Software Test Automation using DEVSimPy Environment

Keywords: discrete event simulation, DEVS, medical software, Web interface testing, test automation.

Abstract

The paper deals with test automation of GUI (General User Interface) software using simulations. The development of GUI software requires a great amount of time and cost concerning the testing aspects. In order to facilitate and speed up the testing of such GUI software an approach based on discrete event modeling and simulation is proposed. Traditionally, the GUI software test automation approaches require the development of testing procedures which are fastidious to carry on. The idea is to perform test automation of GUI software by integrating of existing GUI software testing environment within a DEVS (Discrete EVent system Specification) formalism framework called DEVSimPy. The proposed approach is validated on a real application dealing with medical software which have to respect very strict formats defined by French governmental institutions.

1. INTRODUCTION

This paper deals with a new approach to GUI testing using a discrete event simulation environment. Testing Automation of GUI is a process that insure the correctness of a software with a computer vision approach. Usually, a tester has to implement the generation of user interface events to invoke the changes that result in the user interface in order to validate the software. The work of the tester requires the combination of manual and automated methods. For the manual part the following steps are usually involved: (i) analyze products; (ii) design and write test code; (iii) run the test; (iv) capture and store GUI output; (v) package test code and results along with documentation process. The automated part relies on existing tools which allow: (i) to run the test code and capture the output; (ii) to compare the output with the desired output; (iii) to analyze the result of the comparison.

The blending of manual and automated testing methods is the best way to perform GUI Testing. The GUI software testing traditional approaches require the development of testing procedures which are fastidious to carry on. These is due to the difficulty to manually reproduce the interaction between the interface of the software and the users.

In order to efficiently combined the manual and automated parts an approach based on discrete event modeling and simulation is proposed. The idea is to perform test automation of GUI software by integration of existing GUI testing environments [1,2,3] with DEVSimPy [4,5] framework based on DEVS (Discrete EVent system Specification) formalism [6]. DEVS has been introduced by Professor B.P. Zeigler in 1976 based on the system and automata theories [7]. DEVSimPy is an easy-to-use collaborative environment allowing the development of DEVS model libraries used for the modeling and the simulation of DEVS systems. The DEVSimPy framework allows the implementation of a model library dedicated to the GUI test automation.

The proposed approach is validated on a real case application which use Xplore software provided by the EDL[1] company. EDL is leading European company in the domain solutions for the services of Radiology and Medical Imaging. It develops the Xplore web manager that is software product in charge of services like: medical imaging, personal medical data management and reliability, activity statistical and appointments management. A major social issue related to the medical field concerns exposure risks of software malfunction. In the field of software for health, France has defined national regulations which are constantly changing. They provide a consistent set of regulations to ensure the quality and security software in the field of health; health information technology is becoming an activity area of economy that propose qualification processes. A common set of certification must be implemented, covering the quality of software production and complying with international benchmarks validated by each agency in charge of health in France. The development of medical application software requires a great amount of time and cost concerning the testing aspects which have to respect very strict formats defined by governmental institutions. We will point out how our approach is a efficient solution to facilitate and speed up the testing of such medical application software.

The rest of the paper is organized as follows: section 2 gives an overview of the proposed approach. The main existing tools and approaches allowing to perform GUI test automation are described and analyzed. Resulting problems are pointed out and the proposed solution has been introduced. Section 3 deals with the development of a set of DEVS models allowing the integration of GUI testing tools in the DEVSimPy framework. The library of models is described after a brief presentation of both the DEVS formalism and the DEVSimPy software. The validation of the proposed approach is given in section 4. A real case application coming from the health software French company is presented and the validation of the software using the developed DEVSimPy library presented in section 3 is detailed. Finally conclusions remarks

[1]This research work was supported by the EDL company under Contract: HBFD-345432. Contact: 17 Bis avenue du 8 mai 1945 13130 BERRE L'ETANG

and perspectives are given in the last section.

