Picture-based Task Definition and Parameterization Support System

Vítor Machado[1] Nuno Lopes[1] J.C. Silva[1] José Luís Silva[2,3,4]
[1] EST/DIGARC - Instituto Politécnico do Cávado e do Ave
[2] Madeira-ITI, Funchal, Portugal
[3] DCTI/ISCTE - Instituto Universitário de Lisboa, Lisboa, Portugal
[4] ISTAR-IUL, Lisboa, Portugal
vitormachado1988@hotmail.com , nlopes@ipca.pt, jcsilva@ipca.pt, jose.luis.silva@iscte.pt

Abstract. Applications for task definition and automation are valuable tools to automated software engineering area. This paper describes a solution to support a parameterized task definition using screen capture images. The approach allows the capture of a sequence of actions defined by the user. Through the captured sequence of actions, the approach assists in the implementation of task automation processes.
Based on picture-driven computing the proposed tool aims to reduce the challenges that users face while trying to define tasks. This approach provides also a foundation for the creation of picture-driven based tests for interactive systems, enabling to test any interactive system but also allowing for the definition, parameterization and execution of tests that might involve the use of several independent interactive systems.

Keywords: Automation; Picture-driven computing; Software testing.

1 Introduction

In the user interface of software systems, two interrelated sets of concerns converge. Users interact with the system by performing actions on the graphical user interface (GUI) widgets. These, in turn, generate events at the software level, which are handled by appropriate listener methods. In brief, and from a user's perspective, graphical user interfaces accept as input a predefined set of user-generated events, and produce graphical output. The users' interest is in how well the system supports their needs.

From the programmers' perspective, typical WIMP-style (Windows, Icon, Mouse, and Pointer) user interfaces consist of a hierarchy of graphical widgets (buttons, menus, text-fields, etc) creating a front-end to the software system. An event-based programming model is used to link the graphical objects to the rest of the system's implementation. Each widget has a fixed set of properties which have discrete values at any time during the execution of the GUI, constituting the state of the GUI. The programmer's interest, besides satisfying the user, is

in the intrinsic quality of the implementation, which will impact the system's maintainability.

As user interfaces grow in size and complexity, they become a tangle of object and listener methods, usually accessing a common global state. Considering that the user interface layer of interactive systems is typically the one most prone to suffer changes, due to constant changes in the requirements and added features, maintaining the user interface can become a complex and error prone task. Integrated development environments (IDEs), while helpful in that they enable the graphical definition of the interface, are limited when it comes to task automation or software testing areas. The first area, task automation, aims towards the simplification of task execution, reducing the number of steps/actions that a user has to perform. The second area, software testing, aims at improving the overall quality of the software product, which is of great importance for organizations.

Still, organizations have some difficulties in testing software. The implementation of processes related to quality assurance are difficult by means of a constant technological evolution together with the required time for the implementation of new processes. Software testing is essential to provide organizations with information about the quality of software [1, 16]. Software testing provides a global overview of the software allowing the evaluation of the risks of software implementation. Several techniques are usually applied. These include executing a program or application with the intent of finding software bugs (errors or other defects). Functional testing is a testing type. It refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation. Functional tests aim to validate if the application execution satisfies the requirements. Considering functional test, tools for task automation allow organizations to simplify the process of definition and execution of tests [12, 11, 7]. These tools enable the automatic execution of a high number of tests [4].

This paper presents the development of a system to assist in the automatic creation and execution of tasks representing the graphical interaction between a user and an application. These tasks can then be used to support software testing through the analysis of the graphical interface, i.e., without requiring access to the application's source code.

Using existing technologies of interaction with the Graphical User Interface (GUI), the approach capture actions performed by the users through images/clippings (*keylogger* application type). It is intended that users perform tasks naturally, as usual. At the same time, this system records the graphic objects manipulated by users. These objects will then be used for the construction of an interpretative script for an open source tool named Sikuli [21]. Sikuli is an application used to perform actions through picture-driven computing. In this project, Sikuli will be used for the execution of scripts generated by analyzing the flow of the captured actions and graphical object interactions.

In one hand, the paper intends to describe how to generate and automate the execution of tasks through a picture-driven computing technique. In other hand, the paper aims to present a proof of concept showing that this approach

supports the generation and automation of tasks. The automation of tasks is also helpful to test the behavior of interactive systems.

