CTest Integration of Sikuli Automated GUI Testing

Release 0.00

Evan Schwab, Arnaud Gelas, Lydie Souhait, Nicolas Rannou, Kishore Mosaliganti, Sean Megason

June 21, 2011

Megason Lab, Department of Systems Biology, Harvard Medical School

Abstract

In order to test our software GoFigure2 [1] for the visualization and the processing of extremely large $3D+t$ microscopy images, we utilized the visual programming technology Sikuli [4] to automate our GUI testing. We then integrated these Sikuli tests into CTest and used CDash to report the results on a dashboard with the view to test our software automatically every night and detect bugs as soon as they occur. Our rate of releases more than doubled as a result.

Source code and examples are available here: https://github.com/gofigure2/SikuliCTest

Contents

1 Introduction

GUI testing is a key aspect of software development. Having a well defined testing protocol is very important in maintaining a productive flow, especially on the eve of a new release. As the software becomes more complex, testing each feature can be immensely time consuming. We collected a list of over 400 different GUI tests for the GoFigure2 software [1] which involved endlessly repetitive clicking and checking. This task took around 5 hours to complete each time a new release was ready. And debugging would push release dates back days or even weeks. We decided to tackle this problem by utilizing a new scripting language called Sikuli [3, 4, 5] that can automate any visual operation. In this way we could write scripts to automate the GUI testing. This cut the time to complete each test drastically. Furthermore, we have integrated the automated tests with CTest. Therefore we are able to run the collection of tests every night and find bugs as they arise instead of the night before a new release. As a result, the rate of new releases increased drastically from one release every five months to one release every two months.

The paper is organized as follows, first we will briefly introduce what is Sikuli (in section 2.1) and show one simple example (in section 2.2). Then we will show a real test example on GoFigure2 (in section 3) and how to integrate unit tests in CTest (in section 4). Finally, we will review the current limitation of Sikuli and this approach for GUI Testing (in section 5) and conclude (in section 6).