2. OVERVIEW

The main contribution of the presented work is to propose a DEVS simulation approach for test automation of GUI software. In this section we first present the major open-source GUI test automation software. The problematic associated to the use of a combination of these traditional test automation tools is pointed out. A efficient solution is then introduced.

2.1. Test Automation software

2.1.1. RobotFramework

RobotFramework [8,9] is a generic test automation framework for acceptance testing and Acceptance Test-Driven Development (ATDD). It has easy-to-use tabular test data syntax and utilizes the keyword-driven testing approach. Its testing capabilities can be extended by test libraries implemented either with Python or Java language, and users can create new keywords from existing ones using the same syntax that is used for creating test cases. *RobotFramework* is open source software. *RobotFramework* includes the popular Selenium [10-12] web testing tool. *RobotFramwork* allows some interesting features like:

- Providing support for *Selenium* for web testing, Java GUI testing, running processes, Telnet, SSH, and so on,
- Providing easy-to-read reports in HTML or TEXT format,
- Being platform and application independent,
- Offering a modular architecture that supports tests creation even for applications with several different interfaces,
- Providing a simple library API (Application Programming Interface) for test libraries.

2.1.2. Sikuli

Sikuli [13,14] is a visual technology to automate and test GUI using images (snapshots) of the software under test. It is an open-source research project (released under MIT License) developed at User Interface Design Group, MIT computer Science and Artificial Intelligence Laboratory (CSAIL). It may be used like *Selenium* to control a web page and also other PC (Windows, Mac OS X and Linux) applications, and even an iPhone or Android application [15]. *SikuliScript* is a visual scripting API using Jython (a Java implementation of the Python language) to create automation scripts. *Sikuli* includes an IDE for writing visual *Sikuli* scripts with snapshots. The efficiency of *Sikuli* is due to: (i) the use of a powerful fuzzy based image recognition algorithm to execute actions and (ii) the possibility of using conditional statements to make complex test sequences. *Sikuli* finds on-screen matches with a reference image. It can then perform any keyboard or mouse action at or near the matches. *Sikuli* can use any kind of GUI as native, Flash/Silverlight, cross-platform. The *Sikuli* scripts can be written using its Python API by hand or in the *Sikuli* IDE.

The main limitations when using *Sikuli* to perform GUI test automation are: (i) to write a wrapper to launch scripts; (ii) to handle return values; (iii) to install *Sikuli*. We describe in the next sub-section the drawbacks that arise when using this kind of tools. Sub-section 2.3 will introduce how the proposed solution allows to efficiently respond to these limitations.

2.2. Traditional approach limitations

As explained in the introduction, the main difficulty for software developers is the testing phase. In order to speed-up the development of the set of tests numerous frameworks have been developed [9,12,13,16,17,18]. However the deployment of such a solution is very time-consuming since the user has to integrate different kinds of software. Three kinds of tools can be found in the literature when dealing with GUI application testing:

- Graphical layer testing [13]
- DOM (Document Object Model) analysis [9]
- CLR (Common Language Runtime) objects analysis [16-17]

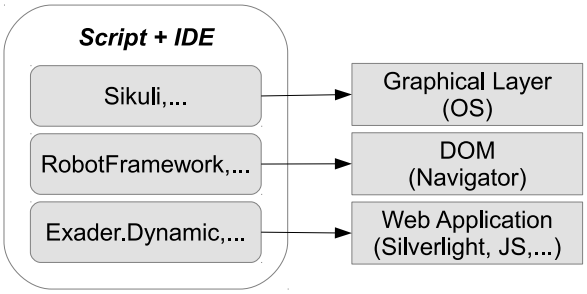

Figure 1. Three kinds of tools for web application testing

In both cases the testing activity relies on the use of scripts or IDE in order to write the tests corresponding to a given software. Figure 1 points out for the three kinds of tools and the main software which can be found: *Sikuli* for the graphical Layer testing [13], *RobotFramework* for the DOM analysis and *Exader.dynamics* [18], *Watin* [16], *WET* [17] for the web application testing through analysis of CLR objects.

