IMTC 2003 – Instrumentation and Measurement
Technology Conference
Vail, CO, USA, 20-22 May 2003

Automatic Testing of Graphical User Interfaces

Tamás Dabóczi, István Kollár, Gyula Simon, and Tamás Megyeri
Dept. of Measurement and Information Systems
Budapest University of Technology and Economics
H-1521 Budapest, Magyar tudósok krt. 2., HUNGARY
Tel.: +36 1 463-2065, Fax: +36 1 463-4112
Email: {daboczi,kollar,simon}@mit.bme.hu, megyof@sch.bme.hu

Abstract *– Graphical User Interfaces are very difficult to test, since testing requires simulation of the activity of a person. The paper presents an approach where "guided" random selection and activation of the controls is performed. Guidance is implemented on the basis of a probability table.*

The technical means to perform the test is an action recorder (event recorder). Besides testing, this is a useful tool to perform demonstrations and self-guided introduction to the GUI. The recorder has been implemented in MATLAB, and it is available on the WEB.

I. INTRODUCTION

Most of today's application software provide some kind of graphical interface to the user. This is called Graphical User Interface (GUI). The advantage of using a graphical interface is quite obvious. If well designed, it is easy to use, the user needs to follow logical and intuitively natural steps to reach the desired result. It is easy to visualize the progress of data processing, and to give the necessary background information. The actions (if appropriately constructed) can be accessed by simple mouse clicks on graphical objects. GUI's may provide also textual information where necessary, and give the possibility of entering arbitrary alphanumerical values into edit boxes. Graphical visualization is not a constraint, but rather an added benefit [1-2].

An application software with graphical user interface is usually much more complex than conventional alpha-numerical control. The possibility of undetected mistakes in the program is also increased with the many parallel program branches. Because of the desired reliability of the program, there is a larger and larger demand for the testing of the GUI, and also for the test of the whole system, governed by the GUI.

The whole performance of an embedded system (e.g. mobile phone, Automatic Teller Machine, etc.) or a measurement instrument (digital oscilloscope, car diagnostic analyzer, etc.) depend strongly on the software running on the dedicated hardware. The Quality of Service cannot be met without ensuring a certain level of reliability of the software. In the world of virtual instruments the role of testing the software is even more important, since the application software is embedded in a large operating system [3-5]. The application software needs to be tested in an environment which is many times not well documented, sometimes it is released with bugs and features.

One of the big software companies tests graphical user interfaces by letting a couple of people sit in front of the software for a couple of weeks, and letting them try to catch errors. Although this approach might be useful, it is rather arbitrary, time consuming and expensive, and provides no reliable coverage of all often occurring cases. People are different: some try features which are never touched by others. The repeatability of the test is also hard to assure.

Another usual approach is to write a program which tries the functionalities of a GUI by a specially written program. Theoretically this might also work, however, in many cases such a program is as arbitrary as the above one, based on human testers, moreover writing such a program is cumbersome, and it is difficult to reach a good cover of the cases.

Considering the possibilities of testing, the systematic one is based on a formal description of the software system, which allows automatic generation of a testing program. This works well in theory, but in practice, it is rather rare to have a good formal description of the system. Therefore this is rather a theoretical option than a practical one.

We need to mention that for all the tests described above, there is a limitation that running a program without having it erroring out, assures only that none of the programs involved (the GUI, the application program, the operational system) have syntactic errors in the executed program branches, and does not assure that they have no semantic errors. E.g. calculation of the wrong result can immediately be seen from some informative plots like the pole/zero pattern or the transfer function by a specialist, but not by the computer. Until now, ideas of automatically capture and evaluate the computer plots were unsuccessful. In this respect, "natural intelligence" usually outperforms the computer – therefore, having a good engineer before the screen, which is programmed to provide easy-to-capture information for the user, is more effective than any kind of computer programs. A method which tirelessly allows "non-subjective" trials in the software, maybe combined with a human observer, is very effective.

It is a reasonable idea to *simulate the specialist* who tests the software. Why not let the computer act as an ordinary

0-7803-7705-2/03/$17.00 ©2003 IEEE

user would do it; providing mouse clicks, entering numeric or alphanumeric inputs, activating user controls etc. Exhaustive testing of the application software is usually not possible, since the number of possible inputs from the users is virtually non-limited. A random test can help to find many bugs. Completely random test is not always a good approach: it often does not describe well reality. Thus, we have found a heuristic approach with guided randomness appropriate.