2 Task Automation

The interaction model incorporates several stages (perception, evaluation, intention and implementation). According to Parasuraman et al. [14], each step can be automated. The same authors define that automation can be applied to 4 classes of functions, i.e., information acquisition, information analysis, decision and action selection, and action execution. Each action can be automated to different degrees (from no automation to full automation) and according to various criteria. Automation in different classes may have several implications, for example, in terms of the performance of users and cost of the consequences of a given decision/action.

Task models enable the identification of the interactions between the user and the device, focusing on the features that should be considered when designing interactive applications [3]. A task is an activity that must be performed to achieve a certain goal. The task models represent the manner of use of applications and describe users interaction rules. These are used as support through the development and test phases [18]. Whereas that the ultimate objective is to specify a methodology to redefine GUI layer of interactive systems, task models seem to be a promising basis for the definition of the methodology, i.e. tasks can be identified using task models and then related to the new GUI layer [10, 13].

Task automation aims towards the simplification of task execution, reducing the number of steps/actions that a user has to perform. Different approaches provide support for task automation. For example, picture-driven computing is useful for task automation of systems already developed and without requiring access to their source code. As stated Sikuli is an example of a tool that follows this paradigm, automating anything that appears on the screen via scripts.

Despite advances in tools that assist the users' tasks, users still encounter difficulties in defining tasks. Several approaches help users through capture and replay techniques as starting points [9]. These approaches provide users an environment to define tasks and execute them, but none provides an approach enabling the parameterization of graphical tasks through the user's actions, i.e., copying and parameterizing the interaction with the graphic objects manipulated by the user to be used later for the generation of an (executable) script.

2.1 Graphical User Interface Task Automation

Several tools can be used for the automation of tasks in systems already developed and/or without access to their source code. This section intends to list some of them.

The first one is a picture-driven computing tool, named Sikuli, that uses image recognition to identify and control GUI components [21]. Sikuli is a programming environment making use of the picture-driven paradigm and is used as

a basis for several research works [17, 19, 20]. Selenium [6] another tool enabling task execution of web applications by the browser in an automated way. Selenium run tests on a finalized system, directly in a browser. Automa [2] is a tool for Windows that automates repetitive tasks on interactive systems. RIATest [15] is an automation tool for GUI tests capable of automating any item on the screen, which is accessible through the Windows UI Automation API. Finally, Eggplant Functional tool [8] enables to execute functional test automation, using an approach based on patterns of pictures. The approach allows the test of any technology on any platform to be made from the user perspective.

We believe Selenium, Automa, RIATest or Eggplant Functional could have been successfully used as an alternative to Sikuli.

3 System's Design

This section is intended to describe the design choices for the support system considered in this paper. First, the system's components are presented and then multiple functional requirements of the application itself described.

The system is made of the two components, the picture-driven execution engine (Sikuli) and a graphical user interface application designed to present the user with the possibility to create and execute scripts.

The creation of the scripts is made through the listening of GUI events and automatic creation of script code, compatible with the Sikuli engine. The execution of the script is made by the Sikuli engine in the background, so the user do not need to code any part of the script.

Script generation is performed during recording. The user triggers the recording by clicking on a button of the application GUI (see Figure 1) and after the recording starts, the application window is minimized freeing the desktop for user GUI interactions. All actions taken on the application itself are ignored by the "listening" events. Thus the user can freely perform the actions over the GUI with the typical applications as it usually does. Finally, the user must signal the application to stop recording (this is done by pressing a button of the application).

To execute a generated script, the user interacts with the same application, but now for running the recorded script through Sikuli.

3.1 Capturing Image Snippets

During the recording of the script, it will become necessary for the user to specify an image snippet for representing the visual region of the GUI which should be interacted with. Two interaction types are available for the user: an action to be made on the image region (e.g. clicking) or an assertion to be made (e.g. check the image is present). Additionally, the system should offer a means to capture an area of the screen and store it automatically.

3.2 Expected Outcome

The expected result of this system is to implement an application that allows the user to create and perform tasks without knowledge of the features of the tools that supports the process, in this case Sikuli.

With this application, organizations or users can build an automation script in an assisted manner, making it easier to build scripts without the need to write any code.

This application extends the Sikuli application [5] by assisting the creation of scripts which can be parameterized, i.e., the application will help the user to create an executable Sikuli script but at the same time it will allow for the user to specify parameters on-the-fly, through dialog windows as the script is executed. This feature allows the reuse of a single script for multiple cases, without the need to change the specific values of the script. The application creates a Sikuli script making use of the functionality available in such tool, in particular the interactive actions and screen image assertions, while remaining intuitive for the user.