In order to develop an efficient test automation scheme for interface web software, a combination of the two first kinds of tools are usually used. The traditional approach when developers have to test an interface web software is highlighted in figure 2(a).

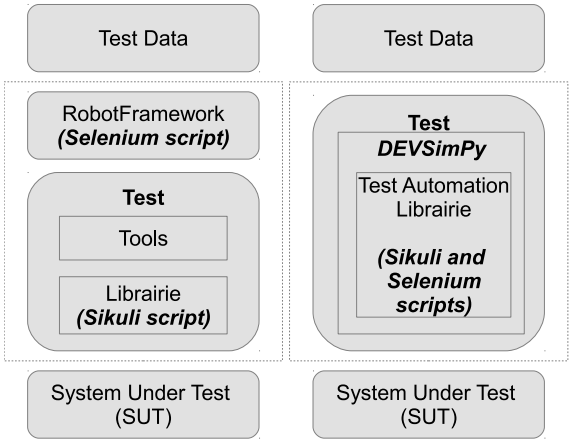

Figure 2. (a) Traditional approach dedicated to the test software (b) Proposed approach based on DEVSimPy environment

The traditional approach involves the following steps:

- From the Test Data specification the developer has to write test scripts defined using the *Sikuli* environment. The obtained scripts compose the Test Library
- The developer has to write a set of scripts in order to visualize the results of the previous test scripts execution. These scripts are written using the *RobotFramework* environment
- The developer has to write a launch script that will allow to both execute the test scripts of the Test Library and the visualization of the obtained results

Each step has to be repeated anytime a new test data specification has to be taken into account. Furthermore each one of these steps is time-consuming since the developer has to write manually the scripts which are very often not so easy to write.

2.3. Proposed solution

In order to facilitate the use of test automation environments combining Sikuli and RobotFramework, we propose to use DEVSimPy simulations as described in figure 2(b). The use of DEVSimPy allows:

- an automatic generation of test scripts using the simulation of models representing the test data specification,
- an automatic generation of the scripts allowing the visualization of results,
- an automatic execution of the previous test scripts and the visualization of the results through a DEVSimPy plugin,
- an automatic install of both the RobotFrameWork (including Selenium) and the Sikuli packages.

Writing test GUI automation consists to create (or manipulate) new graphical models (or existing models stored in libraries) which contain elementary tests. When the user want to implement large test cases, it uses reusability for atomic model and coupled models for structuring instead of manipulate a textual syntax which can introduce additional issues. Furthermore, unit tests becomes possible on atomic models and consolidates the use of DEVSimPy for automating GUI tests. Another point to consider is the benefit provided by the DEVSimPy implementation when user must modify the test in order to validate new specifications imposed by certification organism. The delete or the add of new elementary test is performed by deleting or adding new DEVSimPy model. The order of test execution can be managed using the list of priority of DEVS models. DEVSimPy provides an dialog to manage this one and the priority is changed with a simple drag-and-drop.

The following section presents the DEVSimPy framework and the DEVSimPy implementation of the proposed approach in detail.

3. DEVSIMPY FOR TEST AUTOMATION

This part describes the DEVSimPy library of DEVS models allowing to facilitate the use of automation test software. A first sub-section briefly introduces the DEVS formalism, the second one presents the DEVSimPy framework while the last one deals with the library of DEVS models which have been developed in order to integrate GUI test software.