II. CAPTURING/REPLAYING ACTIONS: THE ACTION RECORDER

Let us summarize first what are the features we would like to have:

- we would like to be able to act similarly as the user,
- if an error occurs, we would like to be able to store and reproduce it.

Both requirements tell that we want to act *as the user would do it.* We need to produce and collect computer simulated user actions, and replay them when necessary. That is, we need something which records and replays actions. What is more intuitive for a person than to have an *action recorder*, similar to tape recorders, dealing with actions instead of voice and sound? We developed such an action recorder, which will be described here.

Before going into details, we need to consider the system requirements which are necessary to be able to implement such functionality. We need to be able to

- act on any controls (pushbuttons, edit boxes, menus, etc.) as the user can, that is we need to be able to emulate all the user actions,
- record such a sequence,
- replay this sequence,
- bring the GUI into a well-defined initial state.

For testing, it is enough to be able to record programmed actions only, but if properly programmed, the action recorder can record both programmed test actions AND human actions. This allows additional testing. The most typical user actions (typical according the designers) can be recorded and stored for later replay – this allows a quick test of the most common actions in the GUI. Such a possibility can also be used to store demonstration sequences, which can provide an introduction to the GUI, and the same time be used as test sequences assuring that all introductory steps work indeed. Observing the results of such sequences also allow to roughly check the semantics (proper calculations) of the GUI.

The basic operation for such a device is the mechanism to capture each action in the GUI. Here there are two possibilities. One is when the operational system or the application program under which the GUI is realized can return the information of each action. This seems to be THE solution, however, in many cases this functionality is not provided. In such cases we are referred to the second one: the routines which are called after each user action (callbacks) contain a program sequence which stores the corresponding action when executed.

Replaying action can also be done in two ways. If the operational system or the application program offers the possibility to program the mouse to move above an object and perform the push or writing, the user action can be precisely emulated. If this is not the case, the callbacks need to be activated, and maybe the mouse cursor can be moved to the desired place.

The MATLAB environment we used in this project allowed the callback-based operation only, so we implemented the recorder in this way. In the subsection below we are going to briefly describe the main functionalities of our recorder. The functionality of each control is implemented also as a callable function, therefore the recorder actions can also be programmed, allowing the implementation of the random tester (see below).

A. *Functionalities of a recorder*

Figure 1 illustrates the graphical interface of the recorder.

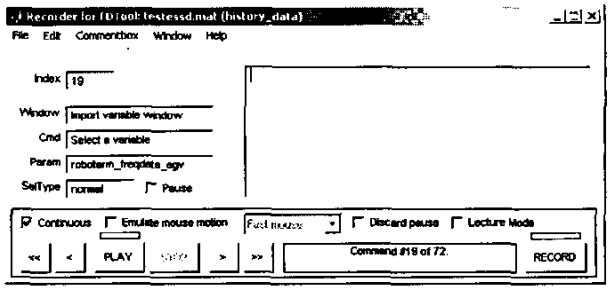

Fig. 1 Display of an action recorder

Meaning of the controls of the recorder:

- the buttons « ⟨ PLAY STOP ⟩ » provide the usual event recorder functionalities using a traditional tape-recorder front-end: fast backward (home), one step backward, play, stop, one step forward, fast forward (end),
- the text box on the right-bottom side is a message display, providing information about the state of the recorder
- RECORD is the start of recording button – this starts the recording, and steps forward after each recorded action
- the tickbox "Continuous" selects between non-stop or step-by-step record and replay
- the tickbox "Emulate mouse motion" makes the recorder to move the mouse cursor above the active control during replay
- the tickbox "Pause" allows to set a breakpoint: continuous replay can be stopped if this tickbox is on for an action (this can be edited manually, after recording)
- the tickbox "Discard Pause" allows continuous execution even if the Pause tickbox is set for certain actions, to allow continuous execution of a demonstration which usually contains pauses
- the tickbox "Lecture mode" allows special stopping: when the recorder stopped: not the recorder display will

appear in the foreground, but the active GUI window, to allow explanations during a lecture
- the large text box on the right-center side is for longer explanations for a user observing demonstrations.

