vmware®

Loong: General Solution to UI Automation

TECHNICAL REPORT

TR-2013-002E

Yingjun Li , Nagappan Alagappan

Abstract

We have two different solutions for UI automation. First one is based on accessibility technology, such as LDTP [1]. Second one is based on image comparison technology such as Sikuli [2].

Both of them have serious shortcomings. Accessibility technology cannot recognize and operate all UI controls. Image comparison technology contains too many flaws and hard-coded factors which make it not robust or adequate for UI automation.

The principles of the two technologies are so different with each other. This means it is possible that we use accessibility technology to overcome shortcomings of image comparison technology and vice versa.

In this paper, we integrate accessibility technology with image comparison technology at the API level. I use LDTP and Sikuli to demonstrate the integration. Firstly, our integration overcomes respective shortcomings of the two technologies; Secondly, the integration provides new automation features. The integration is named Loong. It is a general solution to UI automation because it not only solves problems but also provides new automation features to meet various requirements from different teams.

1. Introduction

I use LDTP to represent LDTP (Linux), Cobra (Windows) and PyATOM (Mac OS X) because they are all based on accessibility technology and provide same APIs. If UI controls are standard - provided by operating system and have accessibility enabled), accessibility technology such as LDTP is a good choice to recognize and operate them.

However, LDTP cannot recognize and operate controls which do not have accessibility enabled. Controls may have accessibility enabled but they have been customized by developers. LDTP has difficulty operating such controls. In addition, sometimes the verification point is in guest machine and without the agent installed on the guest machine, LDTP is not useful in this situation because screen of guest machine is just an image to LDTP on host machine. Briefly, LDTP cannot handle all controls on screen. This is a general problem to automation based on accessibility technology.

Sikuli is an automation tool which is based on image comparison technology. Typically, Sikuli uses pre-captured images to match wanted controls on part or entire screen. If Sikuli successfully finds a control which matches a pre-captured image, Sikuli can operate it such as clicking. Sikuli does not care if accessibility is enabled or not on controls. Any control is just an image to Sikuli.

On the surface, Sikuli is simple and powerful. However, Sikuli is still not robust or adequate for UI automation. If multiple controls are similar in appearance, it will be difficult for Sikuli to distinguish the desired control from the rest. Sikuli would find and operate the wrong control due to such interference factors. We should have Sikuli to work on part of the screen which has excluded similar controls. However, Sikuli only has a hard-coded way of directly inputting specific screen coordinates or coordinate offsets to do this. That is to say, scripters estimate and input specific value of (x, y, width, height) such as (100, 200, 300, 400) to Sikuli by themselves. This is the region for Sikuli to work on. This hard-coded way would decrease the stability and accuracy of UI automation. This is a general problem to automation based on image comparison technology.

As above discussion, both the two technologies have respective shortcomings that obstruct their application in UI automaton.

After some investigation and research, we were able to successfully integrate LDTP with Sikuli at the API level. We use LDTP to calculate and provide the proper region of screen for Sikuli to work on and we use Sikuli to operate the controls which are unrecognizable to LDTP. This solution is named Loong. It not only overcomes respective shortcomings of the two technologies and improves the stability and accuracy of UI automation, but also provide new automation features to meet various requirements from different teams.

2. Approach

2.1 Runtime image

The traditional image comparison technology requires us to capture a lot of images firstly. It is a big effort. Sikuli includes an OCR module which is used to handle texts. However, OCR is very unstable to use. This section will introduce a new technology: runtime image. We will automatically generate images for texts at runtime and no need to capture images for texts any more. This technology will save much effort, and it can also be considered as a replacement for OCR.

Software contains both text and graphs on its UI. Generally text is much more than pure graphs (I mean, the graphs which don't contain text). Using Workstation as an example, only several icons on tool bar are pure graphs, no text on them. Except that, almost all other controls have texts on them. So if we can generate high quality image from text in runtime environment, which exactly matches the wanted text on the control of UI, we will solve most of the problems. The image is generated and used in runtime environment in every operating system automation scripts run, so it does not harm the cross platform of automation scripts. And it is generated automatically; scripters only need to provide the wanted texts and necessary parameters, so scripters are liberated from capturing and saving various images. We use the generated image not OCR to match the text and provide the position of the text for further operating. So OCR is discarded and we have a better replacement for it.