3.1. DEVS formalism

The Discrete EVent system Specification (DEVS) formalism introduced by Zeigler [6] provides a means of specifying a mathematical object called a system. Basically, a system has a time base, inputs, states, outputs, and functions for determining next states and outputs given current states and inputs. The DEVS formalism is a simple way in order to characterizes how discrete event simulation languages may specify discrete event system parameters. It is more than just a means of constructing simulation models. It provides a formal representation of discrete event systems capable of mathematical manipulation just as differential equations serve this role. Furthermore by allowing an explicit separation between the modeling phase and simulation phase, the DEVS formalism is one of the best ways to perform an simulation of systems using a computer.

In the DEVS formalism, one must specify: (i) basic models from which larger ones are built, and (ii) how these models are connected together in hierarchical fashion. An atomic model allows specifying the behavior of a basic element of a

given system. Connections between different atomic models can be performed by a coupled model. A coupled model, tells how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to hierarchical construction.
A simulator is associated with the DEVS formalism in order to exercise instructions of coupled model to actually generate its behavior. The architecture of a DEVS simulation system is derived from the abstract simulator concepts [6] associated with the hierarchical and modular DEVS formalism. What can provides the DEVS simulation in test automation ? The integration of Sikuli anf RF using DEVS open new perspectives for the automation of test scenarii simulation. For example, with DEVS the user can plan (using plugin) multiple tests in a automated way using a process dedicated to executes multiple simulations.

For the last ten years, research work has been oriented towards the development of environments based on the DEVS formalism [19-24].

3.2. DEVSimPy environment

DEVSimPy (Python Simulator for DEVS models) [5,6] is a user-friendly interface for collaborative modeling and simulation of DEVS systems implemented in Python language. Python is a programming language known for its simple syntax and its capacity to allow modelers to implement quickly their ideas [25]. The DEVSimPy project used the Python language and provides a GUI based on PyDEVS [26] API in order to facilitate both the coupling and the reusability of PyDEVS models. This API is used in the excellent multi-modeling GUI software named ATOM3 [24] which allows to use several formalisms without focusing on DEVS. DEVSimPy is an open source project under GPL V3 license and its development is supported by the University of Corsica Computer Science research team. It uses the wxPython graphic library which is a wrapper of the most popular WxWidgets C library. DEVSimPy can be downloaded at http://code.google.com/p/devsimpy.

The main goal of this environment is to facilitate the modeling of DEVS systems using the GUI dynamic libraries and the drag and drop functionality. With DEVSimPy, models can be stored in a dynamic library in order to be reused and shared (left panel in figure 3). The creation of dynamic libraries composed with DEVS components is easy since the user is coached by dialogs and wizard during the building process. With DEVSimPy, complex system can be modeled by a coupling of DEVS models and the simulation is performed in a automatic way. Moreover, DEVSimPy allows the extension (or the overwrite) of their functionalities in using special plugins managed in a modular way. The user can enabled/disabled a plugin using a simple dialog window. We

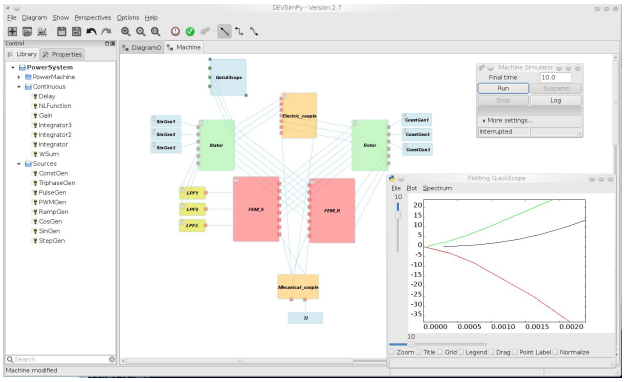

Figure 3. DEVSimPy general interface

propose in this paper the DEVS modeling of GUI test automation scenario through DEVSimPy by implementing a specific dynamic library. Thereby, this library can be reused any time GUI test automation is needed during the application development.