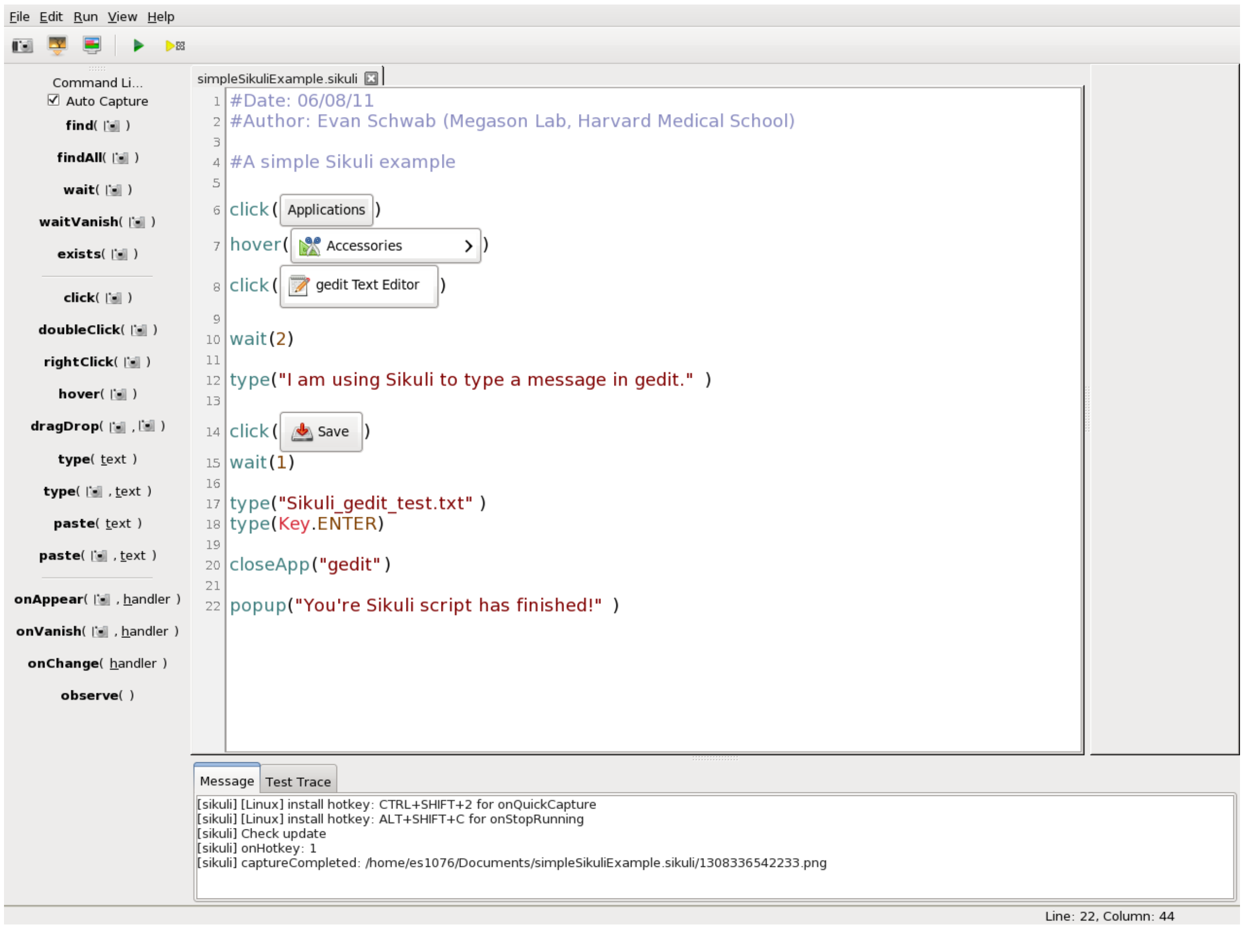
2 Sikuli

2.1 What is Sikuli?

Sikuli [3, 4, 5] is a visual technology to automate and test graphical user interfaces (GUI) using images (screenshots). Sikuli includes *Sikuli Script* and *Sikuli IDE*.

Sikuli Script automates anything you see on the screen without internal API's support. Sikuli Script is a programming language that uses screenshot images as variables and objects in order to automate graphical user interface functions. The Sikuli Script is built on a Jython (Python for Java platform) library which uses Python syntax in addition to a number of special Sikuli functions for acquiring and handling screenshot images and performing mouse, keyboard actions.

In the *Sikuli IDE*, the user can write scripts that include screenshot thumbnails so they can visually track the functions of their code. The most useful functions from Sikuli scripts are conveniently found as buttons on the IDE. When using the IDE, Sikuli then saves a script in a directory with a unique extension .Sikuli. The directory consists of a python script file (script file to be used by Sikuli), an html file, and a list of all the screenshot images used in the script.



```
#Date: 06/08/11
#Author: Evan Schwab (Megason Lab, Harvard Medical School)

#A simple Sikuli example

click( Applications )
hover( Accessories > )
click( gedit Text Editor )

wait(2)

type("I am using Sikuli to type a message in gedit." )

click( Save )
wait(1)

type("Sikuli_gedit_test.txt" )
type(Key.ENTER)

closeApp("gedit")

popup("You're Sikuli script has finished!" )
```

```
[sikuli] [Linux] install hotkey: CTRL+SHIFT+2 for onQuickCapture
[sikuli] [Linux] install hotkey: ALT+SHIFT+C for onStopRunning
[sikuli] Check update
[sikuli] onHotkey: 1
[sikuli] captureCompleted: /home/es1076/Documents/simpleSikuliExample.sikuli/1308336542233.png
```

