Utilizando o Sikuli Script para a construção de Testes Automatizados.

Construindo Testes Unitários De Interface Gráfica Usando a Linguagem Java e as Bibliotecas do Sikuli Script e WebDriver.

Deuel Dias Lopes

deueldiaslopes@yahoo.com.br

Deuel Dias Lopes é atualmente graduando do curso de Sistemas de Informação na Faculdade Metodista Granbery sendo um entusiasta da área de teste de software e no momento trabalha como técnico de suporte.

Marco Aurélio P. Silva

marcoaureliops@outlook.com.br

Marco Aurélio é graduando e bolsista de iniciação cientifica do curso de Sistemas de Informação na Faculdade Metodista Granbery, estagiário de desenvolvimento de sistemas e entusiasta do software livre e do sistema linux.

De que se trata o artigo
O artigo abordará a utilização da API do Sikuli Script na construção de uma aplicação java na qual fará um teste funcional automatizado. Serão descritas também, algumas funções do Sikuli que serão utilizadas nos exemplos citados. Este mostrará as principais funções que o Sikuli Script pode oferecer iniciando uma abordagem teórica sobre os conceitos que compõem a tecnologia e posteriormente apresentando um exemplos prático.
Em que situação o tema é útil
Este artigo pode ser útil para aqueles que precisam encontrar uma solução livre na construção de testes automatizados para aplicações desktop, móveis e até mesmo aplicações Web reproduzindo ações que o Selenium não consegue como, por exemplo, a manipulação de imagens(Mapas, gráficos e etc).

Quando precisamos nos comunicar é muito comum recorrermos a referências visuais, principalmente quando é difícil descrever verbalmente alguma coisa, por exemplo, quando pedimos um salgado na padaria, em geral apontamos para o mesmo ao invés descreve-lo. Porém, quando interagimos com um computador várias de suas interfaces não são otimizadas para utilizar referências visuais na comunicação e nos vemos forçados a optar por outras alternativas. Um exemplo é o teste e automação de interfaces de usuário gráficas que em geral requerem a escrita de scripts que enviam comandos para determinados componentes da interface com o objetivo de controlá-los e verificar se os mesmos apresentam o comportamento esperado. Ao realizarmos a escrita desses scripts enfrentamos um grande desafio em como referenciar os componentes da interface que desejamos testar.

Uma solução seria utilizar os nomes dos componentes como referência, porém estas informações não costumam ser visíveis ao usuário pelo fato do código fonte da aplicação, em geral ser fechado e com acesso restrito. Outra opção seria utilizar as coordenadas (x,y) dos componentes na tela, porém devido ao fato de existirem várias resoluções diferentes sendo as mesmas podendo ser alteradas e também a possibilidade dos componentes trocarem de posição após o mapeamento das coordenadas tornam esta solução em determinadas situações inviável.

As duas soluções são exemplos de alternativas que não utilizam referências visuais e que nos forçam a aprender uma nova forma de operar um sistema em já estamos familiarizados e por consequência prejudicando sua usabilidade. Uma das maiores dificuldades encontradas na automação e teste de interfaces gráficas reside no fato da não existência de canais de comunicação e protocolos padrão para as interfaces gráficas das aplicações. Algumas aplicações bem construídas oferecem um leque de API’s ou seguem os padrões de acessibilidade dos sistemas operacionais no qual são executadas permitindo a comunicação com a interface através dessas API’s. Porém, na prática a maioria das aplicações não é construída dessa forma. Podemos considerar como o único elemento em comum dentre todas as interfaces gráficas das aplicações os pixels utilizados em sua construção.

Com o objetivo de resolver este problema o Sikuli Script foi desenvolvido pelos pesquisadores do MIT Tom Yeh, Tsung-Hsiang Chang e Robert C. Miller e posteriormente convertido em um projeto de código livre. O projeto pode ser considerado um sistema de script que permite que usuários utilizem como referência as imagens dos componentes da interface que desejam controlar. O sistema é baseado na linguagem de programação Python permitindo ao autor do script a ter acesso no momento de sua construção a todos os recursos que a linguagem tem a oferecer. Com o Sikuli Script podemos pedir ao computador que "Mova todos os documentos do Word para a lixeira" usando o comando dragDrop e as imagens dos ícones do Word e da lixeira respectivamente. No Sikuli Script as imagens são consideradas como objetos que podem ser vinculadas a variáveis, serem o retorno de uma função ou serem passadas como parâmetros. A utilização de imagens como referência proporciona vários benefícios sendo um deles a portabilidade. No caso da aplicação a ser testada possuir versões para diferentes sistemas operacionais bastará a criação de um único script e o mesmo funcionará independente da plataforma na qual a aplicação está sendo executada. Com o objetivo de facilitar a escrita do script os responsáveis pelo projeto construíram um editor chamado Sikuli IDE. O presente artigo não abordará a instalação e nem a utilização dessa ferramenta devido ao foco desse trabalho ser a utilização dos recursos do Sikuli Script.