3.3. The TestAutom library

3.3.1. Overview

In order to facilitate the generation of test scripts when using the existing tools presented in section 2.1, the proposed approach consists in integrating these tools into the DEVSimPy framework. This integration will allow the software developers to use the previous tools in an integrated way. The scripts will be automatically generated and the different tools are automatically launched own to a simulation process. The simulation process leans on the modeling phase where the software developers have to build a DEVSimPy coupled model using some drag and drop. The drag and drop are accomplished from a set of atomic models belonging the DEVSimPy library called *TestAutom*.
The library has been developed according to two main subdivisions: the first one called *Sikuli* offers a set of models allowing to automatically generate the test scripts dedicated to the test of GUI and to automatically launch the effective test of the software; the second one called *Selenium* allows the user to complement the previous test by automatically generate log files describing the results of the previously defined tests. Figure 4 highlights the *TestAutom* library (on the left part of 4) and an simple example of use.

The *Sikuli* sub-part of the *TestAutom* library involves a set of atomic models allowing the developer of GUI to both automatically generate test scripts and easily execute them. The models are described in sub-section 3.3.2. The *Selenium* part of the library allows to generate the results of the execution of the test. This part is described in sub-section 3.3.3. Finally we show on a pedagogical example how to use this library in sub-section 4.3.4.

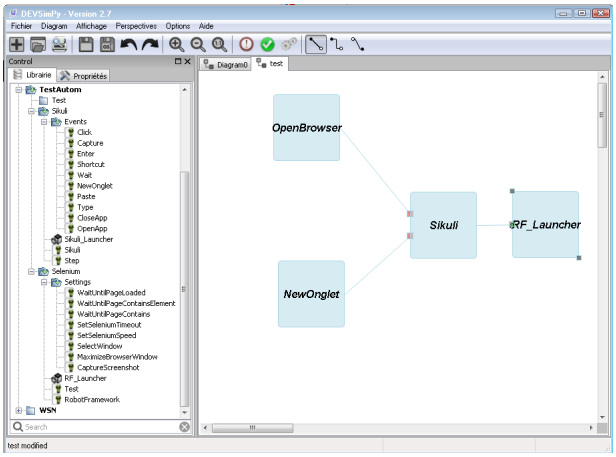

Figure 4. The *TestAutom* library with an example of use in DEVSimPy

3.3.2. The Sikuli sub-part library

The models belonging to *Sikuli* sub-part library are the basic elements which are used to build the model which will be simulated in order to generate the test scripts. These basic elements belong to the events set (left part in figure 4):

- *Click*:In order to simulate the mouse click event
- *Enter*: To simulate the *enter* key press
- *Capture*: To execute a Snapshot operation
- *Shortcut*: To simulate *shortcut* keys combination press
- *Wait*: To simulate a timing statement
- *NewOnglet*: To invoke a new tab in the web navigator
- *Paste*: To simulate the paste function
- *Type*: To populate a text area
- *CloseApp*: To simulate the closing of an application
- *OpenApp*: To simulate the opening of an application

All of these models can be instantiated and combined using drag and drop functionalities in order to build a complex action.
Two other basic elements (atomic models) are playing an important role:

- *Step*: To define an complex action by combining some of the previous events
- *Sikuli*: To aggregate the different steps involved in the definition of the test scenario

All the *Step* instances obtained from the unique *Step* model by simple Drags and Drops are connected to an *Sikuli* instance. This instance is in charge to generate the final test scripts.

3.3.3. The Selenium sub-part library

The models offered in the *Selenium* sub-part library mainly concern the possibility to generate automatically snapshots of the execution results for the test scripts generated using the *Sikuli* sub-part library. Tree atomic models are proposed:

- *RF_Launcher*: To execute both the test and visualization scripts using a plugin contained in its class
- *RobotFramework*: To execute the *Sikuli* scripts and is included in the *RF_Launcher*
- *Test*: To execute *Sikuli* scripts using a command line way (optional functionality)

Furthermore a set of atomic models belonging to the settings directory can be used by the user in order to precise some attributes when dealing with Snapshots or test format printing.
The final step consist in connecting a *Sikuli* instance to a *RF_Launcher* for the execution of *Sikuli* scripts and their visualization using DEVSimPy simulation.

