iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 2 **Issue:** VI **Month of publication:** June 2014

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Server side image string patterns processing

Komal Arora[1], Sanjla Sindhwani[2]

[1]Assistant Professor, SPGOI College of engineering, MD University, Rohtak (India)

[2]M.Tech Student, SPGOI, College of engineering, MD University, Rohtak (India)

Abstract*— To make the string search more reliable, the images of the string, used as a basis by the search algorithms should be produced by the same font rendering engine, as is used by the strings in the application under test. So, we need an algorithm which takes over the rendering and produces the output using System font rendering and font smoothing. So, a service needs to be developed which can be deployed on the same machine on which the application under test is running and it provides those images at run time using the font renderer provided by operating system of the system under test.So now for searching of string, image rendering will happen at the system on which the application under test is running and will make use of rendering engine provide by the operating system of that machine, in contrast to the earlier scenario where AWT rendering was happening at the machine where AUTO-UI gets installed.

1. INTRODUCTION

Commercial software product now a days demands high on quality and reliability. Therefore every organization makes ample efforts for delivering quality product to customers and puts lots of efforts in testing the product to ensure quality of the product to be delivered. Different quality teams for the product makes use of multiple type of testing infrastructure and tools to make sure delivery of high quality products. Quality engineers work in co-ordination with developers to define different test strategies can be integrated with some test frameworks which ensures certain quality standards such as variable assignments, memory leaks, code coverage, unit level function testing. Quality engineers will have access to the application code and they will write unit test to verify application's functionality at unit level. As the stability of the application under test increases, these can be very well integrated with automated build system and automation execution. Overview reports can be analysed for each of the builds. So if a functionality break in a particular build, it will be caught there itself. The second category of test, so called black box testing, is a different approach which instead of bringing the extra testing code in the application, the application is tested from the user's perspective. The interface to the application essentially consist of mouse and keyboard input, and output on the screen. The steps to be performed are formulated in form of test cases and the result of the same can be evaluated on screen.

2. PROBLEM ANALYSIS

As already described in the introduction, the problem is that the string search is unreliable and doesn't operate uniformly across all platforms. String search fails many times, with no specified reasons. Initially, it was the assumption that the search is not tolerant enough while comparing pixels.

To examine this suspicion, the existing AutoUI code was debugged and the current implementation of the algorithm was analyzed step by step. From the debugging and analysis, it emerged out that the pixel comparison was working correctly and the correct tolerance was taken into account while comparing pixels. So there was no issue during comparison of pixel values. But with further investigations it was noticed that the color values for many pixels in the screenshot was different from the one generated for the string using the AWT rendering engine. In order to confirm this suspicion, AutoUI code was extended so that the AWT generated image which was used to search the relevant portion of the image and the screenshot can be saved to hard disk.

Therefore, there must be other factors that are not taken into account while creating the image using current rendering process. Rending of the string, identical to the image from the screenshot is necessary, as the comparison is happening on pixel by pixel basis. So, these unknown factors needs to be

uncovered and incorporated while rending the image from string.

Every operating system has an inbuilt software component which is used to display text. It is called as “Font Engine or Font rendering Engine”. It is the assumption that the Font rendering Engine of each operating System uses different rendering algorithm and thus produces a different result. To realize this implementation there are several technologies. We will make use of Java Servlets for our implementation, as AutoUI is implemented in Core java, so it will reduce the effort of learning and maintenance, if the implementation is being done using Java Servlets. Hereafter, the Web service will be referred to as “**Font rendering service**”.

Depicts the generic overview of the infrastructure to be implemented and Fig depicts the components of each VM, in which the Application under test will be running and the Font renderer will be deployed on each VM. After these two fundamental elements have been defined, there comes the need to communicate between these.

In a first prototype implementation, there was some shared directory on a network, which contains folders for each of the supported OS, which in turn has the font rendering service order files, in the form of XML documents. Because each request against the font rendering service starts a new thread instance of the servlet, a unique thread ID is associated with the order file. This order file is then read on the VM by the respective font renderer, which in turn renders the string described in it.

The rendering result is finally saved as an image in the network directory, which in turn is again uniquely identified by the Thread ID. The Font Renderer Service then picks the image from the network directory and delivers the result to the requesting client.

The realization of the main features of the software components that were created in the context of this work is described below. Since socket communication and build scripts for all the components have played a significant role, each has been entered into its own section.

