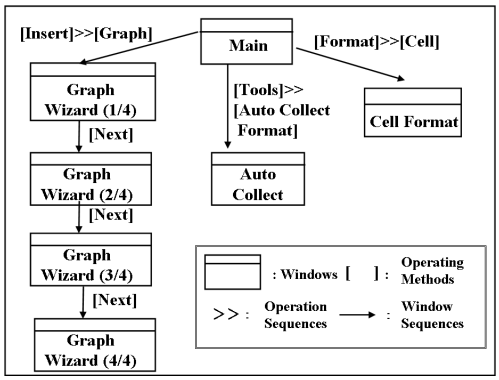

Figure 1: Example of an operation map

An example of an operation map for an operation in Microsoft Excel 2003 is shown in Figure 1. All the windows in the target software and their relationships are represented.

In operation maps, arrows show the sequences of the windows. When window switchings occur, some user events, such as pushing a button and selecting a menu, become triggers. These triggers are shown next to the arrows. The names of windows used as triggers are identified by square brackets. The operation sequences of these triggers are represented by ">>". For example, in Figure 1, [Insert] >> [Graph] means that end users select the menu button "Insert", then they select menu item "Graph", and then Microsoft Excel 2003 switches from the "Main" window to the "Graph Wizard(1/4)" window.

In some types of software including Microsoft Excel 2003, a window is created several times for each operation. For example, Microsoft Excel 2003 creates a window of working spaces for each selection of the menu button "File" followed by "New". In this case, operation maps represent only the window switchings, and these windows are represented as a single window.

2.3 Operation map system

To realize the support of learning operations using operation maps, it is necessary to display the current step of the end user and the operations in the windows. When an end user operates software, the current step of the end user, i.e., the window currently focused on, changes. Thus, it is necessary to acquire the window currently focused on. The system used for this is called the operation map system. In the operation map system, an operation map is displayed, and the window currently focused on is acquired, then displayed and highlighted in the operation map.

By displaying window the currently focused on, end users can recognize the step that they are at in the entire software operation. End users can thus consider their next operation. When end users make mistakes in an operation, or when errors occur, they can consider how to recover from them by referring to the operation map.

3 Preliminary

To extract the sequences of window switching, we classify the types of widgets and generating windows. These classifications comprise the "listener database" and "window database", respectively.

3.1 Listener database

Window switchings often occur in software. For example, the flow for saving contents to a file in a simple text editor is shown in Figure 2 . This text editor was written using Java and Java Swing packages.

When window switching occurs, the widgets that are operated on users occur often become triggers. There are many types of widgets in the Java Swing packages; however, only some widget types are used as triggers. We

classify widgets into those that become triggers and those that do not become triggers. We call the widgets that become triggers "action widgets", and those that do not become triggers "nonaction widgets". Action widgets are widgets that react to user events and become the triggers of window switching, and nonaction widgets are widgets that do not react to user events. For example, button widgets and menu item widgets are action widgets. When user events occur due to their use, then some processes are performed. That is, they cause user events. Label widgets and list box widgets are nonaction widgets. Even if user events occur due to their use, no process is performed in many cases. That is, they do not cause user events.

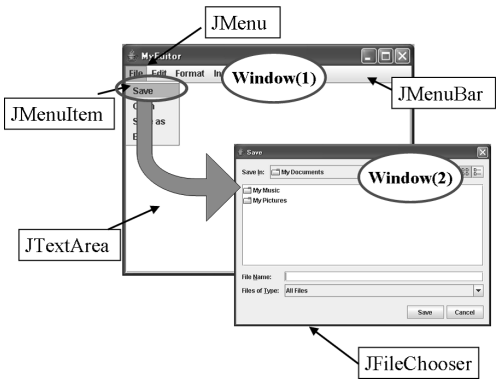

Figure 2: Example of action widgets

In the windows of Figure 2, Java Swing widgets such as JMenuBar, JMenu, JMenuItem, JTextArea and JFileChooser are used. In this text editor, when end users select the "File" menu, some menu items, such as "Save", "Open", "Save as" and "Exit", are shown. When end users select "Save", a window for saving the file is displayed. These menu widgets, such as "File", are JMenu widgets, and these menu item widgets are JMenuItem widgets in Java Swing. Windows are displayed by interacting with JMenuItem widgets, thus they are classified as action widgets.

However, no action is performed by interacting with JMenuBar and JTextArea, because JMenuBar widgets are mats to place JMenu widgets on, and JTextArea widgets are used for inputting text. Thus, JMenuBar and JTextArea widgets are nonaction widgets. In this way, we classify all widgets in Java Swing into action widgets and nonaction widgets.