3.3.4. How to use the TestAutom library

In order to use the previously described library, the user (a software developer who wants to test GUI software) has to build (by some drag and drop actions) an interconnected model which will permit through a DEVSimPy simulation to automatically generate the test scripts and execute them in order to obtain the test results under Snapshots of log files format. An example of an interconnected model is given in figure 4. This model involves two atomic models (events) : *OpenBrowser* which is an *OPenApp* model and *NewOnglet*. These two models are aggregated using the *Sikuli* model. The *RF_Launcher* model is connected to the *Sikuli* model to execute the test and visualize the results as explained in 3.3.3.

4. VALIDATION OF THE PROPOSED APPROACH

The previous *TestAutom* library has been validated on a real case application used in the medical domain. The approach is validated in a simple case which shows the basis of use of DEVSimPy for the automation of test GUI.

4.1. Validation benchmark

The sub-section describes the real test case used in order to validate the proposed approach. It concerns an integrated application designed specifically to meet management needs of imaging services public and private institutions in France. A tool named Xplore Management has been designed by a software industry called EDL and it has been chosen for the validation. Xplore Management web software can:

- Accommodate the patient and prepare his case

- Enter the examinations and automatically calculate associated quotes
- Editing reports: use of the word processor *Word* and associated features (templates, glossary of standard paragraphs, ...); incorporation of the text of report in the database (or in the patient record)
- Edit care sheets and technical packages
- Edit schedules and transport of patients, monitor patients unlisted, and much more
- Edit activity statistics
- Manage patient records and history
- Define and manage user profiles (radiologists, radiographers, secretaries, ...)
- Edit and adapt the settings to the service environment
- Be integrated in an existing hospital information system

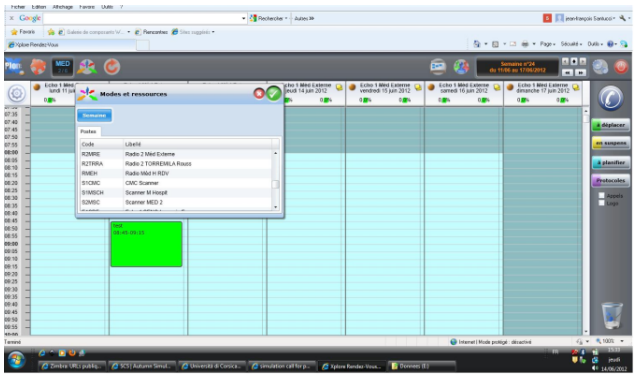

Figure 5. Example of a public appointment activity

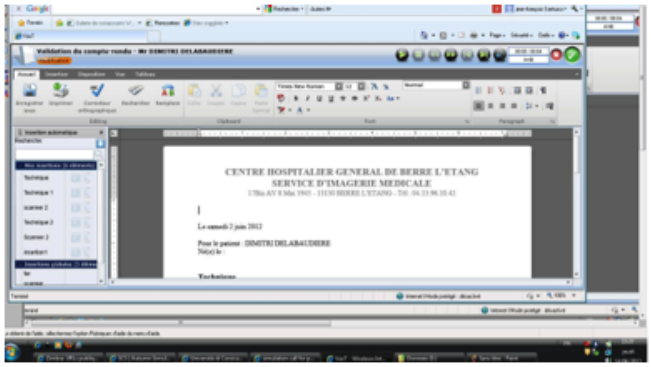

Figure 6. Example of a private report editing

Figures 5 and 6 are two screenshots that point out some of the features of the Xplore management software which is the system under test.