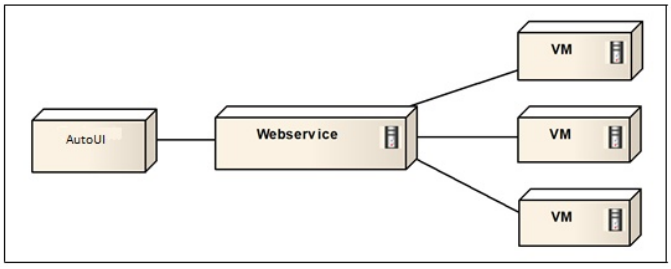
FontRenderer

The task of the “Font Renderers”, which was realized as an AIR application in Flex, is the **rendering of a string**. Flex has a variety of components, which represent texts; the simplest can be being a Label. The setStyle() method for a label can be used to set the style properties of the text. This method accepts a key-value pair as input and can set the values for the properties which are represented by the key passed as parameter to the method. Using this we can set properties like “font Weight”, “font Family”, “font Style” , “ font Size” etc

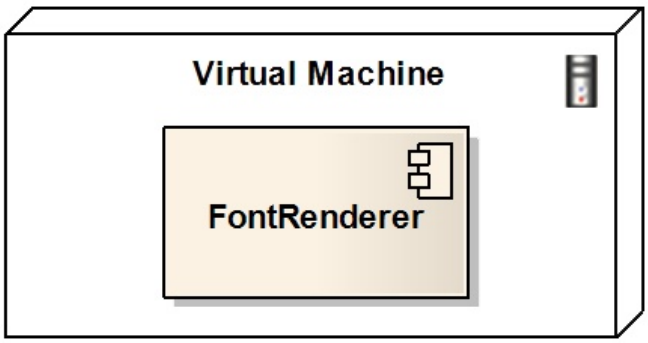
and the real characteristics of text, as discussed in analysis part, can be implemented. We have chosen “font Family” to identify whether a font exist on System or not. If the specified font exists on the System, then Flex uses “System font rendering engine” of the machine, on which application is running, to display text, otherwise Flex embedded Rendering engine is used to render and display the text. The property "font Size" still a needs some consideration. Flex renders all text with 72dpi, which was consistent with the behavior of the system font rendering engine of Mac systems. On Windows systems, 96 dpi is assumed by default. This corresponds to that a 9pt font on Windows XP will correspond to 12pt-font (96dpi/ 72dpi * 9pt = 12pt) on Mac. So the rendering service should have this logic inbuilt to take care of this logic.

which implement the interface **IBitmapDrawable**, to be converted directly to a bitmap. Such an object is the label which can be used to render the string. Since the image is to be sent over a number of socket connections, the use of image compression is recommended.

To make sure that the quality of the image is not changed, it must be compressed to a lossless image format, such as PNG Format. In Flex “PNGEncoder-Class” is available for this purpose. Unfortunately, it is not possible in Flex to draw a UI component and set its style properties directly by invoking a function call. This is due to the single threaded architecture of Flex. It just works on based of events. So when user calls **setStyle**() method on the label to convert it to a bitmap, it merely fires an event.



Overview of the structure of the backend



Details of contents of VM

This event will be processed only when the component gets the CPU cycle next time. The priority of updating the display of a component can be maximized by calling **invalidateDisplayList**(), so that it is ensured that they will be updated directly after the currently working method is finished.

3.CONCLUSIONS

Overall, the achieved result is a significant improvement over the old string search. As a by-product, a font rendering backend is developed, which provides a very simple and universal web service interface and could be used in any products which needs to render images from given strings. It provides significant improvement, e.g., the self-imposed ceiling of a maximum of one second response time for the rendering of the average long strings could be significantly undercut to 80-200ms. This has also reduced the maintenance efforts of the test scripts, as there is no need to maintain images for each of the supporting SUT, as was the case earlier.

Unfortunately, the development in some places lasted longer than planned. For example, functionalities like embedding fonts rendering and the event-based socket communication in Flex expanded beyond the planned time.

REFERENCES

1. Tristan Richardson,Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Vietual network computing .IEEE internet computing pages 33-38 1998.
2. Richard P. Martin. Heisenbugs and Bohrbugs. Rutgers University, 2003. http://www.cs.rutgers.edu/~rmartin/teaching/spring03/cs553/papers01/06.pdf
3. Susan Haigh. Optical character recognition as digitization technology.Network Notes #37, 1996
4. John Zukowski. Java AWT Reference. O'Reilly, 1997. http://www.oreilly.com/catalog/javawt/book/
5. Nabeel Al-Shamma. Font Smoothing in Flash Player, Mai 2007. http://blogs.adobe.com/alshamma/2007/05/font_smooth_in_flash_player_1.html
6. Adobe Systems. Adobe Flex 3, June 2009. http://www.adobe.com/products/flex/
7. Prof. Dr. Stefan Fischer. Verteilte Systeme. TU Braunschweig, 2002 http://www.ibr.cs.tu-bs.de/courses/ws0203/vs/PDF/VS-0203-Kap01-Einfuehrung- c 1S.pdf
8. Cornel Creanga. Bringing data into Flex applications, November 2008. http://www.cornelcreanga.com/2008/11/
9. VMware Inc. VMware Fusion, 2009. http://www.vmware.com/de/products/fusion/
10. Field of Experts for Image Based Rendering

