M.R. Roboto:

Macro Record Robot Language
White Paper

Authors:

Adam Marzryk (alm2126@columbia.edu)
Hema Krishnan (hk2230@columbia.edu)
Jason Kopylec (jkk2106@columbia.edu)
Sanjit Tewari (sot5@columbia.edu)

Introduction

Macro Record Robot Language (M.R. Roboto) places in the hands of the programmer a powerful tool for controlling GUI based operating systems. By simulating the actions of a user, one can test interactive applications and automate repetitive tasks with ease. M.R. Roboto provides a simple script-style interface, while leveraging the strengths of the Java programming environment. This paper proposes a language that:

1. Contains a simple interface where users can easily create scripts that mimic keyboard and mouse input to automate repetitive tasks
2. Serves to model virtual user input using common computer vision, artificial intelligence and machine learning techniques
3. Interfaces with Java to utilize the rich language libraries without restricting the applications to be Java-specific

Background

The first programs to allow real-time user interaction came shortly after the invention of the Video Display Terminal in the mid-1960's which incorporated a television style monitor with an electric typewriter [Bellis]. In the years following, real-time interactive operating systems were being developed, culminating in the introduction of UNIX in 1970 [Milo]. Along with the ability to interact with the user via the command line came tools for automating user input. Shell scripting and IO redirection provide the groundwork for application testing and automation of repetitive tasks and have become invaluable programmer tools.

The Graphical User Interface (GUI) has become a staple of the modern, mainstream operating system since its inception in the 1979 Macintosh [Tuck]. With the proliferation of personal computing and the World Wide Web, user interaction based on keyboard and mouse (or similar pointing devices) input is a main function of modern operating systems. Unfortunately, there is a lack of simple scripting tools for automating this type of interaction. There is no widely used batch/script type language to easily automate user functions as with command line operating systems. Usually knowledge of large and complex operating system API is required to manipulate individual screen elements which are bulky and counterintuitive. User interaction is much easier when thought of in terms of mouse and keyboard commands.

Java provides an interface, albeit a clumsy one, for modeling and performing user input in a more natural way. By streamlining the Java model and providing a simple script-style interface, programmers would have an ideal tool for testing their interactive graphical applications and be able to automate tasks in a way analogous to UNIX shell scripts or MSDOS batch files.

Existing Technologies

Current programming languages, applications and operating systems do provide mechanisms for manipulating the user desktop and applications, but each comes with inherent restrictions that M.R. Roboto serves to correct. To outline a few examples:

Microsoft Windows API

Microsoft has built into its latest generation operating systems (Win 2k, XP) a multitude of API to control and handle every aspect of the kernel and windowing system. Windows allows scripts and applications to manipulate individual components of the user desktop.

The first problem with this API is its sheer size. The Windows API supports literally thousands of procedures to control the desktop environment. It is quite easy to get overwhelmed and lost in the Microsoft Developers Network website which holds the OS and language specifications [MSDN.com].

Secondly, these APIs do not reflect accurately the actions of interactive users. For example, to create a script that closes the currently open window, the script would require you to know the name of that window, get a handle to it and then evoke a method that closes the window and kills the process. What a user would do is simply move their mouse over the “X” in the corner and click.

Traditional Batch Files

The traditional batch file is comprised of lists of commands that can are executed sequentially (with some limited flow control) as if the user had typed them one after another. This is powerful because it is easy to automate repetitive tasks and the script can use the same language as the user to communicate with the operating system.

The problem with traditional batch files is that they do not support multiple windows or mouse actions. The only way that batch files can send input to the applications they execute is through the command line or by redirecting data from standard input. This limits mouse-driven applications or programs with specialized, non-sequential input controls, such as web pages.

JUNIT

As large scale open source project, the sole focus of JUNIT is to automate application and code testing. In particular, the module JFCUnit was built to test graphical applications by providing input from a virtual test user. It serves to “start the application and then interact with it, typing in some text and maybe pressing some buttons.” [Hammell]

JUNIT supplies the Java applications programmer with a plethora of tools for running tests on code, but this is also its limitation. It was developed to test only Java applications, and runs in the same thread as the application, so if the application fails, so does the user interactivity [Clark]. This does not at all supply portability to general problems of testing application, modeling user artificial intelligence, or automating regular windowing tasks.