4.2. Case of study: Xplore testing

We describe in this section the design of a test scenario according to the previous validation benchmark. The test scenario consist in the search of a patient by name. It involves four steps (figure 7):

1. The launching of a web navigator (like Google Chrome) and the creation of a new tab using respectively an *OpenChrome* model and a *NewOnglet* model
2. The private web site access by referencing the corresponding url in the navigator url area. As it can be seen in figure 7, four models are used to realized this step.
3. The Back-office access with authentication using login and password. This step is accomplished own to six different event models (*ClickCode*, *Passcode*, *ClickPasswd*, *PastePasswd*, *Enter*2, *Wait*2).
4. The research of the patient called MARIN by its name following be the capture of the results under the form of a screenshot (figure 8). In this case, five event models allows to realize the research of the patient and a last one allows the caption of the snapshot.

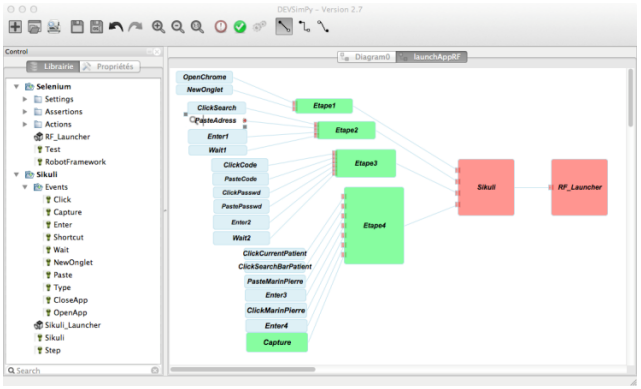

Figure 7. DEVSimPy model of the case of study

As already seen in section 3.3.4, the *Sikuli* and *RF_Launcher* models are used to execute the test and visualized the final results (figure 8). The simulation final result shown in figure 8 highlight the success of the research test since the patient named MARIN has been found.

5. CONCLUSION

The paper deals with an efficient solution for test automation of GUI software using discrete event simulations. We have pointed out that the DEVSimPy framework has brought a solution to the lack of homogeneous test automation environment. Furthermore, DEVSimPy simplifies the combination of test automation software as *Sikuli* and

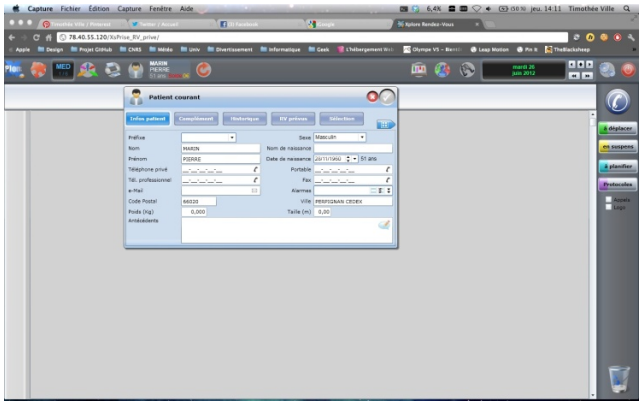

Figure 8. Screenshot of final results corresponding to the simulation of the test case study

RobotFramework. This combination allows: (1) to automatically generate test scripts based on DEVSimPy models stored in dynamic libraries; (2) to automatically execute the test scripts and visualize customizable logs using a simple click approach. Furthermore, using DEVSimpy several variants of original test can be experimented and combined using simulation. A real case application has been employed in order to validate the benefits to apply DEVSimPy simulation. The proposed approach has been adopted by EDL which is a major French company in charge of medical software development. We plan to exploit the DEVSimPy library for GUI testing in various application domains. We envision also to embed analysis of CLR objects using DEVSimPy in order to complete the test scripts according to a code oriented view.

6. REFERENCES

[1] Prabhu, J., and N. Malmurugan, 2011 , *A Survey on Automated GUI Testing Procedures*, European Journal of Scientific Research 64 (3): pp. 456-462.