Testando uma Aplicação Desktop

Nesta sessão demonstraremos um teste de uma aplicação java desktop realizado com auxilio do Sikuli Script.

Uma aplicação simples para destop pode ser testada sem problemas usando o Sikuli Script. Antes de mais nada precisamos preparar o ambiente para a execução dos testes, portanto vamos descrever os passos para a preparação do ambiente de trabalho. Para este artigo, preparamos um ambiente de teste em Linux Ubuntu 13.04 porem, pode ser repetido em qualquer sistema operacional.

1. Esteja certo de que o Java Development Kit (JDK) esta instalado. Caso não esteja, faça o download neste link: http://www.oracle.com;
2. Instale o Eclipse ou NetBeans, caso ainda não o tenha feito, e tenha certeza de estar usando a versão de 32 bits do IDE escolhido pois o Sikuli não é compatível com as versões de 64 bits;
3. Faça o download do Sikuli IDE em http://sikuli.org;
4. Abra o seu projeto no NetBeans ou Eclipse ou crie um novo projeto;
5. Clique com o botão direito no projeto e escolha "propriedades" , depois bibliotecas (no netbeans) ou buildpatch (no eclipse);
6. Na janela que se seguira, devemos escolher a opção "adicionar jar/pasta" e então navegamos até a pasta de instalação do Sikuli e selecionamos (sikuli-script.jar) e depois em "ok";
7. O próximo passo é criar uma classe de teste. A classe de teste não sera usada para um teste unitário real, ela sera usada para testarmos a interface de usuário através do sikuli.
8. Adicione tantos casos de teste quantos desejar.

Seguindo com nossa implementação prática, logo que criarmos nosso caso de teste (Junit Test) ele se apresentara desta forma:

```
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

public class novo {

   public novo() {
   }

   @BeforeClass
   public static void setUpClass() {
   }

   @AfterClass
   public static void tearDownClass() {
   }

   @Before
   public void setUp() {
   }

   @After
   public void tearDown() {
   }
}
```

Algumas importações são desnecessárias, portanto, vamos manter apenas a importação da biblioteca junit e o metodo setUp() para o nosso objetivo. Eliminando os códigos desnecessários, nossa classe ficara desta forma:

```
import org.junit.*;

public class novo {

   public novo() {
   }

   @Before
   public void setUp() {
   }
}
```

Agora vamos adicionar a importação da biblioteca sikuli e adicionamos também alguns atributos a classe que iremos usar.

```
import org.junit.*;
import org.sikuli.script.*;

public class novo {

    private SikuliScript script;
    private Screen tela;
    private Pattern projeto;
    private Pattern executa;
    private Pattern soma;
    private Pattern um;
    private Pattern dois;
    private Pattern tres;
    private Pattern igual;



    public novo() {
    }

    @Before
    public void setUp() {
    }
}
```

Um atributo do tipo “Pattern” é usado para associar um arquivo de imagem com atributos adicionais usados nas operações de localização de um objeto. Atributo do tipo “Screen” representa a tela em que sera realizada alguma ação.

Após inserir os atributos, completamos o método setUp() que vai instanciar nossos atributos antes de realizar o teste. Soluções de testes baseado em imagem podem ser usadas não apenas para inspeções visuais, mas também para testes funcionais, no entanto, a tecnologia ainda não está madura o suficiente para substituir as soluções orientadas a objeto, como o selenium, por exemplo.

Seguindo com nosso exemplo temos:

```
import org.junit.*;
import org.sikuli.script.*;

public class novo {

    private SikuliScript script;
    private Screen tela;
    private Pattern projeto;
    private Pattern executa;
    private Pattern soma;
    private Pattern um;
    private Pattern dois;
    private Pattern tres;
    private Pattern igual;
```

```
   public novo() {
   }

   @Before
   public void setUp() {
                try {
script = new SikuliScript();
script.run(gcalctool) //gcalctool é o comando para executar a calculadora nativa do Ubuntu Linux
       } catch (AWTException ex) {
          Logger.getLogger(TesteSikuli.class.getName()).log(Level.SEVERE, null, ex);
       }
       tela = new Screen();
       projeto = new Pattern("/img/projeto.jpg");
       executa = new Pattern("/img/executa.jpg");
       soma = new Pattern("/img/soma.jpg");
       um = new Pattern("/img/1.jpg");
       dois = new Pattern("/img/2.jpg");
       tres = new Pattern("/img/3.jpg");
       igual = new Pattern("/img/igual.jpg");
   }
}
```