Stand Alone Applications

Searching for "Windows Macro" within *Download.com* displays a number of applications such as Macro Express, Workspace Macro and Phantom Sidekick to name a few. The main goal of these applications is to automate repetitive tasks. Often this is done by pressing a "record" key and then typing and clicking the procedure and pressing "stop" when completed.

These applications have their share of problems though. The first is the representation of the macro once it has been recorded. Some applications represent the macro in terms of Windows API, which suffers from the same hindrances of the API itself, namely its huge size and not modeling user input as a sequence of keystrokes and mouse actions.

Applications that do not translate into Windows API either compile directly into machine code or a proprietary intermediary. This is problematic because if a small part of a large repetitive task changes, then the entire macro must be recorded again from the beginning.

In addition to these limitations, stand-alone apps are more difficult, if not impossible to utilize as virtual user interfaces to generate input on the fly, or incorporate into larger Java applications.

Java.awt.Robot

Java API provides a class for manipulating the user interface in a much more user oriented fashion. It includes methods for moving the mouse to a specific coordinate on the screen, mouse clicks, keyboard input and some rudimentary methods for retrieving the screen contents ["Robot"]. The java.awt.Robot class can work either within or outside the scope of java applications. The Java Robot requires no user knowledge of the underlying operating system specifics or API.

The downside to the Java implementation is that is a bit clumsy. There are a number of places where in addition to needing to know about Object-Oriented and basic Java programming, there are special constants and API necessary for forming the expected output results. For example, each keyboard character must be pressed and released in separate calls and is indexed by a special constant [Baldwin]. This is quite a hassle if you want to automatically type the text "http://www.google.com" into a web browser.

Putting the Pieces Together: M.R. Roboto

Macro Record Robot Language (M.R. Roboto) is a programming language that leverages the assets from the following sources:

1. The ease of making batch files
2. The power of Java (and particularly the Java.awt.Robot class)
3. User ability to interact with environment via keyboard, mouse and monitor

Each of these tools on its own has limitations, but by modeling a language that puts them together, a complete array of applications both trivial and complex can be developed with flexibility and ease. The audience for this language spans from the casual user who wants to check email faster without a lot of typing and clicking to the machine learning or computer vision researcher who wants to model user interaction with Windows to programmers who want automate the testing of their GUIs.

What it Looks Like – Example 1

A simple application that automates a repeated user function would typically look like a number of mouse clicks and keystrokes. For example, let's say we wanted to write a script that checks Yahoo! email. Code would probably look something like the following:

`CheckYahooMail.mrr`

```
click(20, 755)          //click Internet Explorer icon
wait(10000)             //wait 10,000ms (10 sec) for the page to load
type("username")        //type Yahoo mail username
type("\tab mypassword") //Hit tab-key and my Yahoo password
type("\tab \enter")     //Hit tab-key again and hit enter
```

After compiling and running the code, one should be taken directly to their Yahoo! Mail inbox.

Artificial Intelligence – Example 2

Applications that act as intelligent agents controlling a desktop can become quite complex, but we can also show a simple application that give their human counterparts some real competition.

As an example, lets create a script that plays a simple game that imitates a shooting range (see Fig. 1) ["Shooting"]. The user scores a point for each bull's-eye that is "shot" by pressing the mouse button on it. A point is deducted for every misfire. The difficult part of the game is that it contains virtual human movement, so the gun sight shifts back and forth and shooting causes recoil that loses aim as a human would. The user must try to shoot as fast as possible once the crosshairs are over the bull's-eye.

