A Hybrid Approach to User Activity Instrumentation in Software Applications

Martin Dostál and Zdenek Eichler

Dept. Computer Science
Palacký University Olomouc, 17. Listopadu 12
77146 OLOMOUC, Czech Republic
{dostal,eichlerz}@inf.upol.cz

Abstract. This paper introduces novel approach to logging user activity in software applications that provide accurate high-level information about issued commands, commands parameters and user interaction styles without a need for inferring user activity from logged user interfaces events as it is required on most loggers these days. We also demonstrate the proposed approach on a prototype implementation under the OpenOffice.org suite.

Keywords: usability instrumentation, understanding the user, user commands, user interface events.

1 Introduction

Automated capturing of user interaction is an integral and well established part of measuring usability. Although logging user interface events is one of the most commonly used techniques of instrumentation [1,4,6], it has serious limitations in situations where high-level, immediate and unambiguous information about user activity in an application is required since the recorded events must be further extensively processed in order to infer issued user commands from user interface events. Unfortunately, in some cases the user interface events do not provide enough information for correct inference of user commands. Linton [5] used a different technique to capture user activity which enables logging user activity at the level of user commands without processing user interface events. Unfortunately, his approach is unable to retrieve information about used interaction styles (e.g., whether a user command was initiated by a toolbar).

In this paper we introduce a hybrid approach to logging which advantageously combines the both mentioned techniques. Basically, our logging technique is based upon the user-command based logging which provides information about issued commands and their parameters (e.g., when InsertTable command is issued, the parameters contain information about a number of rows, columns and a table name). User command-based logging is surrounded by recording certain user interface events such as opening a menu or opening a window in order to infer information about used interaction style. The combination of the both logging techniques provide high-level, rich (user command and parameters used to issue the command) and accurate information about user activity including used interaction style. Next part of the paper introduces the OpenOffice.org

C. Stephanidis (Ed.): Posters, Part I, HCII 2011, CCIS 173, pp. 566–570, 2011.

Interceptor—a logging tool for the OpenOffice.org suite as a proof-of-concept implementation of the proposed hybrid approach. We also provide a description of the internal architecture and the background on the logging technique behind the Interceptor.

2 Techniques Used to Logging User Activity

In general, there are different approaches to capture user activity. This approach provides immediate and contextually rich information about user interaction with an application. Nevertheless, this way has serious limitations: it is time-consuming for the experimenter and thus not well suited for long term observations as well as for observing a broad group of users. Another, a more automated approach offers the screen recording systems which allow recording of the visual state of the screen and some low-level user events, such as mouse movements, clicks or key press events. The data is usually stored as a movie file. User activity can also be captured using loggers which automatically capture low-level or high-level user interface events or another type of appropriate events triggered by an application. The captured data is stored in a file called a *log*. This approach overcomes some disadvantages of human observation and screen recording systems; logging is usually light on system resources, the data can be further analyzed by corresponding software and logging can be used for short- as well as long-term analysis on a virtually arbitrary number of users.

2.1 Logging User Interface Events

Logging user interface events is based on recording low-level (e.g., keyboard input, cursor movement, mouse clicks or window resizing) and high-level (menu or toolbar selection) user interface events generated by running applications [3]. It represent the most widely used approach today. This approach is represented by MSTracker [6], AppMonitor [1] or RUI [4], for instance. User interface events-based logging provides very detailed, fine grained, low-level data which are useful for a certain kind of user studies such as determining optimal size and layout of user controls. On the other hand, for studies that focus on functionality of an application (e.g., which commands are used, how they are used, which parameter values are preferred) user interface events centered data is not well suited. Logged data must be further extensively processed in order to provide high-level, semantically rich information about user activity. We performed a simple experiment with RUI and AppMonitor loggers on logging a common user command for selecting the “Times New Roman” 14pt font in the font dialog in Microsoft Word 2003. RUI produced 1040 log entries, although due to the type of logged information it is not possible to extract information that the user performed a font change to “Times New Roman” 14pt. Using AppMonitor’s default settings (that does not log low-level events) the same user command produced about 50 log entries. In our approach, such user activity would result in a single line of log file. Since MSTracker is not available for public use we could not perform the same experiment on this logger. Nevertheless, reportedly [6] a menu item selection would result in up to 150 lines in a log file. In addition, further log processing is an unwelcome property for applications which require an on-line, immediate and unambiguous response from the logging framework. This is the case of intelligent user interfaces which continuously track and evaluate user’s activity in order to support the user.