Upon operating on action widgets, user events, such as pushing buttons and inputting by keyboard, occur. There are several classes for detecting these user event occurrences in Java Swing, and these classes are called "listeners". In our method, the relationships between action widgets and listeners, i.e., the listeners that can be used to obtain events on a certain action widget, are written in the listener database.

When end users operate a widget, various processes corresponding to the operation of the widget are invoked. Listeners are used to detect the occurrence of user events. There are several types of listeners, such as those detecting mouse clicks and key strokes. That is, different listeners detect different user events. Also, listeners are added to each widget. In many cases, the types of user events occurring on a widget depend on its type. That is, the types of listeners added to a widget depend on its type. Thus, it is necessary to store the correspondences between listeners and widgets.

For example, "ActionListener" is one of the listeners, and it can be used for detecting mouse clicking events on JButton widgets. Thus, ActionListeners are added to JButton widgets to detect mouse clicking. This relationship between ActionListener and JButton is written in the Listener Database. An example of a listener database is shown in Table 1.

Table 1: Example of listener database

Action widgets	Listener	Type
JButton	ActionListener, ChangeListener, ItemListener	Left Click
JMenuItem	ActionListener, MenuKeyListener, ItemListener	Left Click
JTextField	ActionListener, KeyListener	Keyboard Type

3.2 Window database

In Java Swing packages, there are two main types of window generation schemes. One is the type in which windows are displayed by creating objects in the windows. In our method, this type is called the "object creation" type. Examples of this type of window are JFrame and JDialog in Java Swing. In these types of windows, the objects that are created are shown using a method called "setVisible". The other is the type in which windows are displayed by calling the specific methods used in each window. In our method, this type is called the "method usage" type. Examples of this type of window are JOptionPane and JFileChooser in Java Swing. For example, JFileChooser windows are displayed by "showOpenDialog", "showDialog" and "showSaveDialog".

According to the usage of windows, these types of window generation methods and windows, and the methods for displaying the windows are written in the window database, an example of which is shown in Table 2.

4 Architecture of our method

The architecture of our method is shown in Figure 3. Operation maps and the operation map system are generated

Table 2: Example of window database

Window display scheme	Widgets	Method of display
Object creation	JFrame	
Method usage	JFileChooser	showOpenDialog, showDialog, showSaveDialog
	JOptionPane	showMessageDialog, showOptionDialog, showInputDialog,etc

by the following three steps in our system.

1. Translating the source programs into JavaML format.
2. Generating operation maps and operation map systems using the JavaML format of the source programs.
3. Weaving the operation map system into target software.

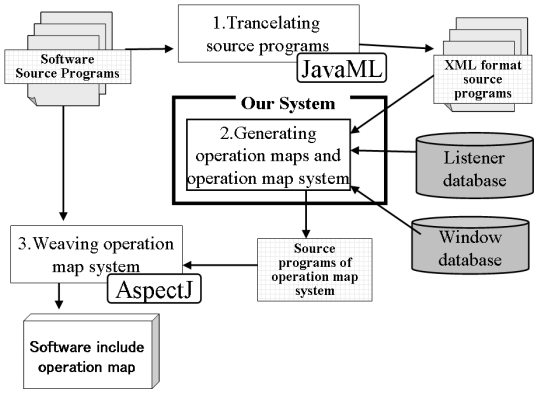

Figure 3: The Architecture of Our Method

4.1 Translating source programs into JavaML format

Our system translates source programs into JavaML format using a JavaML converter. After this step, the translated JavaML format is used for the source programs.

4.2 Generating operation maps

To generate operation maps, the following are extracted on the basis of each window (referred to here as "A").

- Window before A (called "A_{before}").
- Necessary operations switching to A from A_{before}.
- Window switched to after A (called "A_{after}").
- Necessary operations for switching to A_{after} from A.

The relationships among these four items are represented in an operation map. Operation maps consist of the windows of the GUIs, the operation sequences between the windows of the GUIs, which are described as nodes, and the operation sequences, which are described as direct edges. For example, in Figure 1, when window A is the "Graph Wizard(1/4)" window, A_{before} is the "Main", and A_{after} is the window "Graph Wizard(2/4)". The necessary operation for switching to A from A_{before} is [Insert] >> [Graph].

Operation maps can be generated by extracting the relationships involved in all window switchings in the target software. Then, operation maps are generated by connecting all the extracted relationships between windows.

As an example, parts of the source programs of the GUI in Figure 2 are shown in Figure 4. To make the explanation more understandable, the format of Java programming language is used in this figure. However, our system actually uses the JavaML format.