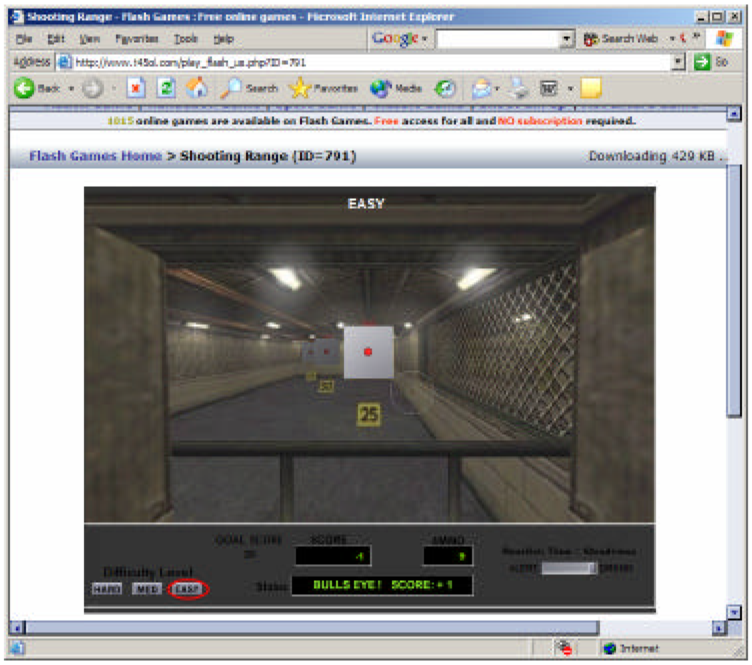

Shooting Range - Flash Games : Free online games - Microsoft Internet Explorer
Flash Games Home > Shooting Range (ID=791)
EASY
BULLS EYE! SCORE: +1

Fig. 1 Shooting Range Game.

We want to generate a program that plays the game with a decent level of skill which can be done with the following M.R. Roboto program:

ShooterGame.mrr

```
//Assume that the shooter game window has been opened and started
numShots=22         //User gets 22 shots at the target
bullI_X=350         //X screen coordinate, center of the bull's-eye
bullI_Y=500         //Y screen coordinate, center of bull's-eye

//Try to click on the bull's-eye for each shot
for i=1 to numShots
      click(bullI_X, bullI_Y)
      delay(2000) //2sec delay between shots to allow for recoil
next
```

We see here the program executes the same actions that an ideal user playing the game would. Someone new to the game could definitely increase their high scores by letting M.R. Roboto play for them.

Language Tools

To make the language more expressive than simply "click here, click here, type here, etc" a number of programming language primitives and structures will be incorporated to facilitate solving a broad array of programming problems. These include:

Data Types

Support for data types such as *integers* and *strings* will be included as basic programming tools. In addition, there will be primitives for dealing with display elements, such as retrieving a bitmap of the screen which can be compared to other bitmaps. This provides the groundwork for designing reactive components and run-time error checking.

Flow Control

Flow control structures such as *for-loops* and *if-then* statements will be in place to work with the primitive data types for conditional flow control. Procedures and functions with syntax similar to java will also be supported to facilitate program segmentation.

Screen Output

Taking feedback from the display opens up a complex set of computer vision issues that may not be solved in a project this size, but the aim is to incorporate some primitive methods for retrieving pixel colors and screen shots to illustrate the possibilities of using display feedback to drive automation and/or artificial intelligence applications.

