Refactoring Using Event-based Profiling

Adithya Nagarajan
Department of Computer Science
University of Maryland
College Park, Maryland, USA
sadithya@cs.umd.edu

Atif Memon
Department of Computer Science
and Fraunhofer Center for
Experimental Software Engineering
University of Maryland
College Park, Maryland, USA
atif@cs.umd.edu

Abstract

Refactoring is a disciplined process of restructuring software code in order to improve it, e.g., to make it more reusable, reliable and maintainable. The source of information that guides the refactoring process may be the software's user profiles. An increasingly important class of software is event-based software. Event-based software take an event as an input, change their state, and perhaps output an event. They provide new opportunities for refactoring. For example, reorganizing the objects related to an event and restructuring the event handlers based on the behavior of the software. These opportunities require that we collect user profiles at the level of events rather than the code and model the software in such a way that allows refactoring of event handlers. We present new techniques to collect event-level profiles and organize event handlers. We describe our techniques on one class of event-based software – Graphical User Interfaces (GUIs). We demonstrate the practicality and usefulness of our techniques on a large software system.

1 Introduction

Refactoring is the process of restructuring code in a disciplined way [8]. It has multiple goals including improvement of a software's structure for increased reusability [6], reliability [11], and maintainability [12] of code. Refactoring is carried out in two phases: phase one is the research and understanding phase used to identify and understand the code architecture; phase two involves using this understanding to actually restructure the code. The source of information required for refactoring can either be the developer's understanding of the code or can be obtained automatically using program analysis techniques or user profiles. For example, tools such as IBM's Eclipse [2], Refactorit [3], and Flywheel [4] perform refactoring of code automatically by analyzing the code architecture; specialized refactoring techniques such as code optimization rely on user profiles to identify hot paths in the program and use this profile information to perform restructuring [5].

An increasingly important class of software is event-based software. Event-based software take an event as an input, change their state, and perhaps output an event. Common examples of event-based software are Graphical User Interfaces (GUIs), Web applications and Network Protocols. The way that events interact in an event-based software provides new opportunities for refactoring. For example, refactoring of such software may involve reorganizing the objects related to an event and restructuring the event handlers based on the behavior of the software. Also, since most event-based software are developed using a collection of event listeners (usually implemented as methods), they provide opportunities for automatic refactoring. As event interactions are dynamic, one source of information that can be used for refactoring are user profiles of events.

In this paper we describe a technique for refactoring of event-based software. Our technique involves: (1) collecting user profiles at the event level, (2) refactoring driven by the event level profiles, and (3) *dynamic refactoring*, i.e., restructuring, without access to (or need for) source code, of objects in the software with the help of a set of tools that we have developed. We describe our technique using one important class of event-based software – those with a GUI front-end. We have selected GUI based software for our study because refactoring of such software is easy to visualize, they are widely used, and the resulting set of tools are useful for our GUI testing software called GUITAR [1].

In the next section, we explain opportunities that GUIs provide for refactoring and the challenges involved. In Section 3, we provide details of our event-based profiling technique and a dynamic GUI modeling technique. We present an evaluation study that we conducted on GUITAR in Sec-

tion 4. Finally, we conclude with a discussion of future work in Section 5.

2 Graphical User Interfaces

Most of today's software have a GUI front-end and up to 60% of the code in a software is dedicated to GUIs [10]. Maintaining such software involves making changes to the design of the GUI layout based on users' feedback and reorganizing/deleting parts of the code based on usage. GUIs by their very nature provide the following opportunities for refactoring:

- *Changes to the GUI layout*, i.e., reorganizing graphical objects in the GUI. Examples include changing the menu structure and rearranging buttons on the toolbars.
- *Removal of unused event-handlers*, i.e., removing event handlers associated with events that are never used by certain classes of users.

These opportunities require that we collect user profiles in terms of the GUI's layout, thus enabling identification of commonly used events, the GUI widgets and associated event handlers.