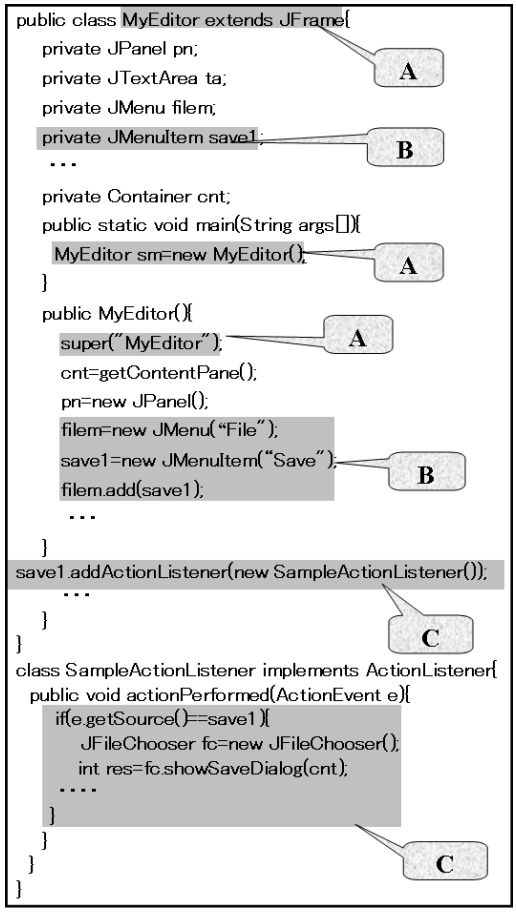

Figure 4: Parts of source programs of GUI in Figure 2

Operation maps and the operation map system are generated by the following procedure:

1. Codes for defining the variables of windows are ex-

tracted from the source programs based on the window database. In Figure 4, the program segments denoted as A are extracted from the source programs in this step.

2. Action widgets on each window extracted in step 1 are extracted. In Figure 4, the program segments denoted as B are extracted from the source programs in this step.
3. Windows displayed by action widgets extracted in step 2 are extracted. In this extraction, the methods that are defined in the listeners for detecting user events are analyzed on the basis of the listener database. In Figure 4, the program segments denoted as C are extracted from the source programs in this step.
4. If codes for displaying windows are found in the extracted codes in step 3, then step 1 is repeated.

From these four steps, the following five items are extracted, then the relationships between windows are constructed from them, and an operation map is generated.

- Titles of windows.
- Class names of action widgets triggering window switching.
- Label names of action widgets triggering window switching.
- Types of user event occurring on action widgets, such as single click and double click.
- Windows switched by the action widgets.

For example, the title of the window in Figure 2 window(1) is "MyEditor". The class name of the action widget is JMenuItem, and the label name of this action widget is "Save". The label name of JMenu before operating JMenuItem is "File". Also, the user event type is "Left click". All these items are extracted from the source programs shown in Figure 4. Finally, our system generates the logical structures of the operation maps and the source programs of the operation map system used for displaying operation maps. The generated operation map for Figure 4 is shown in Figure 5.

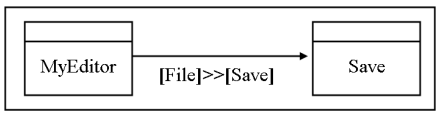

Figure 5: Generated operation map for Figure 4

4.3 Weaving operation maps into software

The generated operation map system is added to the target software using AspectJ. Using the operation map system, the window focused on is highlighted during the end user operation, and this window changes with the end user operation. To realize this highlighting, the operation map system must acquire the window currently focused on from the target software. This acquisition procedure must be added to each class for generating the windows. There are many classes used for generating the windows in the software. Thus, AspectJ is suitable for adding the operation map system to the target software. In this case, the source programs of the target software do not require modification.

5 Evaluation

We evaluated the generation of operation maps from the source programs of the target software using our method. The target software samples are a video rental system (Sample A), an address book management system (Sample B) and text editor software (Sample C). The results are shown in Table 3. In this table, WinH (number of windows by human resources) is the total number of windows in the target software, WinS (number of windows using our system) is the number of windows displayed in the operation maps, EdgH (number of edges by human resources) is the total number of edges that represent all window switchings in the target software, EdgS (number of edges using our system) is the number of edges generated in the operation maps, and Correct Edg (correct number of edges) is the number of correct edges in EdgS.

Also, in this evaluation, the ratio of the total number of windows generated by our system to the total number of windows that exist in the target software is called the window cover rate. The ratio of the total number of directed edges of the window map generated by our system to the total number of edges in the actual window map created manually is called the directed edge cover rate.

Table 3: Correctness of generating operation maps

	WinH	WinS	EdgH	EdgS	Correct Edg
Sample A	21	20	21	19	19
Sample B	4	3	3	2	2
Sample C	21	12	20	11	11

Comparing WinH with WinS, the values of WinH are 1, 1 and 9 more than those of WinS in Samples A-C, respectively. That is, one window in Samples A and B, and nine windows in Sample C are not generated in the operation maps.

This reason for this is that window switchings depend on the state of nonaction widgets. For example, in the address book management system, an entry is selected from a list box, a button is pushed, and then a window is displayed. During this time, the displayed window is determined according to the selection. When no entry is selected, no window is displayed. Thus, in our system, list box widgets were not defined as action widgets. Whatever is selected in the list box widgets switches win-

dows in the operation maps and is extracted as the same window. This is why one window in Sample A, one window in Sample B and four of the nine windows in Sample C were not extracted.

Another cause of the nongeneration of windows is that when action widgets were generated using specific methods, the switched windows were not extracted. For example, in Sample C, developers define a "buildFileMenu" method that generates the instance of button widgets and adds listeners. This is why five of the nine windows in Sample C were not generated.

According to Table 3, the window cover rate is 57% - 95% and the directed edge cover rate is 55% - 90%. Considering these rates, most windows were generated correctly. Considering the direct edge cover rate, most sequences of window switchings were extracted correctly. Also, in these three software samples, the directed edges included in the generated window maps were all appropriate. Therefore, in our method, the directions of window switchings were correctly extracted. We can conclude that our method is an effective means of developing operation maps.

6 Related work

Some other methods have been proposed to help end users learn how to operate software. A method for visualizing the structures of GUIs has also been proposed.

Encarnacao and Stoev proposed a method that helps end users learn operating methods using operation logs [6]. In this method, the target end users are not beginners but those who have operated the software several times. To realize this method, the operation logs of end users are collected, these logs are analyzed, and the frequencies and patterns of function operations are extracted, on the basis of which, instructions on how to operate the software are given. In this method, knowledge of how the target software is operated can be shared. End users can learn operations by referring to this knowledge, thereby making it effective for learning operations that are not written in manuals. However, the logs that end users can base operations on are not exhaustively collected; thus, end users may not always find the solution to the problem.

Michail and Xie proposed a method for supporting end users in recovering from errors [7]. In this method, end users report bugs observed while running software to a database. On the basis of the database, other end users can avoid the bugs by viewing the reports. This method does not support recovery from errors other than the bugs and mistakes encountered by the end users.

Memon et al. proposed a method of generating test cases by extracting the structures and behaviors of GUIs [8]. In this method, the structures and behaviors of GUIs are visualized, then developers examine GUI test cases. To generate the test cases, some graphs are generated, such as "GUI Forest" graphs, which represent the relationships between windows, and "Flow of Events" graphs, which represent the behaviors of GUIs when the software is running. These graphs are generated by displaying all the windows and widgets in the target software. Thus, these graphs can be generated without source programs by executing the software; thus, not all the necessary windows can be extracted. In our method, the windows are extracted from the source programs; thus, all the windows can be extracted.

7 Conclusions

We proposed a method of generating operation maps and adding them to the original software. Using the operation maps, end users can learn software operations effectively. Future works include the following:

- Supporting more window-switching patterns.
- Improving operation maps for more effective learning.

References

[1] B. Shneiderman, C. Plaisant: Designing the User Interface: Strategies for Effective Human-Computer Interaction 4th Edition, Addison Wesley, 2004.

[2] F. Garcia: CACTUS: Automated Tutorial Course Generation for Software Applications, Intelligent User Interfaces (IUI2000), 2000.

[3] T. Bouhadada, M. T. Laskri: DB-TUTOR: An Intelligent Tutoring System Using a Troublemaker Companion, ACS/IEE International Conference on Computer Systems and Applications (AICCSA'01), 2001.

[4] G. J. Badros: JavaML: A Markup Language for Java Source Code, 9th International Conference on the World Wide Web, 2000.

[5] The AspectJ project at Eclipse.org, http://eclipse.org/aspectj/

[6] L. M. Encarnacao, S. Stoev: "An Application-Independent Intelligent User Support System Exploiting Action-Sequence Based on User Modelling", 7th International Conference on User Modeling, 1999.

[7] A. Michail, T. Xie: Helping Users Avoid Bugs in GUI Applications, International Conference on Software Engineering (ICSE'05), 2005.

[8] A. Memon, I. Banerjee, A. Nagarajan: GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing, 10th Working Conference on Reverse Engineering (WCRE03), 2003.