4 Implementation

This section describes the implementation of the application proposed in this paper. The application follows the architecture described previously which is responsible for assisting the user in creating a versatile executable script to be run by Sikuli.

4.1 Application's User Interface

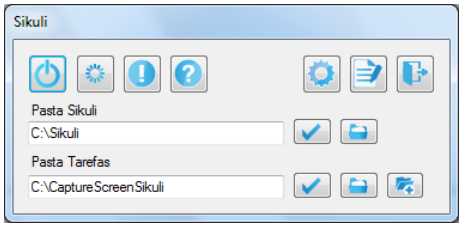

Fig. 1. Application GUI

The Figure 1 presents the graphical user interface of the created application. The left side of the application contains the record/stop, wait, verification and question buttons. The right side of the application contains buttons to run tasks, choose an already created task and exit the application. The buttons positioned at the bottom of the application enable the user to specify the folders for the Sikuli executable files and created recordings.

4.2 Capture User actions

User actions are events performed through mouse's click / double-click and keystrokes. To capture user actions, a so-called library will be used, i.e. *HookManager* library. The library is open source and allows to use the features *user32.dll* windows library to capture events executed with the mouse or keyboard.

The approach makes use of the *MouseDoubleClick* function to create an event to capture mouse actions. This function is also responsible for creating the event control function double click *GetDoubleClickTime* to obtain the double click time which is set on the computer.

The same library allows the use of the *KeyDown* function related to *listening* the keyboard's actions. This function is set to capture the key events. The *SetWindowsHookEx* is used to create keyboard events. An input parameter *WH_KEYBOARD_LL* serves to indicate what type of *listening* needs to be observed.

The applications enables also to capture regions. To set the clipping the user should use the mouse and set the starting point dragging to the point of arrival. The area of the cut will have red contour which can be resized. Consequently the user can increase or decrease previous defined regions. Captured regions are only triggered when one of the following events types occurs: click event, double click event, button *waiting* click or button *question* click.

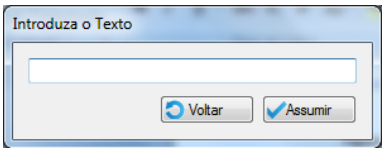

Fig. 2. Window to introduce customizable text

The *waiting* and *question* buttons are related to features that can be useful in a given task. They have the same concept, i.e. validate whether an element exists in the screen. The *waiting* and *question* are intended to search for a particular element on the screen. The first button, *waiting*, indicates that the application is stopped as an element does not appear on screen. The *waiting* features automatically opens the capture region tool to set a region. The second button, *question*, serves to introduce a data in a given element. An click in this button triggers a dialog box that allows the user to define a text that will be presented to the user when executing the task (cf. Figure 2). This functionality allows the creation of scripts which can be parameterized, i.e., the application allow for the user to specify parameters on-the-fly as the script is executed.

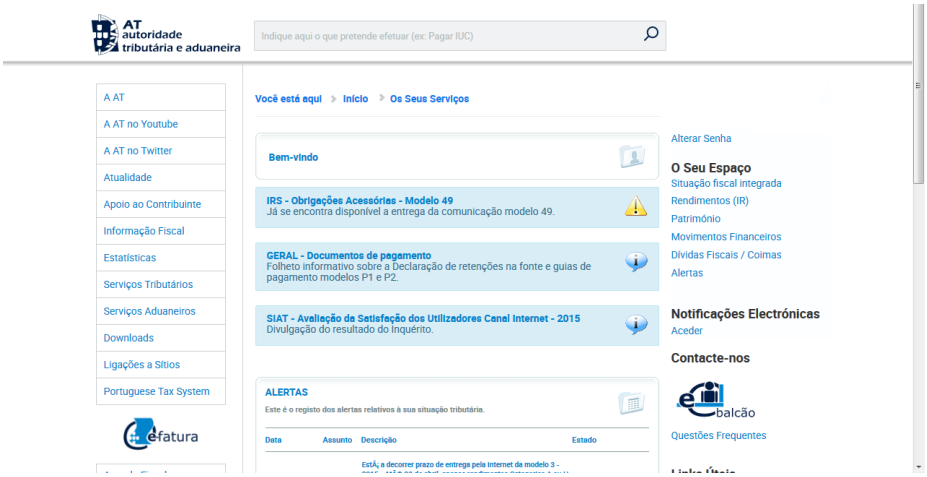

Fig. 3. Home Portuguese finance portal

The graphics library *Graphics*[1] available in the C# programming language was used to record images. The library enables the definition of images' locations and sizes and to copy them in the *png* format.