[2] Chang, T. H, 2011, *Using Graphical Representation of User Interfaces as Visual References*, in Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology, pp. 27-30.

[3] Gupta, P., and P. Surve, 2011, *Model Based Approach to Assist Test Case Creation, Execution, and Maintenance for Test Automation*, in Proceedings of the First International Workshop on End-to-End Test Script Engineering, pp. 1-7.

[4] Zeigler, BP, H. Praehofer, and TG Kim, 2000, *Theory of modeling and simulation*, 2nd Edition, Academic Press.

[5] self—

[6] DEVSimPy, Open-source collaborative software, http://code.google.com/p/devsimpy/

[7] Kohavi, Z. 1978. *Switching and Finite Automata Theory*, McGraw-Hill.

[8] Robotframework. http://code.google.com/p/robotframework.

[9] Laukkanen, Pekka, 2006, *Data-Driven and Keyword-Driven Test Automation Frameworks*, Master Thesis, Helsinki University of Technology , http://eliga.fi/Thesis-Pekka-Laukkanen.pdf.

[10] Hellsten, C. 2006. *Using Selenium with Automation Regression Test*, IBM Developer Works.

[11] Kim, E. H, J. C Na, and S. M Ryoo. 2009. *Implementing an Effective Test Automation Framework*, In Computer Software and Applications Conference, 2009. COMPSACâ09. 33rd Annual IEEE International, 2: pp. 534-538.

[12] Selenium. http://seleniumhq.org/

[13] Yeh, T., T. H Chang, and R. C Miller. 2009. *Sikuli: Using GUI Screenshots for Search and Automation*, in Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology, pp. 183-192.

[14] Chang, T. H, T. Yeh, and R. C Miller. 2010. *GUI Testing Using Computer Vision*, In Proceedings of the 28th International Conference on Human Factors in Computing Systems, pp. 1535-1544.

[15] Song, H., S. Ryoo, and J. H Kim. 2011. *An Integrated Test Automation Framework for Testing on Heterogeneous Mobile Platforms.*, In Software and Network Engineering (SSNE), 2011 First ACIS International Symposium On, pp. 141-145.

[16] WaTin. Web Application Testing in dotNet, http://watin.sourceforge.net.

[17] WET. opensource web automation testing tool, http://wet.qantom.org.

[18] Exader.dynamics. open-source web automation testing tool, http://feed.nuget.org/packages/Exader.Dynamic.

[19] Bergero, F., and E. Kofman. 2011. *PowerDEVS: a Tool for Hybrid System Modeling and Real-time Simulation*, Simulation 87 (1-2): pp. 113-132.

[20] Sarjoughian, H. S, and B. R. Zeigler. 1998. *DEVSJAVA: Basis for a DEVS-based Collaborative M&S Environment*, Simulation Series 30: pp. 29-36.

[21] Wainer, G. 2002. *CD++: a Toolkit to Develop DEVS Models*, Software: Practice and Experience 32 (13): pp. 1261-1306.

[22] Quesnel, G., R. Duboz, É Ramat, and M. K Traoré. 2007. *VLE: a Multimodeling and Simulation Environment*, in Proceedings of the 2007 Summer Computer Simulation Conference, pp. 367-374.

[23] Traoré, M. K. 2008. *SimStudio: a Next Generation Modeling and Simulation Framework*, in Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, pp. 67.

[24] Lara, J., Vangheluwe, H., 2002, *AToM 3: A Tool for Multi-Formalism and Meta-Modelling*, Fundamental Approaches to Software Engineering, pp. 174-188.

[25] Sanner M.F., 1999, *Python: a programming language*

for software integration and development, J. Mol. Graphics Mod, vol. 17, pp. 57-61.

[26] Bolduc, J. S., Vangheluwe, H., 2001, *The Modelling and Simulation Package PythonDEVS for Classical Hierarchical DEVS*, MSDL Technical Report MSDL-TR–01. Montreal, Quebec, Canada.