2.2 User Command-Based Logging

User command-based logging has been used by Linton in the OWL [5] system to log user activity in Microsoft Word for Macintosh. In this approach, a logger directly captures issued user commands at the level of the underlying function call, not at the level of user interface events. The logging mechanism is thus entirely separated from the user interface elements. Unfortunately, it has one serious disadvantage; it is not possible to retrieve information about the interaction style (e.g., whether a user command was initiated by a toolbar button or a menu item) used to invoke the user command. For instance, if we invoke the "Open" command three times, firstly using a menu, secondly using a toolbar and thirdly using a keystroke, it will result in three identical log entries. User command-based logging has been used scarcely until today because there is no standard way to capture such information at the level of APIs provided by operating system.

2.3 A Hybrid Approach

We propose a hybrid approach to logging that advantageously combines the both mentioned approaches. Basically, our logging technique is based upon the user command-based logging. In order to enable logging of user commands including the information about used interaction style, we also process user interface events related to interaction styles such as opening a menu or opening a window. The information about the issued user command and logged user interface events is then processed using an interpreter that correctly assigns used interaction style to the corresponding user command. The interpretation of the data is non-trivial. We outline the interpretation algorithms used in the prototype implementation in Section 3.

2.4 Hybrid Approach Integration into Present Desktop Environments

Today's desktop environments strive for consistency and uniformity of applications not only in terms of user interfaces, but also to some extent to unify the internal architecture using various APIs or frameworks. For instance, modern desktop environments provide universal, system-wide APIs for managing user interface events, clipboard operations, undo/redo operations or accessibility (it provides the information about the contents of user interfaces for assistive technologies in order to meet specific needs of disabled people). Unfortunately, the problem with our hybrid approach is that the present environments do not provide such a standardized, system-wide API for logging user commands issued in applications. It makes the implementation of system-wide hybrid approach-based logger infeasible. However, such a simple "user command observing API" that would provide information on issued user commands and their parameters would be fairly easy for developers of desktop environments. In fact, it would be fairly similar to the implementation of an undo/redo framework which tracks issued commands and information on how to undo or redo changes caused by the command. The "user command observing API" would require information about the issued user command and parameters applied to the user command. For instance, when a user inserts a table named "Salary" which has two columns and four rows, the command would be "InsertTable" and parameters will be equal to: "TableName: Salary;Columns: 2;Rows: 4". The next

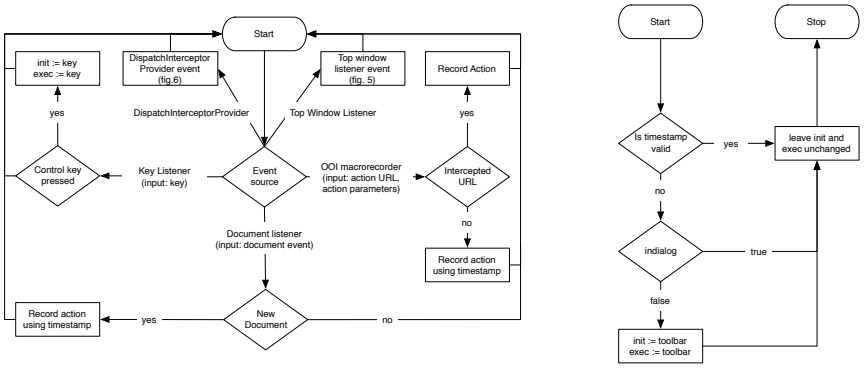

Fig. 1. The Logging Algorithm: The Main Part (left), Record action using timestamp (right)

section introduces the architecture behind the prototype implementation of the hybrid approach-based logger that operates under the OpenOffice.org suite. The "user command observing API" is there substituted by the built-in macro recording framework that provides the information about issued commands and their parameters.