1024

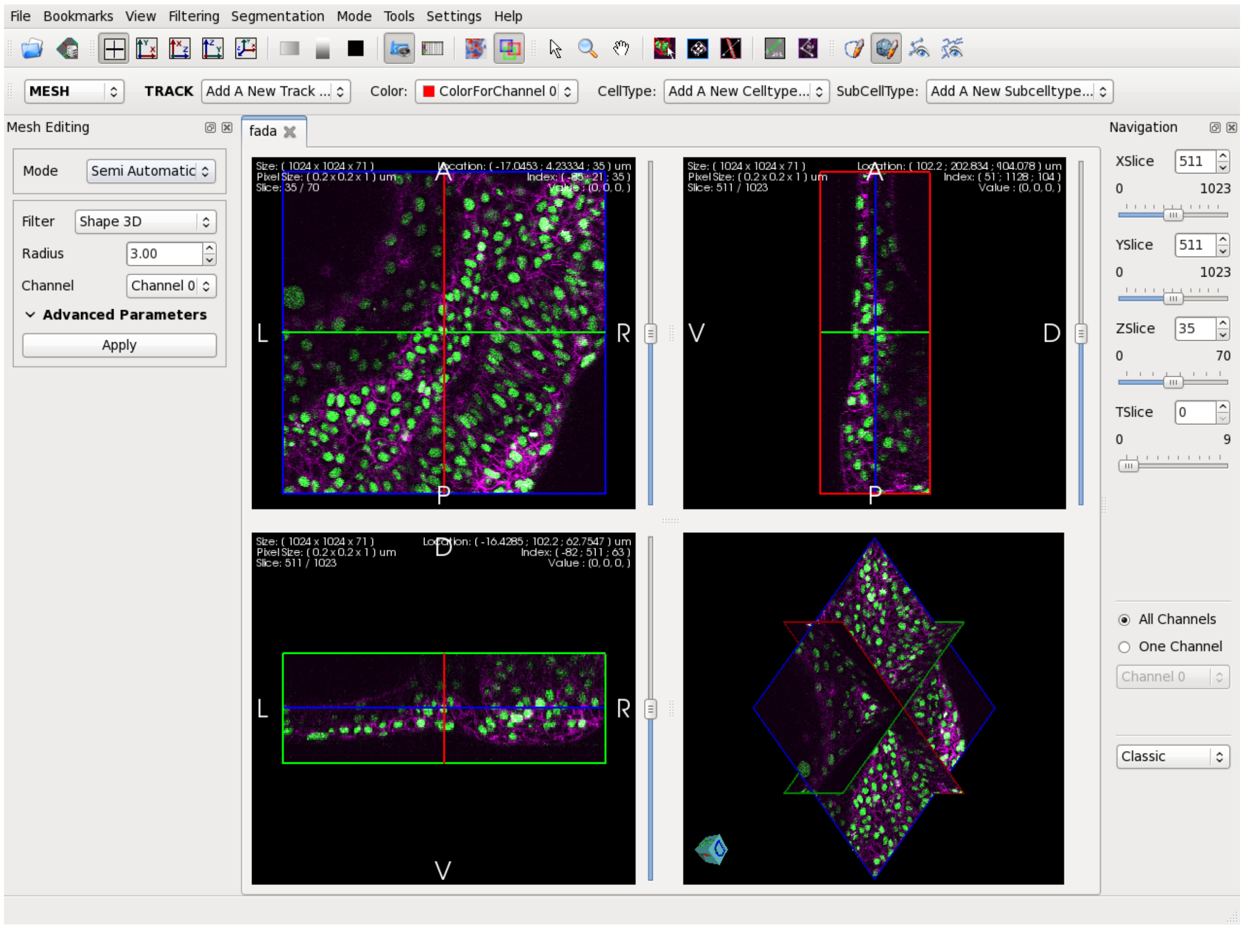
Figure 1: Simple Sikuli Script Example in Sikuli-IDE

2.2 Simple Sikuli example

Figure 1 shows the Sikuli-IDE for version 0.10.2. On the left-hand side menu is a list of commonly used Sikuli commands like *find*(), *click*(), and *type*(). The picture of the camera within the parentheses indicates the screenshot input variable. Clicking on any of these menu functions will automatically set up the Sikuli screenshot mode, whereby the Sikuli-IDE disappears and what is on the screen at that moment is available to capture by selecting a rectangular region of interest. The function will then appear in the scripting domain with the snap shot in parentheses as shown in Figure 1. The user may also type in the function and invoke the screenshot mode by clicking on the camera in the top menu or by using a keyboard shortcut. The simple Sikuli script example in Figure 1 will open gedit Text Editor, type a message, and save the file.

3 Example

The GoFigure2 software [1] (See Figure 2) is designed to automate cell segmentation and tracking for high resolution microscopy data and the Megason Lab uses it to create cell lineages of developing zebrafish emrbryos. The Sikuli script in Figure 3 shows a fragment of a unit test for our GoFigure2 software. The unit being tested here is the Toolbar menu which consists of different modes used to analyze data set plots.