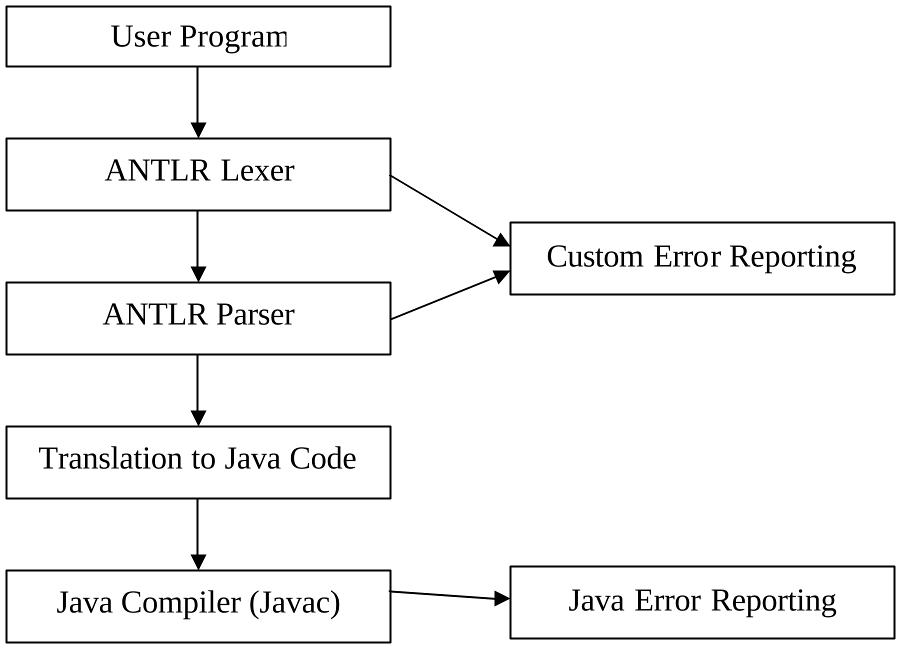
Imbedded Java

To utilize the far reaching power of Java without muddying the syntax of the language, native Java code can be placed within brackets anywhere in the code. This opens the door for advanced programmers to utilize this simple scripting language as a core part of larger Java development projects.

Compiler

The M.R. Roboto programs are compiled into native Java code that can then be compiled and run as normal Java programs. Many similar languages are interpreted, but by compiling into Java (and then into Java byte code) the scripts can be fully incorporated seamlessly into larger Java programs. The M.R. Roboto program is parsed and translated by a custom procedure developed using ANTLR and Java. See Figure 2 for the compilation process.

Fig. 2 M.R. Roboto Compilation Process



ANTLR Lexer/Parser

Providing custom programming tools requires defining a syntactic grammar and semantic rules. ANTLR provides the tools to easily create lexical analyzers, parsers and other compiler components based on defined grammars [Parr]. M.R. Roboto will use ANTLR to develop the front-end syntax checking required prior to the conversion to native Java. ANTLR will also be used to report custom errors back to the user about syntactic or semantic errors. Any native Java in the source code is ignored.

Translation to Native Java

Once the code is deemed "valid" by the ANTLR parser, a module will create a Java template class that initializes a Java.awt.Robot instance. The M.R. Roboto commands are translated into java code and procedures statement-by-statement. This class can then be compiled into a working java program. Any errors in Java from the source code would be indicated during compilation.

Summary

M.R. Roboto takes advantages of a script style interface while maintaining the full features of a large scale programming language like Java to provide what other tools have fallen short of in terms of providing a virtual user interface. The language can be used to easily model and solve a multitude of problems from process automation to artificial intelligence.

Bibliography

Baldwin, Richard. "Introduction to the Java Robot Class in Java" Developer.com 27 May 2003 <http://www.developer.com/java/other/print.php/2212401>

Bellis, Mary. "The History of the Computer Keyboard." About.com 2004 <http://inventors.about.com/library/inventors/blcomputer_keyboard.htm>

Clark, Mike. "JUnit FAQ." JUnit.org 23 Sept. 2004 <http://junit.sourceforge.net/doc/faq/faq.htm>

Hammell, Thomas. "Extreme Java GUI Testing." Developer.com 26 Apr. 2002 <http://www.developer.com/java/other/print.php/1016841>

Milo. "History of Operating Systems." OSData.com 26 Sept. 2000 < http://www.osdata.com/kind/history.htm>

MSDN.com 2004 <http://www.msdn.com>

Parr, Terrence. "ANTLR Reference Manual" ANTLR 9 May 2004 <http://www.antlr.org/doc/index.html>

"Robot (Java 2 Platform SE v1.4.2)" Sun.com <http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Robot.html>

"Shooting Range" Animated Game. Flash Games <http://www.t45ol.com/play_flash_us.php?ID=791>

Tuck, Mike. "The Real History of the GUI" SitePoint.com 13 Aug. 2001 <http://www.sitepoint.com/print/real-history-gui>

"Windows Macro." Website Search. Download.com 26 Sept. 2004 <http://www.download.com/3120-20-0.html?qt=windows+macro&tg=dl-2001>