The challenges involved in refactoring of GUIs are as follows:

- *Unavailability of source code*. Most GUI applications are built using COTS products or library components for which source code is not available. Hence, conventional code-based program profilers [7] cannot be used to collect user profiles for GUIs. A profiler is needed that does not make use of the source code.
- *Unavailability of techniques to incorporate GUI layout information with user profiles*. Existing code-based program profilers do not provide GUI layout information with user profiles. In order to perform refactoring of GUIs at the event level, a profiler is needed that integrates GUI layout information with user profiles.
- *Bad software engineering practices*. Current software engineering practices often produce software in which the GUI layout information is closely tied to the underlying code and hence is difficult to change. A new GUI modeling technique needs to be developed that ensures loose coupling between GUI layout information and the underlying code. This will allow changes to the GUI layout without modifying the source code.

In subsequent sections, we will present a novel technique for profiling of GUI events and apply it to refactor GUI applications. What distinguishes our technique from existing techniques is our ability to collect event-level profile information even in the absence of source code. The profile information includes GUI widget information and associated events. Widgets such as Button, Menu, TextField and Checkbox are the sources of events. The type of events associated with the widgets include mouse-clicks, type-in-text and item-selection. We use this event level profile information to dynamically restructure the GUI layout. We also describe a technique to de-couple the GUI layout information from the actual source code of the GUI, enabling dynamic restructuring.

3 Event-based Profiling for GUI Refactoring

To address the challenges involved in refactoring of GUI applications, we have developed an event-based profiling technique. In this section we provide details of this technique. We also describe a new GUI modeling technique that allows refactoring without the need for source code.

3.1 Design of Event-based Profiler

We have developed an event-based profiler based on our previous work of reverse engineering technology called GUI Ripping [9]. We have designed the profiler with the following goals in mind: (1) it should be light-weight so that it does not influence the performance of the application, (2) it should be generic to be applied to any type of GUI applications, and (3) it should be easy to implement.

Widgets such as Buttons, Menus, TextFields and Labels are the building blocks of a GUI. Some of these widgets (e.g., Buttons, Menus and TextFields) allow user interactions whereas other widgets are static (e.g., Labels to display text). The users interact with the widgets by performing events. For example typing a character or pressing a mouse button.

The widgets that handle user events have *event listeners* attached to them. Event listeners are invoked when events are performed on the widgets. For example, a *Mouse-Over* event listener for a toolbar button may display a tool-tip. In Figure 1(a) *actionPerformed* is a method of *ActionListener* event listener that handles events on the *Save* menu-item. Note that multiple event listeners can be attached to a widget. For example, a TextField may have a key event listener and a mouse event listener attached to it.

The key idea of our profiling technique is to detect the existing listeners and attach our own listeners. Hence, whenever a user performs an event on a particular widget, our listener gets a message. The choice of event listeners depends on the type of the widget. For example ActionListener is a listener that is attached to widgets such as Buttons and Menus, and ItemListener is attached to Checkboxes.

Our profiling technique involves two steps. First, to reverse engineer GUIs to extract widgets from them. Second, to identify the existing listeners attached to the widgets and

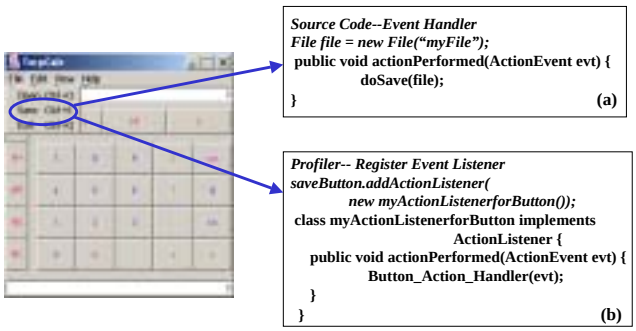

Figure 1. Event-based Profiling

attach our own listeners at runtime. These two steps are described next.