Mode: Semi Automatic
Filter: Shape 3D
Radius: 3.00
Channel: Channel 0
Advanced Parameters

Size: (1024 x 1024 x 71)
Pixel Size: (0.2 x 0.2 x 1) um
Slice: 35 / 70
Location: (-17.0453 ; 4.23334 ; 35) um
Index: (-85 ; 21 ; 35)
Value : (0, 0, 0,)

Size: (1024 x 1024 x 71)
Pixel Size: (0.2 x 0.2 x 1) um
Slice: 511 / 1023
Location: (102.2 ; 202.834 ; 104.078) um
Index: (51 ; 1128 ; 104)
Value : (0, 0, 0,)

Size: (1024 x 1024 x 71)
Pixel Size: (0.2 x 0.2 x 1) um
Slice: 511 / 1023
Location: (-16.4285 ; 102.2 ; 62.7547) um
Index: (-82 ; 511 ; 63)
Value : (0, 0, 0,)

XSlice 511 (0–1023)
YSlice 511 (0–1023)
ZSlice 35 (0–70)
TSlice 0 (0–9)

Figure 2: GoFigure2 GUI Interface

In total there are four modes which are represented in the first line as the vector Toolbar and for each of these modes we want to test their expected mouse functions. For instance, if we are in the zoom Toolbar mode then we want the data set plot to zoom in when the user scrolls the mouse wheel forward. Likewise, if we are in pan mode then we want the plot to move to the left if the user holds down the left mouse button and moves the mouse to the left. Also importantly, we want to see that these two mouse actions are unique for their respective Toolbar modes; the test will include an assertion that scrolling forward will zoom in when using zoom mode and will do nothing when using pan mode.

This breakdown of modes and actions lends itself to a nested for-loop scripting approach. By representing these variables as a vector the software developers can easily add new modes or actions to each list to be tested, when the software is updated. Similarly, the tester can easily replace any of the screenshot images with images of updated GUI icons.

In addition to representing the modes and actions as the vectors Toolbar and Mouse, respectively, we have included the different data set plots in the vector ViewRegion. These images represent the *xy*-, *xz*-, *yz*-, and *xyz*-views of zebrafish microscopy data taken in the Megason Lab. After capturing these images we can then use them to locate their initial pixel coordinates within the GoFigure2 GUI using the *find*() function. These points become our reference positions to calculate directional movement. For example, when testing the pan mode, we mark the initial position of each image, use the *mouseMove*() function to a chosen location, and then use *find*() to obtain the new coordinates. We can then assert if the image panned as it should have. Similarly for zoom, we can first find the area of the image and then find the area of the zoomed in image. To test the result when the user clicks in a certain mode, we've made use of Sikuli's *assert exists*() function to ascertain the existence of our desired results (see Figure 3).

This one GUI test is then combined with other GoFigure2 tests in succession and the Sikuli Unit Testing CTest integration will identify which tests pass and which fail and when. Many of the units need to be tested in a certain order. For example, in order to test the Toolbar modes on the data sets, one first needs to test the software's data import functionality.

4 Integration with CTest

4.1 CMake

One of Sikuli's primary uses is the automation of GUI testing for software developers. For GUI testing, it is useful to compartmentalize different software functions and test each unit individually.

The newest versions of Sikuli come equipped with a Unit Testing feature based on JUnit [2]. Unit Testing can be called using the IDE or by command line and enables the user to write multiple functions in a script which are run individually. At the end of the run, the Sikuli output will tell the user how many tests ran successfully and which ones failed and where.

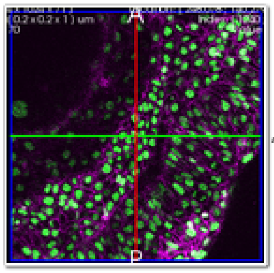
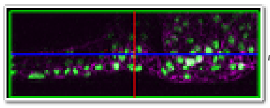
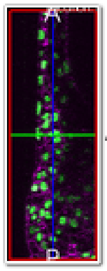
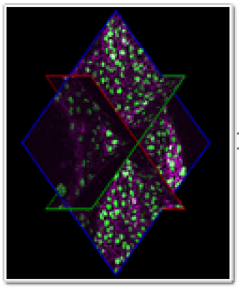
Here we connect this Unit Testing feature to CTest, which allows us to report results on a public dashboard using CDash. To this end, we developed cmake functions to discover where Sikuli is installed and special functions to add Sikuli tests in the list for CTest.