3 Prototype Implementation under the OpenOffice.org Suite

This section describes the internal architecture, in particular the logging algorithm and some implementation issues of the Interceptor. The main algorithm is outlined on the left part of the Fig. 1. In fact, the depicted blocks of the logging algorithm is a way simplified. Most user commands imply that the logging algorithm must be performed repeatedly in order to assign used interaction style correctly. Note, the algorithm variables are not reinitialized at the start of each iteration of the logging algorithm.

Most user commands and their parameters are logged by a custom macrorecorder which replaces the embedded one. Since we use the macrorecorder, Interceptor works under Writer and Calc only. However, some user commands are not macrorecordable. For those user commands we use a DispatchProviderInterceptor object (a part of the DispatchFramework) and a DocumentEventListener object (it listens for document-related events such as creating a new document).

The main issue related to logging interaction style is that there is no straightforward way how to determine interaction style since user command calls are separated from the user interface events. Furthermore, a complication arises from the fact that the user command is logged after the originating user interface event. Nevertheless, the performed action must be associated with an originating interface event that was triggered by activation of the user command. We record events of a TopWindow - a general OpenOffice.org window object, represented by menus, pop-up menus and dialogs (not the system ones), and a keypress events (handled by a KeyListener). Also the Accessibility API is used in order to infer used interaction style. TopWindow events are used to inform about opening, closing or focus change of a TopWindow. We handle only certain kinds of a TopWindow, such as menus, pop-up menus and dialogs. A KeyListener is responsible for determining the keystroke interaction style.

The basic idea behind logging interaction styles is straightforward, see Fig. 1 (right): almost every user interface event initiate a setting of three variables: `timestamp` (current time in miliseconds), `init` and `exec` (interaction style used to initiate and execute user command, values depend on a particular user interface event). Therefore, when a user command is logged, the timestamp and current time values are compared. If the difference between the timestamp value and current time is less than 500 ms, then `init` and `exec` variables are proclaimed as valid and therefore, their values are used to identify the interaction style used to initiate and execute the user command. In other case, the variable values are considered as invalid and then in most cases we interpret such case as toolbar interaction style, since such a style neither do not cause opening or closing of any TopWindow, e.g., a menu or a dialog. The next important issue is related to handling dialogs, especially the non-modal ones. Between opening of non-modal dialog and executing user command from such a dialog the user can perform other user commands from other parts of application. That is why we handle an internal set of currently opened dialogs. This information is used when a dialog is used to execute a user command.

4 Summary

This paper introduced novel approach to logging user activity and a prototype implementation of the proposed approach. Since the architecture behind the logger has been outlined briefly we recommend a tech. report [2] that provides comprehensive implementation details. We believe that our logger could be useful to some other HCI researchers and practitioners and we provide the logger upon e-mail request for free use.

References

1. Alexander, J., Cockburn, A., Lobb, R.: Appmonitor: a tool for recording user actions in unmodified windows applications. Behavior Research Methods 40(2), 413–421 (2008), `http://dx.doi.org/10.3758/BRM.40.2.413`
2. Dostál, M., Eichler, Z.: The implementation of logging user activity in the openoffice.org interceptor. Tech. rep., Palacký University of Olomouc, Olomouc, Czech Republic (March 2010), `http://dostal.inf.upol.cz/data/hci/ooi-techreport.pdf`
3. Hilbert, D.M., Redmiles, D.F.: Extracting usability information from user interface events. ACM Comput. Surv. 32(4), 384–421 (2000)
4. Kukreja, Urmila, Stevenson, William, E., Ritter, Frank, E.: Rui: Recording user input from interfaces under windows and mac os x. Behavior Research Methods 38(4), 656–659 (2006)
5. Linton, F., Joy, D., Schaefer, H.P., Charron, A.: Owl: A recommender system for organization-wide learning. Educational Technology & Society 3(1) (2000)
6. Mcgrenere, J.: The Design and Evaluation of Multiple Interfaces: A Solution for Complex Software. Ph.D. thesis, University of Toronto (2002)