A partir deste ponto já podemos criar quantos casos de teste desejarmos, sempre tendo em mente que não se trata de um teste unitário, mas sim de um teste de interface.

```
@Test
   public void TestSikuliApi() throws Exception {

       tela.click(projeto);

       tela.wait(executa);
       tela.click(executa);

       tela.wait(um);
       for (int i = 0; i < 2; i++) {
          tela.click(um);
          tela.click(dois);
          tela.click(tres);
          tela.click(soma);
       }
          tela.click(igual);

   }
```

Após o teste em si, pode-se inserir um metodo tearDown() com algumas rotinas para finalizar o teste, para nosso exemplo não foi necessário. O resultado final para a nossa classe de teste integrado com Sikuli Script é este:

```
import java.awt.AWTException;

import java.util.logging.Level;
```

```
import java.util.logging.Logger;
import org.junit.*;
import org.sikuli.script.*;

public class TesteSikuli {

    private SikuliScript script;
    private Screen tela;
    private Pattern projeto;
    private Pattern executa;
    private Pattern soma;
    private Pattern um;
    private Pattern dois;
    private Pattern tres;
    private Pattern igual;

    public TesteSikuli() {

    }
    @Before
    public void setUp() {
        try {
            script = new SikuliScript();

        } catch (AWTException ex) {
            Logger.getLogger(TesteSikuli.class.getName()).log(Level.SEVERE, null, ex);
        }
        tela = new Screen();
        projeto = new Pattern("/img/projeto.jpg");
        executa = new Pattern("/img/executa.jpg");
```

```
        soma = new Pattern("/img/soma.jpg");
        um = new Pattern("/img/1.jpg");
        dois = new Pattern("/img/2.jpg");
        tres = new Pattern("/img/3.jpg");
        igual = new Pattern("/img/igual.jpg");

    }
    @Test
    public void TestSikuliApi() throws Exception {

        tela.click(projeto);

        tela.wait(executa);
        tela.click(executa);

        tela.wait(um);
        for (int i = 0; i < 2; i++) {
            tela.click(um);
            tela.click(dois);
            tela.click(tres);
            tela.click(soma);
        }
            tela.click(igual);

    }
}
```

Caso tenha o objetivo de usar bastante o Sikuli Script em seus projetos Java, mais cedo ou mais tarde você poderá querer usar entradas de teclado para entrada de textos ou comandos como Ctrl + Alt + Del, por exemplo. O Sikuli Script suporta entrada de texto alfa-numérico nativamente, mas para executar comandos de teclado ele necessita de um robô. Para simples digitação de textos alfa-numéricos usa-se o método type() da classe Screen(), para uso de comandos entra em ação o nosso robô. Este robô é acessado através da classe Robot() do pacote java.awt e os eventos de teclado são fornecidos pela classe KeyEvent() do pacote java.awt.event. Para eventos do mouse os mais interessantes para nossos estudos são:

- click();
- rightClick() e;
- doubleClick();

Todos estes eventos acessíveis pela classe Screen() do Sikuli Script.

Integrando o Sikuli Script com o Selenium

Ao trabalhar com selenium, às vezes nos deparamos com situações inesperadas em determinados elementos de páginas onde nenhum roteiro de teste é capaz de executar (muitas vezes isso pode ser um problema quando se trabalha com uma páginas que tem iframes ou links que acionam eventos em JavaScript ou aplicações em flash).

Uma correção para esse problema é chamar o Sikuli juntamente com seu script selenium, com isso, podemos executar a parte onde o selenium é ineficaz, e depois continuar com o scripting em Selenium.

Como complemento do nosso exemplo anterior, vamos mostrar a integração do Sikuli Script com Selenium WebDriver. Para tal tarefa, vamos agora apenas incluir em nosso projeto as bibliotecas do Selenium que estão disponíveis para download nos links a seguir:

- selenium-server-standalone-2.25.0.jar: (https://code.google.com/p/selenium/downloads/detail?name=selenium-server-standalone-2.25.0.jar&can=2&q=label%3AFeatured)
- selenium-java-2.25.0.jar: (https://code.google.com/p/selenium/downloads/detail?name=selenium-java-2.25.0.zip&can=2&q=label%3AFeatured)

Após o download das referidas bibliotecas, adicionamos as mesmas ao nosso projeto da mesma maneira que fizemos com a biblioteca Sikuli Script. A seguir, adicione as importações do WebDrive.

- import org.openqa.selenium.WebDriver;
- import org.openqa.selenium.firefox.FirefoxDriver.