    http://nguyendangbinh.org/Proceedings/BMVC/2006/papers/363.pdf
11. RFB Protocols

    http://en.wikipedia.org/wiki/RFB_protocol
12. The RFB Protocols, Real VNC [Tristan Richardson]

    http://www.realvnc.com/docs/rfbproto.pdf

13. Image Comparisons in Java

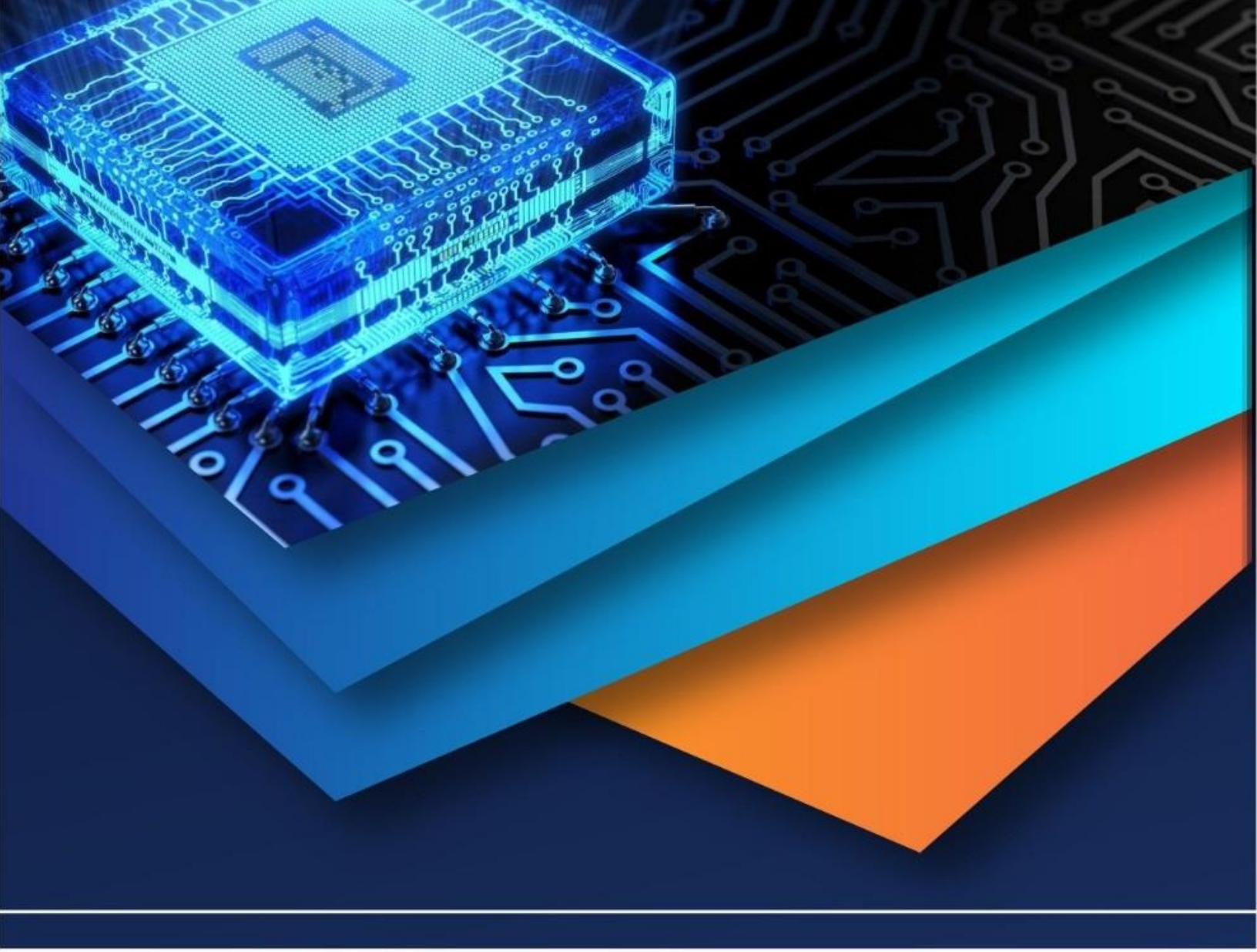
http://manthapavankumar.wordpress.com/2012/11/25/comparing-two-images-in-java/

http://sahits.ch/blog/?p=587

14. Separating Background and Foreground pixels in an image

http://stackoverflow.com/questions/14423615/seperating-background-and-foreground

Image Rendering on Windows



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY