DRT: A Tool for Design Recovery of Interactive Graphical Applications

http://www.cse.unsw.edu.au/~drt

Keith Chan, Annie Chen, Zhi Cong Leo Liang, Amir Michail, Minh Hoai Nguyen, Nicholas Seow
School of Computer Science and Engineering
University of New South Wales, Sydney, Australia, 2052
{kchan,anniec,leozc,amichail,mhng237,nickseow}@cse.unsw.edu.au

Abstract

Nowadays, the majority of productivity applications are interactive *and* graphical *in nature. In this demonstration, we explore the possibility of taking advantage of these two characteristics in a design recovery tool. Specifically, the fact that an application is interactive means that we can identify distinct execution bursts corresponding closely to "actions" performed by the user. The fact that the application is graphical means that we can describe those actions visually from a fragment of the application display itself. Combining these two ideas, we obtain an explicit mapping from high-level actions performed by a user (similar to use case scenarios/specification fragments) to their low-level implementation. This mapping can be used for design recovery of interactive graphical applications. We demonstrate our approach using LYX, a scientific word processor.*

1 Introduction

Almost all productivity applications in use today are *interactive* and *graphical* in nature. By interactive, we mean that the application takes user input (key presses, mouse clicks, etc.) and gives back an immediate response, then awaits for further input, and so on. This contrasts with older batch applications that simply take an input and terminate with the output. By graphical, we mean that the application shows its results in a graphical manner as with a word processor or web browser say. Graphical applications typically have a graphical user interface (GUI) incorporating menus and dialog boxes to facilitate interaction.

In this formal demonstration, we show a way to take advantage of these two characteristics – namely, the interactive and graphical natures of applications – in a design recovery tool. These two characteristics allow us to identify the key actions that are performed by the user in a demonstration of the application, much like use case scenarios, as well as the implementations of those actions. One can then search/browse a database of such actions and their implementations to understand the design of an application.

Intuitively speaking, an application that is interactive allows us to more easily identify actions performed by a user. Specifically, we can look for an "execution burst" that occurs when the user invokes an action. Moreover, the fact that the application is graphical gives us a way to describe the actions identified in this manner. Namely, we can describe an action by using a screenshot of the application display before and after the action is performed.

To illustrate our approach, we have built a tool, named DRT (for Design Recovery Tool), that works with most C/C++ X Window System applications irrespective of the GUI framework used. We shall demonstrate DRT using the scientific word processor LYX 1.2.1. LYX is a rather substantial X Window System application consisting of about 154,000 lines of C++ code.

Our approach is applicable to interactive graphical applications written in an event-driven programming style with alternation between user-initiated events and application responses. Most productivity software including word processors and spreadsheets is of this form. However, certain applications such as flight simulators and games are not, since the application proceeds even while the user is idle.

2 Searching and Browsing Actions

Given a collection of actions — done by others perhaps — it is possible to search/browse those actions to better understand the design of an application. Such an understanding can then be used to perform some maintenance task.

To search an action collection, one can use a *dynamic query*, which involves actually performing an action on a running application. In particular, we would run LYX under DRT and perform an action to see other similar actions

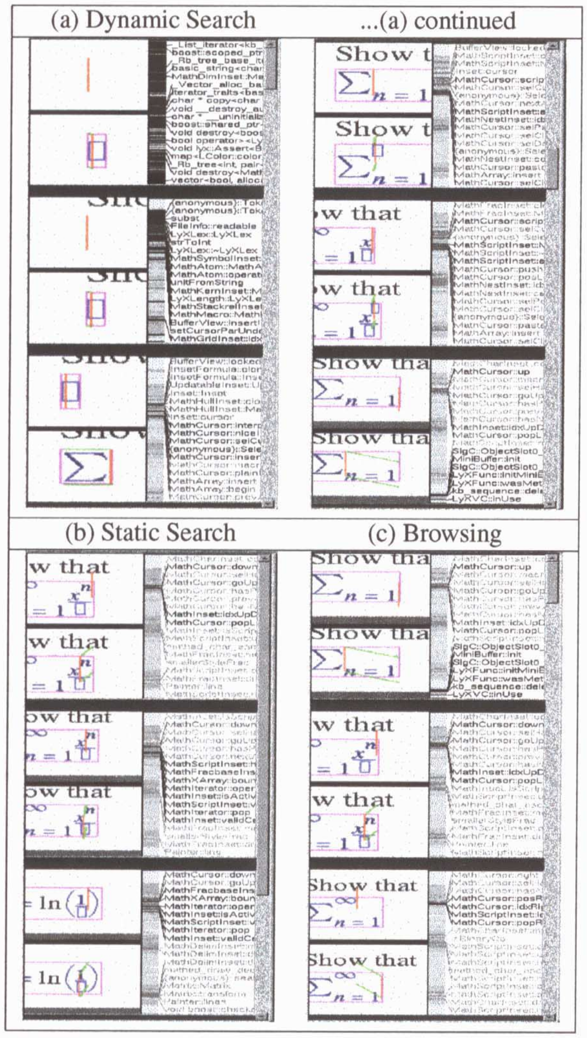

Figure 1. Searching/browsing LyX actions.

already in the action collection. Figure 1, (a) shows such a dynamic query where the user has entered math mode to use the formula editor. The query action is shown at the top followed by similar actions below ranked by similarity to the query action. By looking at the results of this dynamic query, we immediately see typical interactions with the formula editor described visually along with key functions responsible for implementing those interactions.

It is also possible to search by keyword (as with the "cursor down" query yielding the results in Figure 1, (b)) and to browse similar actions (as demonstrated by clicking on the last action in Figure 1, (a), which yields the results in the list of similar actions in Figure 1, (c)).

3 Related Work[1]

There has been related research done on selective tracing of atomic units of activity. For example, the drive-by analysis approach employed in the Jinsight tool [2] allows users to define their own notions of execution "burst" by specifying triggers, filters, requested threads, and burst ending criteria. Jinsight returns dynamic information from the burst only, thus allowing users to prune potentially very large amounts of trace data to only those items of interest. We also have a notion of an execution burst in the DRT tool, but we predefine it to automatically capture bursts that correspond closely to the user's notion of an action in an interactive graphical application. While Jinsight does share the generic idea of an execution burst with DRT, it does not take advantage of either the interactive or graphical aspects of applications to automatically find or describe such bursts.

However, there is an interaction-driven tool, LeNDI, for reverse engineering legacy interfaces [1, 3] that identifies the text screens encountered while an application is used and ways (e..g, keyboard input) for getting from one text screen to another. Such information can be helpful in migrating text-based legacy interfaces to newer platforms (e.g., GUIs or the web) or perhaps wrapping them for interaction with other systems. While it may seem that LeNDI is similar to DRT in the way it automatically extracts screens and transitions between them, LeNDI actually makes no connections to the corresponding burst code. The authors acknowledge that LeNDI does not shed much light on the application code [3]: "Clearly, if the end goal is to extend or modify the system functionality by modifying its current code, then such interaction-based understanding is insufficient because it provides only a model of the user tasks that the system supports and it completely ignores low-level design decisions such as data structures and algorithms."

References

[1] M. El-Ramly, P. Iglinski, E. Stroulia, P. Sorenson, and B. Matichuk. Modeling the system-user dialog using interaction traces. In *8th Working Conference on Reverse Engineering*, pages 208–217, 2001.

[2] W. D. Pauw, N. Mitchell, M. Robillard, G. Sevitsky, and H. Srinivan. Drive-by analysis of running programs. In *Proceedings for of ICSE 2001 Workshop on Software Visualization*, pages 17–22, 2001.

[3] E. Stroulia, M. El-Ramly, L. Kong, P. Sorenson, and B. Matichuk. Reverse engineering legacy interfaces: An interaction-driven approach. In *6th Working Conference on Reverse Engineering*, pages 292–301, 1999.

[1]For more related work, see our ICSE 2003 paper "Design Recovery of Interactive Graphical Applications" by Chan et al.