The key point is generating the image of high quality which can exactly match the wanted text on UI of tested software. That is to say, the image should be good enough to distinguish tiny differences from multiple similar texts. For example, three texts of "Snapshot1", "Snapshot2" and "Snapshot3" appear on one dialog. They are pretty similar in appearance except the difference of "1", "2" and "3" at the end of each text. Generated images should be capable of distinguishing these texts accurately and stably. Similarity (image based, a value between 0 and 1) describes how similar the two matching images are. 1 means the two images are exactly identical pixel by pixel. 0 means no pixel is identical between the two images. Low similarity would cover up tiny differences between the images, and this will lead to wrong control recognition and operation in UI automation. So the higher similarity we make, the better results we will get. High similarity between the generated image and the wanted text (the text is regarded as image to us) on UI can resist jamming (the highly similar texts) in UI automation. Experimental data indicates that similarity should be between 0.98 ~ 1 in order to ensure reliability and stability even in extreme circumstances of UI automation.

2.1.1 Gather information

In order to meet the demands of this key point, we consider and collect six primary properties of text appearance in software and modern operating system:

- Font
- Font style
- Font size
- DPI
- Anti-alias
- Hinting

These properties decide the appearance of a text. We obtain font, font style and font size information from operating system or just guess. Software does not use rare and tricky fonts on UI. With some investigation, we found most of them usually use Tahoma, Segoe UI, Sans Serif and Ubuntu (Ubuntu operating system only). Font size is between 10 and 15. Font styles are Regular, Bold and Italic. It is very easy to obtain correct font, font style and font size information. Even guess and enumeration are pretty enough to collect them.

DPI, anti-alias and hinting belongs to text rendering [3] when drawing text. They are especially important to generate image from text with high enough quality.

DPI stands for dots per inch. DPI is a measurement of monitor or printer resolution that defines how many dots of ink are placed when the image is displayed or printed. Operating system usually uses DPIs of 72, 96, 120 and 144. Some software inherits this value from operating system, some has its own. We acquire this information from operating system or tested software.

Anti-alias is a technology of blending bitmap-based images and text to reduce the stair-stepping or jagged appearance. In areas of transition, the edge pixels are blended to give a smoother appearance. As Table 1 shows, different operating systems or software have different anti-alias algorithms and settings.

ANTI-ALIAS SETTINGS	ALGORITHM DESCRIPTION
"off" or "false"	Disable font smoothing.
"on"	Gnome Best shapes/Best contrast (no equivalent Windows setting).
"gasp"	Windows "Standard" font smoothing (no equivalent Gnome setting). It means using the font's built-in hinting instructions only.
"lcd" or "lcd_hrgb"	Gnome "sub-pixel smoothing" and Windows "ClearType".
"lcd_hbgr"	Alternative "lcd" setting.
"lcd_vrgb"	Alternative "lcd" setting.
"lcd_vbgr"	Alternative "lcd" setting.

Table 1. Anti-alias algorithms

This information is obtained from desktop appearance settings (operating system) or tested software. Some software inherits anti-alias settings from operating system, some has own. If the tested software has own anti-alias settings, we obtain it from source code of the software or enumerate from above table.

Hinting is the use of mathematical instructions to adjust the display of a font outline so that it lines up with a rasterized grid. For the purpose of text display on screen, font hinting designates which primary pixels are interpolated to more clearly render a font. Hinting is usually created in font editor during the typeface design process and embedded in the font. Some operating systems have the capability to set hinting levels (none, slight, medium and full). We also get the information of hinting levels from desktop appearance settings of operating system.

2.1.2 Prepare system

Now we need to validate all the collected information to make sure it is enough exact. First of all, we set the text state machine in our system of automation based on runtime image:

settext(<font>, <font style>, <font size>, <dpi>, <anti-alias>, <hinting>)

If the property information of text appearance is changed on some texts, we only need to reset the text state machine.

Next we try to generate an image and compare it with the wanted text on UI. If similarity is high enough such as between 0.98 ~ 1, this indicates we provided exact property information to our system. Otherwise, we will need to provide property information by settext() and validate it until the similarity is enough high (0.98 ~ 1). Usually, preparing the system is quick and easy because the exact property information is not difficult to acquire.

Possible values of every property are in a finite set. Another option is enumerating and combining these values in backend of the system, then generating images by the possible property combinations and calculating similarity in multi-threading. It is easy to optimize and decrease possible combinations in different operation systems. There should be at least one image which has high enough similarity (0.98 ~ 1). We use this image to match wanted text on UI of tested software. In this implementation, we do not need text state machine and step of system preparation.

2.1.3 Fire up

The system is now ready to use. We generate the high quality image from text in runtime environment of operating system. Use clicking operation as example, click(<region of part or entire UI>, "Snapshot2") will generate an image which contains the text of "Snapshot2" in runtime. We have provided the exact property information of wanted text appearance of "Snapshot2" by settext(). The image is generated by the information, so it can exactly match the wanted text "Snapshot2" on UI of tested software. Then if the text of "Snapshot2" is found accurately, a click operation will be performed on it next. Usually a control has text on it. The text is found and operated by the generated image, so the relevant control is found and operated.