Here is a typical CMakeLists.txt file, to be able to use such functionalities:

```
# first let's include CTest and enable testing

include( CTest )
enable_testing()

# Then let's
#   - figure out where Sikuli is installed
#   - define function to add Sikuli tests

find_package( Sikuli )

# here we add a Sikuli test
add_Sikuli_test( GoFigure2ToolbarTest toolbar.Sikuli )
```

Users can either maintain an image library or keep images in the corresponding Sikuli script directory (default). If one uses an image library, the corresponding path must be set during CMake time in the SIKULI_IMAGE_LIBRARY_DIR variable.

```
Toolbar = ("default", , , )


Mouse = ("Left","Right","Middle","Scroll")
#Actions = ("single_click","double_click","drag_up2down","drag_down2up","drag_left2right","drag_right2left")


ViewRegion = (
```

```
)
Init_view = [None]*4
pointx = [None]*4
pointy = [None]*4
for i in range(4):
        Init_view[i] = find(Pattern(ViewRegion[i]).similar(0.3))
        pointx[i] = Init_view[i].x+ int(round(0.3*(Init_view[i].w)))
        pointy[i] = Init_view[i].y+ int(round(0.6*(Init_view[i].h)))
        print pointx[i],pointy[i], "this is pointx and pointy"
        print Init_view[i]

for mode in Toolbar:

        if mode != "default":
                click(mode)

        for j in range(4):

                #single left click
                mouseMove(Location(pointx[j],pointy[j]))
                view_withmouse = capture(Init_view[j].getRect())
                click(Location(pointx[j],pointy[j]))
                #nothing should happen in all modes in all views

                assert exists(view_withmouse)

                #single right click
                rightClick(Location(pointx[j],pointy[j]))
                #nothing should happen in all modes in all views
                assert exists(Pattern(view_withmouse))

                #double right click

                mouseDown(Button.RIGHT)
                mouseUp(Button.RIGHT)
                mouseDown(Button.RIGHT)
                mouseUp(Button.RIGHT)
                #nothing should happen in all modes in all views
                assert exists(Pattern(view_withmouse))
```

Figure 3: Sikuli Script for Gofigure2 GUI Testing

4.2 Unit Test Script Format

Unit testing scripts are written as a series of functions defined by pythons keyword def. The constructing and destructing methods, *setUp*() and *tearDown*(), are used to start and end the software GUI and methods in between are named with the prefix test.

```
def setUp(self):
  print "setUp";

def testPrint(self):
  print "dummy test to validate, it is printing on the screen!";

def tearDown(self):
  print "tearDown";
```

When this script is run with the Sikuli Unit Testing feature, it will report the results of each test method in CTest and then reported automatically on CDash.

5 Current Limitations

Sikuli's integration with CTest has provided us with numerous advantages in the measure of productivity and convenience. But our goals are still hindered by a few limitations.

5.1 Off-screen Rendering

Our goal in using CTest is that we will be able to test our software automatically every night and bugs can be reported the next day on CDash. But Sikuli is not easily compatible with off-screen rendering. Sikuli requires that the screen remains on so that the script can identify the images on the screen. In addition, external mouse and keyboard actions can disrupt the script at any moment. These factors make it more difficult to set up automated nightly tests.

5.2 Variation of style between machines

Another important issue is the difference between operating systems. Developers need to make sure that their software runs on each OS. But Different operating systems require different libraries for generating GUI images. Therefore fonts, icons, and even layouts will differ between machines. This is also true between versions of the same OS. So, GUI tests that pass on one machine will inevitably fail on another machine for the same exact software. This requires developers to write the same Sikuli script using images from each OS, which will be many times the workload.

5.3 CDash

Furthermore, CDash does not currently support Sikuli script since images used in the script are unable to be displayed on the dashboard. Without these images on CDash, visualize which units are being tested. In addition, CDash does not make the distinction between simple syntax errors versus a complete test failure. These are extensions to CDash that can be corrected for future work.

6 Conclusion

In this paper we discussed the uses of Sikuli unit testing and how they can be integrated into CTest. Although we have recognized limitations within the current infrastructure (Sikuli and CDash), there is great potential to make these technologies more universal thanks to the fast growing Sikuli community. The results of our increased productivity have proven this technology well worth the effort.

Due to the previous exposed limitations (see section 5), provided examples may not run on your machine. However, it is easy enough to adapt these tests to your system.

Note

As of now, Sikuli has been successfully integrated with CTest on linux, and it requires some more work for Windows and Mac. We would welcome contributions to make it happen!

References

[1] Gofigure2 website. http://gofigure2.sourceforge.net/. (document), 1, 3

[2] Junit website. http://www.junit.org/. 4.1

[3] sikuli documentation. http://sikuli.org/docx/. 1, 2.1

[4] sikuli website. http://sikuli.org/. (document), 1, 2.1

[5] Tom Yeh, Tsung-Hsiang Chang, and Robert 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. 1, 2.1