1. We have implemented the profiler in Java. It is implemented as a seperate *Thread* of execution and is activated when the application is invoked. In a Java application, all GUI windows and widgets are instances of Java classes. They are analyzed using Java APIs. For example, API *java.awt.Frame.getFrames()* is used to identify all visible GUI windows of the application. The GUI windows are analyzed, using methods *getComponents* of class *Container* and *getJMenuBar()* of class *JFrame*, to extract widgets.

2. The next step is to analyze the extracted widgets to identify the existing listeners and attach our own listener. For example in Figure 1(b), *myActionListenerforButton()* is the listener that the profiler attaches to the *Save* menu-item, at runtime. Hence, whenever a user performs an event or action on *Save*, the profiler gets a message of the event in addition to the default action that *Save* event performs. The profiler records all this event information.

3.2 Design of Dynamic GUIs

In order to enable efficient refactoring of GUIs, we have developed a new technique of GUI modeling. The goals of our technique are to keep the GUI layout information outside the source code and to make restructuring of GUIs easy.

GUI restructuring involves rearrangment of the GUI widgets and identifying and removing the methods or the event handlers that are not used. The information required for this restructuring can be obtained automatically using the profiler described above or can be obtained from other sources such as users' feedback.

Our technique involves seperating the GUI layout information from the underlying code, and storing it in a XML

```
- <theMenu>
  + <aMenu id="Project">
  + <aMenu id="View">
  + <aMenu id="Generate">
  - <aMenu id="Edit">
      <item type="" action="edit_guiStructure">GUI Structure</item>
      <item type="" action="edit_testCases">Test Cases</item>
      <item type="" action="edit_testOracleInformation">Test Oracle
        Information</item>
      <item type="" action="edit_eventFlowGraphsAndIntegrationTree">Event-
        Flow Graphs and Integration Tree</item>
      <item type=""
        action="edit_coverageReportForGeneratedTestCases">Coverage Report
        for Generated Test Cases</item>
      <item type="" action="edit_coverageReportForExecutedTestCases">Coverage
        Report for Executed Test Cases</item>
      <item type="" action="edit_labelMappings">Label Mappings</item>
      <item type="" action="edit_instrumentedCode">Instrumented Code</item>
      <item type="" action="edit_pddlOperators">PDDL Operators</item>
      <item type="" action="edit_pddlScenarios">PDDL Scenarios</item>
    </aMenu>
  + <aMenu id="Execute">
  + <aMenu id="Options">
  + <aMenu id="Help">
    <stop />
  </theMenu>
```

Figure 2. GUI Layout of a Menu Structure

file. The underlying code reads the XML file each time the application is launched and constructs the GUI automatically. The GUI layout consists of information about the widgets, their positional heirarchy such as location in a menu structure, and their corresponding event handlers. In order to simplify the GUI development and restructuring process we keep each of the event listeners in seperate files.

Figure 2 shows a sample GUI layout information of a menu structure. It consists of the labels of *menu items* and the *action* event listener. The *action* name corresponds to the event listener in the code that should be invoked in response to an event. For example in Figure 2, *GUI Structure* is the label of a menu-item and *edit_guiStructure* is the event listener that handles events on *GUI Structure* menu-item. A user or developer can remove a particular event from the GUI layout by simply deleting the corresponding entry in the XML file and deleting the associated event handler file. For example, the menu-item *GUI Structure* can be removed from the GUI layout by deleting the entry in Figure 2 and deleting the corresponding event handler file *edit_guiStructure* class.

4 Evaluation Study

We implemented both the profiler and XML based GUI. We now present an evaluation study to determine whether our techniques are feasible, practical and useful.