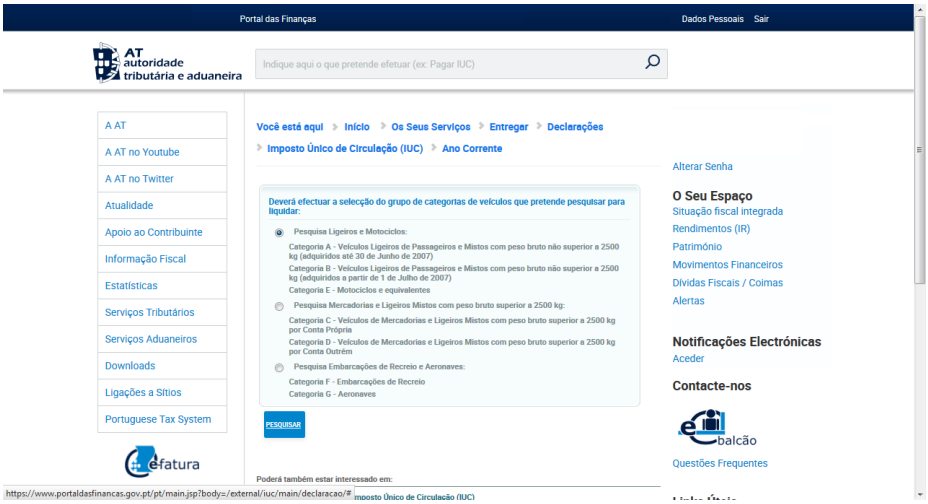

Fig. 4. Find records of the "Current Year"

[1] https://msdn.microsoft.com/en-us/library/system.drawing.graphics(v=vs.110).aspx (last accessed: 5 Dezember 2016)

5 Task Generation Example

This section describes the use of the support system for picture-based task definition. As proof of concept, we propose to access the Portuguese finance portal to extract the contents of tax movements of a given taxpayer related to automobiles and motorcycles owning.

The authentication to the Portuguese finance portal is performed initially through the *My Services* section, as shown in Figure 3.

Using the recording functionality, the support system helps to capture keyboard pressing actions. The sequence of keyboard actions are related to the portal link that contains the text *Pay* which enables to access the *Pay* section.

Within the *Pay* section there are several options focused on the current year. Choosing the option *Search Automobiles and Motorcycles* enables to visualize taxpayer's tax movements, as exposed in Figure 4.

Finally a sequence of keystrokes as follows, first *CTRL + A*, *CTRL + C*, and *CTRL + V* enables to paste the copied data to another field. After finishing recording, the script was created as described in Figure 5.

```
wait("screenSikuli_seq_1.png", 300000);
click("screenSikuli_seq_1.png");
type(Key.DOWN);
...
wait("screenSikuli_seq_6.png", 300000);
click("screenSikuli_seq_6.png");
wait("screenSikuli_seq_7.png", 300000);
type("v" ,KEY_CTRL);
```

Fig. 5. Automatically generated Sikuli script

The execution of the previous script results in the listing from the finance portal of relevant information which may be saved for the user convenience, through an automated procedure. This proof-of-concept was successful and opens the possibility, for users without programming skills, to use this tool and create, run and parameterize automated scripts.

6 Discussion

The validity of the work was demonstrated with an example using one interactive system. In this case the benefits were about the easiness of task definition and consequent automation. This means that an user without programming skills can automate any task. This was illustrated here for tasks running on only one application (i.e. the Portuguese finance portal) but can also be applied to inter-applications tasks. Due the use of a Windows library (i.e. user32.dll) the

approach is currently limited to Windows Operating Systems applications however, we are planning to improve this limitation enabling the automation of tasks performed in Operating Systems.

7 Conclusions and Future Work

This paper presents a tool which aims to be the basis for user defined tasks through picture-driven computing. The process used was described and results were presented and discussed by means of a proof of concept.