The results of the experiment demonstrate excellent reliability and stability performance of our system design of automation based on runtime image. We used Sikuli as baseline to design and prototype the system of automation based on runtime image, and the image comparison algorithm we use is not relevant to the image color. The generated image has very high quality, it meets our demands. The similarity between the generated image and the wanted text on UI of tested software is 0.98 ~ 1. This is vital because it makes sure we find and operate correct control even when jamming (the highly similar texts) exists.

2.2 Architecture

This section will introduce the architecture of Loong. Sikuli has five primary private classes: Region, Screen, Location, Match and Pattern. Region class describes a region of screen where Sikuli search and operate controls. It has four parameters: coordinates (x, y, width, height). Screen class describes which monitor to search when computer has multiple monitors. Location class describes a location of a control. It has two parameters: coordinates (x, y). Match class describes the control Sikuli found. Pattern class describes the control Sikuli needs to find and operate. It has two parameters: string and similarity. String can be several words or a path to an image that Sikuli needs to find and operate. Similarity is a threshold that Sikuli uses to find correct controls. It is not possible for LDTP to access these classes directly, so Loong builds a bridge between LDTP and Sikuli. LDTP and Sikuli use Loong to communicate with each other as Figure 1 shows.

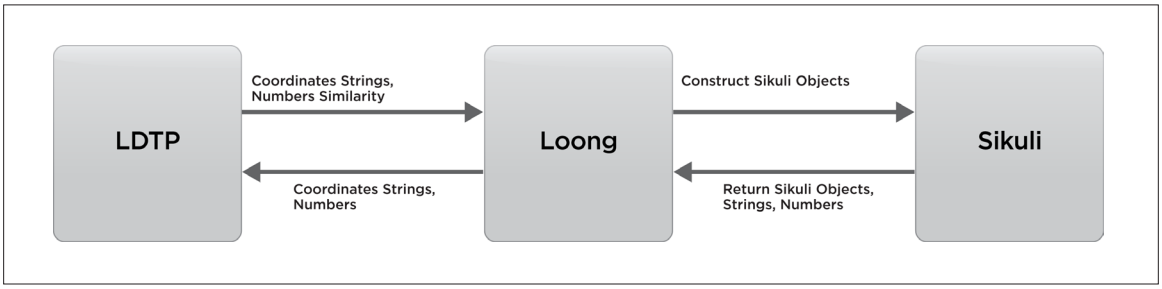

Figure 1. Loong architecture

2.3 Implementation

2.3.1 Loong acquires information from LDTP

If we can flexibly specify a small region which excludes similar controls to Sikuli, Sikuli will work stably, accurately and fast. LDTP has two APIs: getobjectsize(<window name>, <object name>) and getwindowsize(<window name>). Both of them are implemented respectively by underlying accessibility libraries on Windows, Linux and Mac.

getwindowsize() returns coordinates (x, y, width, height) of the specified window or dialog. getobjectsize() returns coordinates (x, y, width, height) of the control on a window or dialog. Usually, not all UI controls have not accessibility enabled, so we can use these two APIs to calculate the coordinates (x, y, width, height) of the wanted small region on the window or dialog. Wanted small region refers to the region that contains non-accessibility enabled controls and/or non-standard controls Sikuli needs to find and operate, but that does not contain interference factors (the controls too similar in appearance).

This coordinates information is calculated and provided by accessibility technology in runtime environment. This is what accessibility technology benefits us. It is much better than hard- coded method used in current image comparison technology. Sikuli needs coordinates information and Loong acquires it. Coordinates (x, y, width, height) represent a region. Coordinates (only x and y) represent a location of a control. Besides coordinates, Loong also accepts strings, similarity (a value between 0 and 1) and numbers for use.

2.3.2 Loong constructs Sikuli objects, passes these objects and other information to Sikuli

Loong constructs Sikuli private objects such as Region, Screen, Location and Pattern by these coordinates information acquired from LDTP and other information such as strings, similarity and numbers. These objects are requisite for Sikuli to work. In detail, Loong uses coordinates (x, y, width, height) to construct Sikuli Region object; uses coordinates (only x and y) to construct Sikuli Location object; uses strings and similarity to construct

Sikuli Pattern object; uses numbers to construct Sikuli Screen object. Next, Loong passes these objects and other information to various and useful Sikuli APIs. Sikuli APIs will use these objects and information to find and operate control in the wanted small region represented by coordinates (x, y, width, height) we calculated and provided.