We selected GUITAR as our subject application for this study. GUITAR is a comprehensive GUI Testing Framework that provides a wide range of GUI testing functionality to testers. Its GUI was developed using the XML-based GUI modeling technique described in Section 3.2. We selected GUITAR as our subject application because it has a comprehensive GUI interface and large user base. The

users of GUITAR include students of the under-graduate software engineering course, our research team, and other testers, who are using it as a testing framework for their organizations. Each of the classes of users has specific requirements. For example, our research team need all the testing functionalities available in GUITAR, whereas students need only a limited functionalities required for their course work.

We began our study by distributing GUITAR with the profiler to different users. We then collected the profile information from each user. Analysis of this information led to restructuring the GUI layout for each user class. The restructuring involved removing certain widgets and their event handlers and rearranging the widgets such as moving the menus, that are frequently used, to the toolbar. We accomplished this task by editing the XML GUI layout file and deleting the related event handler files. We created multiple versions of GUITAR, one for each class of users, based on their usage pattern.

We found the profiling technique useful for restructuring of GUITAR's GUI layout. Moreover, the GUI modeling technique made the restructuring task simple and fast. From this study, it is evident that our techniques are practical and can be employed easily for GUI refactoring.

The study also provided us with additional information that helped us to improve the performance of GUITAR and the profiler. Based on the feedback from users, we discovered that our profiler had an adverse effect on the performance of GUITAR. The profiler's task of analyzing the application and attaching the listeners to the GUI widgets was computationally expensive. The way profiler was implemented, it had a live polling *Thread* that kept analyzing and attaching event listeners to the widgets. We fixed this problem by reducing the polling frequency by having an explicit delay of 500 milliseconds in the thread of the profiler. The profile information also gave us event sequences that led to failures/crashes and thereby helped us debug GUITAR.

5 Conclusion and Future Work

In this paper, we presented a novel technique of using event-based profiling for refactoring. We demonstrated the effectiveness of our technique on an important class of event-based software – GUI applications. We have also provided a GUI modeling technique that enables dynamic refactoring of GUIs. Our feasibility study showed that our technique is efficient, useful and practical. In general, our profiling technique may be used for refactoring of other event-based software such as Web applications and Network protocols.

In future, we plan to study usage patterns on a large scale and conduct more studies on other applications. We envision that these usage patterns may be used to design a better software systems. We would also like to study the performance of our profiling technique on other event-based software and on other object oriented programs.

References

[1] GUI Testing Framework, 2003. http://www.guitar.cs.umd.edu.

[2] IBM Eclipse, 2003. http://www-106.ibm.com/developerworks/library/l-eclipse.html.

[3] Refactorit– Java Refactoring Tool, 2003. http://www.refactorit.com.

[4] Velocitis– Tool for .Net Professionals, 2003. http://www.velocitis.com.

[5] D. Bruening, E. Duesterwald, and S. Amarasinghe. Design and implementation of a dynamic optimization framework for windows. In *4th ACM Workshop on Feedback-Directed and Dynamic Optimization (FDDO-4)*, December 2000.

[6] R. E. Caballero and S. A. Demurjian. Toward the formalization of a reusability framework for refactoring. 2002.

[7] P. P. Chang, S. A. Mahlke, and W. mei W. Hwu. Using profile information to assist classic code optimizations. *Software - Practice and Experience*, 21(12):1301–1321, 1991.

[8] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[9] A. Memon, I. Banerjee, and A. Nagarajan. GUI Ripper: Reverse engineering of graphical user interfaces for testing. In *WCRE 2003, The 10th Working Conference on Reverse Engineering*, November 2003.

[10] A. M. Memon. *A Comprehensive Framework for Testing Graphical User Interfaces*. Ph.D. thesis, Department of Computer Science, University of Pittsburgh, July 2001.

[11] W. F. Opdyke. Object-oriented refactoring, legacy constraints and reuse.

[12] L. Tahvildari and K. Kontogiannis. A methodology for developing transformations using the maintainability soft-goal graph. In *Proceedings of the 9th IEEE Working Conference on Reverse Engineering (WCRE)*, pages 77–86, Virginia, USA, 2002.