There are additional controls in the recorder display: these are usually filled in by the action-recording step, but can also be programmed. They are:

- "Index": this is the serial number of the current action
- "Window": name of the window where the action takes place
- "Cmd": name of the control to be activated
- "Param": value of the action if necessary (tick/untick in tickboxes, string in edit boxes, etc.)
- "SelType": selection type of the user action (e.g. single or double mouse-click)

The recorder contains also menu items: these allow easy modification of action records: save to a file, load from a file, clear/cut/paste action, insert a Matlab command (a special possibility e.g. to quickly set environment variables by calling a MATLAB command), etc.

The recorder also has built-in error handling and checking capabilities. This feature allows the playback of actions that normally would cause warnings or errors during execution, thus the error handling of the program under test can be compared with the expected behavior.

Effective use of the recorder for tests can be helped by additional features not available in demonstration mode.

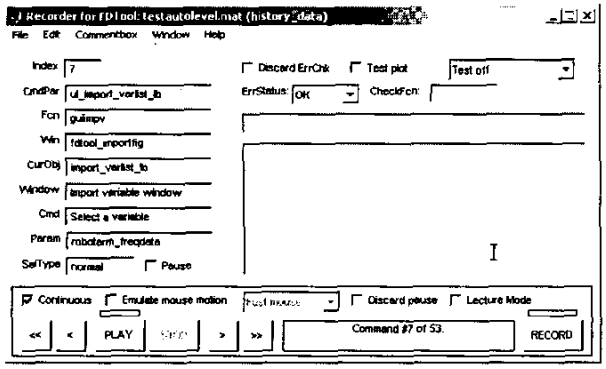

Figure 2 Action recorder in test mode

The controls not shown in Fig. 1 are as follows:

a) Detailed information on each action is extended by precise description:

- "CmdPar": command parameter used in the callback
- "Fcn": the name of the associated function
- "Win": tag of the current window (unique identifier)
- "CurObj": tag of the current object

b) Additional Controls:

- Pull-down menu "ErrStatus": required status of MATLAB after execution of the action: OK/Warning/Error
- tickbox "Discard ErrChk": if set, error status will not be checked
- tickbox "Test plot": collect plots into a postscript file for later quick manual check
- Pull-down menu (currently "Test off"): Test mode off/normal/quick, the mode of the test, e.g. in quick mode long iteration are interrupted after 2 steps
- Edit box "Check Fcn": name of the error checking function to be invoked (usually empty).

The recorder can be used with any MATLAB-based GUI, see [7].

III. THE PROPOSED APPROACH FOR RANDOM TESTING

Our aim was to develop a tester, which simulates the user and automatically tests graphical user interfaces in MATLAB environment.

The tester is a software, which provides the graphical interaction with graphical user interface in a heuristic way. We aimed at catching the software failures, which cause an abrupt termination of the application software. This abrupt termination can be e.g. because of an unexpected input from the user, for which the application software is not prepared. (Users are very talented in this. They will enter the most extraordinary strings, numeric values into edit boxes, thus the application software needs to be tested for a very wide set of possible inputs.)

Although the automatic tester software has been originally developed for improving the reliability of the "Frequency Domain System Identification Toolbox" of MATLAB, the principles are very general and can be applied to other environments also. The graphical user interface of the tester software can bee seen in Fig. 3. The GUI of the tester can be easily adapted to any application software.

A. Testing procedure

To be able to repeat the sequence of testing actions the software environment needs to be brought into some predefined status before starting the testing. This may be a clean, or a well-defined workspace. MATLAB provides both possibilities, which we introduced into the tester software.

After an error has been caught the environment needs to be brought into the same condition than before the test has been started. The action history can be then repeated step-by-step, and the workspace can be investigated after every step.