Feito isso, agora criamos um novo caso de teste com o selenium webdriver. O código para esse teste é o que segue.

```
@Test
public void functionName() throws FindFailed {

        // Instancia o firefox webdriver
        WebDriver driver = new FirefoxDriver();

        // Abre a página do google no firefox
        driver.get("http://www.google.com.br");

        Screen screen = new Screen();

        //Adiciona o caminho da imagem
        Pattern busca = new Pattern("/img/botaoPesquisar.jpg");

        //O método wait() vai aguardar por 10 segundos até que a imagem apareça
        screen.wait(busca, 10);

        //O método click() faz um click na imagem desejada
        screen.click(busca);
}
```

Tudo muito simples e funcional, com isso conseguimos automatizar testes tanto em ambientes web quanto em ambientes desktop. Sikuli é uma ferramenta muito robusta e em pleno desenvolvimento, esperamos que os exemplos mostrados possam ser uteis no desenvolvimento e testes de suas aplicações em Java.

É importante observar também que existem algumas desvantagens em se utilizar técnicas de scripts visuais, como por exemplo, se os elementos da interface de usuário são alterados periodicamente ao longo do tempo, como uma lista dinâmica de itens que são clicáveis ou outra lista dinâmica qualquer, o sikuli pode se confundir quando o visual não corresponder 100% com o que está contido no script. Mudar o nome de um arquivo que deve ser clicado e aberto em um script do sikuli pode interromper a execução do mesmo e causar um erro. O mesmo acontece com as imagens, tudo precisa estar perfeitamente visível, ou então o script pode falhar.

No geral, o sikuli é uma maneira muito interessante para automatizar processos de teste utilizando imagens e comandos fáceis de entender. Agora pode-se trazer um universo de macros como as existentes no Microsoft Office para todo tipo de aplicações. É fácil de aprender, está constantemente sendo atualizado e é disponibilizado gratuitamente.

Considerações Finais

Este artigo apresentou os principais conceitos que compõem a tecnologia por trás do Sikuli Script. Também foram mostrados dois exemplos práticos da implementação do Sikuli em diferentes cenários ilustrando o enorme potencial que a ferramenta possui seja pelo fato da portabilidade que a ferramenta propicia ou pela flexibilidade que ela oferece em atuar em conjunto com outras soluções consolidadas. É de grande interesse ao especialista em testes automatizados ter conhecimento dessa solução a começar pelo fato da mesma ser um projeto de código livre permitindo sua customização e com isso ser adequada as necessidades específicas do projeto no qual será empregado, bem como a ampla cobertura que ela proporciona como visto anteriormente sendo possível construir testes automatizados para aplicações Web, Desktop e móveis. A medida que o tempo passa a comunidade responsável pelo projeto cresce gradativamente a medida que as equipes de testes cada vez mais cobradas a entregarem soluções de testes automatizados a um custo baixo e a realidade nos mostrando ser muito difícil de atingir este objetivo ao escolhermos uma solução proprietária que em geral possuem custos proibitivos que somente contribuiriam em encarecer o produto final. Por fim, esperamos ter inspirado o leitor deste artigo a pesquisar sobre a solução e a tentar implementá-la, bem como a divulgar a existência e o potencial que esta solução possui.

Referencias

Tese de Doutorado **"Using Graphical Representation of User Interfaces as Visual References"** defendida por Tsung-Hsiang Chang. Disponível em : http://groups.csail.mit.edu/uid/other-pubs/vgod-thesis.pdf

Tese de Doutorado **"Interacting with computers using images for search and automation"** defendida por Tom Yeh. Disponível em https://el.trc.gov.om/htmlroot/ENGG/tcolon/e_references/NDLTD/Information%20and%20Technology%20Engineering/Thesis/Interacting%20with%20Computers%20using%20images%20for%20Search%20and%20Automation.pdf

Artigo **"Sikuli: Using GUI Screenshots for Search and Automation"** por Tom Yeh, Tsung-Hsiang Chang e Robert C. Miller. Disponível em http://groups.csail.mit.edu/uid/projects/sikuli/sikuli-uist2009.pdf

Artigo **"Gui Testing using Sikuli And Java"**

Disponivel em http://www.8bitavenue.com/2012/03/gui-testing-using-sikuli-and-java/

Artigo "**HOW TO INTEGRATE SIKULI SCRIPT WITH SELENIUM WEBDRIVER**"

Disponivel em http://devengineering.com/blog/testing/how-integrate-sikuli-script-selenium-webdriver

Documentação do **Sikuli Script**. Disponível em http://doc.sikuli.org/

Documentação do **Sikuli WebDriver**. Disponível em https://code.google.com/p/sikuli-api/wiki/SikuliWebDriver