Based on picture-driven computing and task automation, the proposed tool aims to reduce the challenges that users face while trying to define a task. This approach provides a foundation for the creation of picture-driven based tasks for interactive systems. By using Sikuli, it becomes possible to automate any interactive system but also allows for the definition and execution of tasks that might involve the use of several independent interactive systems (e.g. websites and desktop applications). The major advantages of this approach over other automation tools is the ease with which users can define a task, parameterize it, and complement it to obtain a test or to perform the task automatically and repeatedly. This work creates a link between task definition and interactive system's quality assurance, enabling testers to be more efficient as they can use the tool for naturally defining tasks as a basis for test suite specification and execution.

In future projects we plan to use the approach with examples involving tests accomplished through independent interactive systems, as well as tester's studies, to assess the usefulness, ease of use, effectiveness and tester satisfaction with the tool.

8 Acknowledgments

José Luís Silva acknowledges support from Fundação para a Ciência e a Tecnologia (FCT, Portugal), through project UID/EEA/50009/2013.

References

1. P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.
2. http://www.getautoma.com/. Automa tool, automate repetitive tasks in the graphical user interface, accessed Apr. 12, 2016.
3. A. Barbosa, A. C. Paiva, and J. C. Campos. Test case generation from mutated task models. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '11, pages 175–184, New York, NY, USA, 2011. ACM.
4. M. L. Bolton, E. J. Bass, and R. I. Siminiceanu. Generating phenotypical erroneous human behavior to evaluate human–automation interaction using model checking. *International Journal of Human-Computer Studies*, 70(11):888 – 906, 2012.

5. T.-H. Chang, T. Yeh, and R. C. Miller. Gui testing using computer vision. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1535–1544, New York, NY, USA, 2010. ACM.
6. B. David. *Selenium 2 Testing Tools: Beginner's Guide*. Packt Publishing, 2012.
7. R. de Kleijn. *Learning Selenium: Hands-on tutorials to create a robust and maintainable test automation framework*. Leanpub, 2014.
8. http://www.testplant.com/eggplant/. Eggplant tool, functional tests automation, accessed Apr. 12, 2016.
9. Y. fang Li, P. K. Das, and D. L. Dowe. Two decades of web application testinga survey of recent advances. *Information Systems*, pages 20–54, 2014.
10. D. Giannakopoulou, N. Rungta, and M. Feary. Automated test case generation for an autopilot requirement prototype. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2011.
11. P. Godefroid, N. Klarlund, and K. Sen. DART: Directed automated random testing. *ACM Sigplan Not.*, 40(6):213–223, Junho 2005.
12. C. Martinie, P. A. Palanque, E. Barboni, and M. Ragosta. Task-model based assessment of automation levels: Application to space ground segments. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Anchorage, Alaska, USA, October 9-12, 2011*, pages 3267–3273, 2011.
13. S. Pangoli and F. Paternó. Automatic generation of task-oriented help. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, UIST '95, pages 181–187, New York, NY, USA, 1995. ACM.
14. R. Parasuraman, T. B. Sheridan, and C. D. Wickens. A model for types and levels of human interaction with automation. *Trans. Sys. Man Cyber. Part A*, 30(3):286–297, May 2000.
15. http://www.cogitek.com/riatest.html. RiaTest tool, automate testing of web applications, accessed Apr. 12, 2016.
16. J. C. Silva, J. Saraiva, and J. Campos. A generic library for GUI reasoning and testing. In *SAC '09: Proc. ACM Symp. on Applied Computing*, pages 121–128. ACM, 2009.
17. J. C. Silva and J. L. Silva. A methodology for gui layer redefinition through virtualization and computer vision. In *Computational Science and Its Applications (ICCSA), 2014 14th International Conference on*, pages 58–63. IEEE, 2014.
18. J. L. Silva, J. C. Campos, and A. C. R. Paiva. Model-based user interface testing with spec explorer and concurtasktrees. volume 208, pages 77–93, Amsterdam, The Netherlands, The Netherlands, 2008. Elsevier.
19. J. L. Silva, J. D. Ornelas, and J. C. Silva. Make it isi: interactive systems integration tool. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 245–250. ACM, 2016.
20. J. L. Silva, J. D. Ornelas, and J. C. Silva. Supporting GUI exploration through USS tool. *Journal of Information Systems Engineering & Management*, (ISSN: 2468-4376):1–4, 2016.
21. T. Yeh, T.-H. Chang, and R. C. Miller. Sikuli: Using gui screenshots for search and automation. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, pages 183–192, New York, NY, USA, 2009. ACM.