2.3.3 Loong processes returned Sikuli objects and other values, passes the information to LDTP

When Sikuli finds and operates a control, it may also return it as a Match, Region or Location object. Loong will process these objects and acquire coordinates (x, y, width, height) or (only x and y) from them. Then Loong will pass the coordinates of the control to LDTP for use. If Sikuli only returns simple values such as numbers, strings to Loong, Loong will pass these values to LDTP directly since LDTP can access these values well.

2.3.4 Loong is C/S structure and uses XML-RPC protocol

Currently we implement server side of Loong by Jython and respectively implement multiple language versions (Python, Java, C# and Ruby) of client side. Server side and client side of Loong communicate by XML-RPC [3] protocol. Server side primarily contains three functionalities:

1. Use necessary information to construct Sikuli private objects.
2. Acquire coordinates of controls and other information from returned Sikuli private objects and other values.
3. Encapsulate and provide Loong APIs to scripters.

2.4 Loong APIs

Loong APIs acquire coordinates and other information which are described in section 2.2.1; then Loong server constructs Sikuli private objects. After this, Loong APIs pass these objects and other information to relevant Sikuli APIs and call them. Sikuli APIs are executed; they find and operate controls, next return information (Sikuli objects, strings, numbers, etc.). Loong APIs process the information and pass them to LDTP for use.

Our system provides about 40 Loong APIs for scripters to use. Several key functions are described below.

find(<region>, <object>)
Description: Find the best matched control in the specified region.

findall(<region>, <object>)
Description: Find all matched controls in the specified region.

wait(<region>, <object>)
Description: Wait for the matched control to appear in the specified region.

waitvanish(<region>, <object>)
Description: Wait for the matched control to vanish in the specified region.

click(<region>, <object>)
Description: Click the matched control in the specified region.

doubleclick(<region>, <object>)
Description: Double click the matched control in the specified region.

rightclick(<region>, <object>)
Description: Right click the matched control in the specified region.

input(<region>, [<coordinate>], <text>)
Description: Input the text at the current focused input field or at the click point specified by <coordinate>.

highlight(<region>, <timeout>)
Description: Highlight the specified region for some time.

2.5 Loong features

Below are new automation features Loong provides:

1. As our integration is at the API level, LDTP APIs can talk with Sikuli APIs freely in one script. This is very convenient and powerful to scripters.
2. We use XML-RPC protocol to communicate between server and client of Loong, so Loong supports distributed testing, that is, run tests which consist of several parts executed on different machines and interacting with each other. This is very helpful to guest/host interaction.
3. Currently we implement Python, Java, C# and Ruby clients of Loong, so scripters can choose between these languages (C# client also supports Power Shell and VB.NET languages) to write automation scripts.

   We have a good scalability in programming language of automation scripting. If scripters want to use other languages to write scripts besides above, we just need to implement the client by the language scripters want to use.
4. No bad influence to existing automation scripts.
5. Support Windows, Linux and Mac.

3. Conclusion and future work

Firstly, the solution uses getobjectsize() and getwindowsize() to calculate and provide coordinates information for the wanted small region in Sikuli. We use this way to replace hard-coded way in Sikuli. getobjectsize() and getwindowsize() are implemented by access-ibility technology, so accessibility technology solves the general problem of image comparison technology. We use Sikuli to find and operate the controls which are unrecognizable to LDTP, so image comparison technology solves the general problem of accessibility technology. The solution improves stability and accuracy of UI automation. This paper names it Loong.

Besides the problem solving and improvement of stability and accuracy, Loong also provides five new automation features described in section 2.4. Because of these new automation features, Loong meets various automation requirements from different teams such as View, Workstation and Fusion. Loong is extremely valuable for any tests from UI; it is a general solution to UI automation.

In the future, we plan to introduce runtime image technology which is described in detail in another paper of "Automation Based on Runtime Image" into Loong, which will make Loong more effective for UI automation. We believe that Loong will be the best solution to UI automation after we complete this work.

4. Acknowledgments

We would like to acknowledge the help of Workstation team, who provided much help to test Loong. And we are also grateful to John Green, who reviewed the paper very carefully.

5. REFERENCES

[1] LDTP Project. http://ldtp.freedesktop.org/

[2] Sikuli Project. http://www.sikuli.org/

[3] XML-RPC Protocol. http://en.wikipedia.org/wiki/XML-RPC

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com

 Item No: VMW-TWP-LOONG-GENERAL-SOLUTION-TO-UI-AUTOMATION-USLET-102 08/13