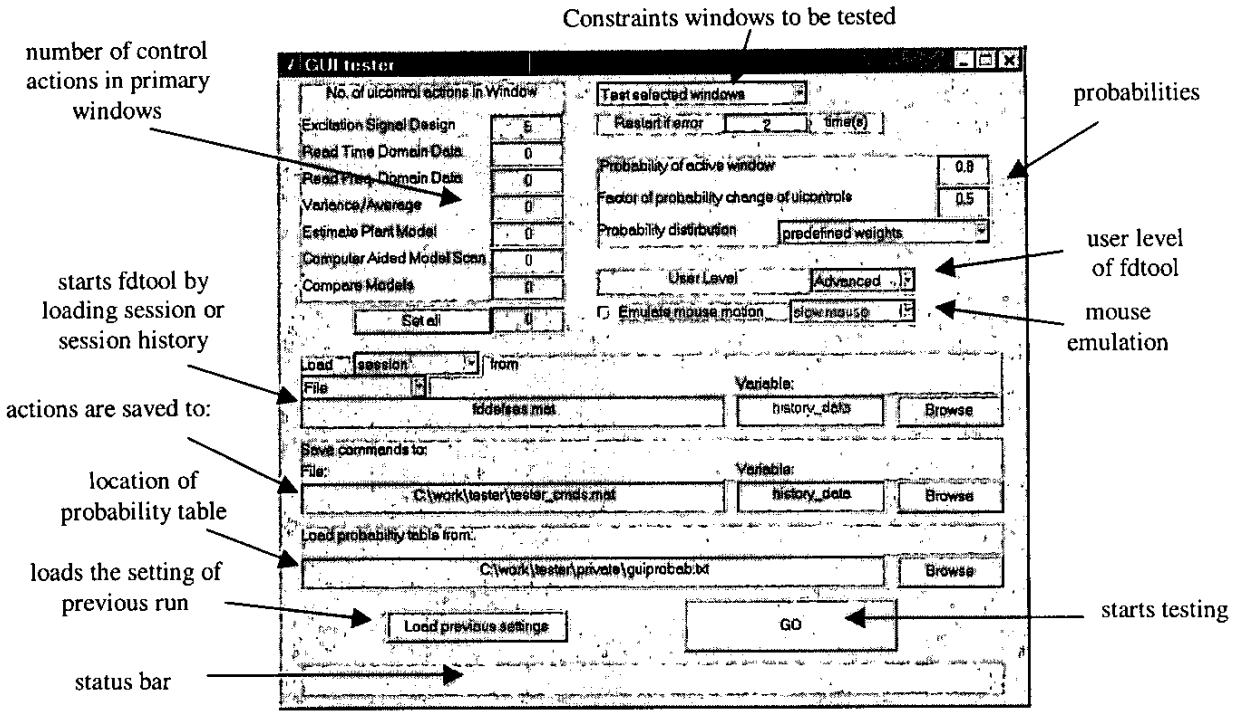

Fig. 3 GUI of the tester

B. Activating the graphical objects through the action recorder

We chose to interact with the graphical user interface through the action recorder. We feed actions into the recorder as if user actions had been recorded, and replay them.

C. MATLAB graphical user interface

Matlab has many graphical objects, like figures, lines, texts, surfaces, user interface controls and menus. All of the graphical objects are uniquely identified with so called handles. The graphical objects have different properties, depending on their type. From our point of view the most important properties are:

- action performed if the graphical object is activated (e.g. click on a push button with left-, right- or double mouse click),
- visibility of the object,
- whether it is active or grayed out.

D. Selecting a graphical object to be tested

There are many graphical objects on a GUI, but not all of them are active at a certain time. Some of them might be inactive, grayed out or invisible. There might be several windows, containing different graphical objects. In MATLAB a window can be made invisible instead of deleting it, allowing to make it available quickly if required again (e.g. help window).

There are two possibilities to select a visible and active graphical object. The first approach is to collect the active objects from all visible windows, and select one from this set. The second approach is to select first a window, and collect the active objects only on that one. We chose this later approach, since it describes better how a user interacts with the GUI. Moreover, this approach is faster.

We defined a certain increased weight to the window in focus, and unity weight to all others. The window is selected on a random basis, making use of the weights. This emphasizes that users usually activate controls on the foreground window.

The tester explores the visible and active graphical objects on the selected window. One of the objects is activated by calling its callback function through the action recorder.

We provided also the possibility to test only a selected window (window in focus). This is useful if software modification affected only a particular window and the others do not need to be tested again.

E. Defining probabilities to uicontrols/uimenus

In order to speed up the testing procedure it is worthwhile to influence the selection of the graphical objects to be activated. There are functions, which are known to be critical, there are others, which do not perform complicated computation or do not contain

branches. It is worth to investigate the critical parts more thoroughly. That is why we introduced weights to graphical objects. The larger the weight is the larger the possibility is that the tester activates it. Assigning zero weight provides the possibility to exclude the object from the testing. The weights are stored in a lookup table, which can be edited with a standard text processor. This requires the unique identification of all graphical objects. Matlab assigns a unique handle to them, however, this happens in runtime. Fortunately, there is the possibility of assigning a so-called tag to every object, which is a string. We propose to assign unique tag to every object and identify them based on this. Different applications provide different ways to identify the objects, but all of them give the programmer the possibility to uniquely identify the created objects. If the programmer of the graphical user interface did not pay attention to that, or the identification is not unique, we loose the possibility to assign weights. Selecting the object based on a uniform distribution is still possible.

One graphical object can be activated several times. Another heuristics, which we introduced, is to decrease the weight of the activated object. The decreasing factor can be adjusted. Assigning zero factor gives the possibility to test one object only ones.

F. Defining sets of possible inputs to edit boxes

A quite complicated question is what numeric or textual input needs to be entered into edit boxes. Since the behavior of the application software depends on the data to be entered, theoretically there is no possibility for an exhaustive test.

We propose to define set of possible inputs to edit boxes. These sets can be assigned either to one particular object or to a group of objects. The sets can be conveniently edited with text processors. These sets can be either additions to completely random inputs, or exclusive sets.

Defining the sets of possible inputs is a hard task. This is the point where an expert of the application software needs to cooperate with somebody who has never seen that particular software. An expert will try to introduce data into the possible input set, which might be a valid data, which is in the range of the required input etc. These data should be included into the sets. However, there is a much larger possibility that the software to be tested is not prepared for an unusual input. E.g. negative number for distance, a string for numeric input, funny characters for filenames etc. These inputs need to be introduced into the possible set also. A fellow who does not know much about the software might be much more useful for this purpose. (A child can be also of great help to extend the set of possible inputs.)

G. Catching errors

The tester stores the actions on the hard drive before it is executed. This ensures that the history of actions and the last action causing the error or an abrupt termination is saved, even if the application software freezes or becomes unstable.

If an error is caught, the testing procedure stops. The whole testing can be replayed step by step, starting from the same initial state as the tester did. The workspace can be investigated after every step.

There is also the possibility to continue the testing after an error has been caught (certainly not from this erroneous state). We can put the system to a predefined initial state, and restart the heuristic random test. If another error is caught, the action history is automatically stored on the hard drive with a different name. This possibility is advantageous at the first couple of test runs of a GUI test, since it collects many errors without the need of user interaction. It might be useful also after a major change in the GUI software.

IV. RESULTS

The concept of testing graphical user interfaces (GUI) has been introduced. We developed a software tool which test GUIs by simulating the user through an action recorder. We proposed a heuristic test procedure: providing random input to GUI, but guiding the randomness with predefined weights assigned to the user controls. The weights change also during the testing process, as the controls are activated. The errors are collected for later investigation.

ACKNOWLEDGMENT

This work has been supported by the Hungarian Scientific Research Fund (OTKA F034900) and the Research and Development Fund for Higher Education (FKFP 0098/2001 and FKFP 0074/2001).

REFERENCES

[1] Virtual Systems, IEEE Instrumentation and Measurement Magazine, Vol. 2, No. 3 (the whole issue), September, 1999.
[2] User Interface Guidelines, http://www.dcc.unicamp.br/ ~hans/ mc750/ guidelines/ newfrontmatter.html
[3] National Instruments, "*G Programming Reference Manual*", 1988.
[4] Baroth, E., Hartsough, C., and G. Wells, "*A Review of HP VEE 4.0*", Evaluation Engineering, Oct. 1997 pp. 57-62.
[5] IVI Foundation Home Page, http://www.ivifoundation.org/
[6] Frequency Domain System Identification Toolbox for MATLAB home page: http://elecwww.vub.ac.be/fdident/
[7] Action Recorder for MATLAB home page: http://www.mit.bme.hu/services/recorder/