Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Projekt 5

Využití nástrojů pro testování grafického uživatelského rozhraní

Plzeň 2015 Jaroslav Klaus

Obsah

1 Úvod

Testování aplikací je nedílnou součástí jejich vývoje a v dnešní době se tomuto oddílu tvorby aplikací věnuje čím dál více pozornosti. Dá se rozdělit do různých skupin, např. podle toho, kdy se testování provádí, jakým způsobem se provádí, jak se k testované aplikaci přistupuje, či jaká část aplikace se podrobuje testům.

Jednou z důležitých součástí je testování grafického uživatelského rozhraní. Zde se testeři soustředí na to, zda daná aplikace vypadá tak, jak to požadují vývojáři a návrh, a zda grafické prvky správně fungují. Dále se zaměřuje na to, zda je aplikace přívětivá k uživateli a práce s ní není příliš komplikovaná.

Při testování grafického uživatelského rozhraní se může spousta testů mnohokrát opakovat, a proto je snaha tyto testy nějak automatizovat. K tomu se může využít některý z nástrojů k tomu určený. Cílem této práce je seznámit se s některými z těchto nástrojů, jeden z nich vybrat a pomocí něj vytvořit sadu ukázkových testů svou filosofií zapadajících do předmětu KIV/OKS.

2 Přehled nástrojů

V této kapitole následuje přehled nástrojů a některých jejich vlastností. V tabulce 2.1 je uveden název nástroje, jeho licence resp. cena, jazyk, ve kterém se testy píší, platforma, na které nástroj funguje a která GUI je nástroj schopen testovat. Z bezplatných multiplatformních nástrojů jsem si vybral tři a ty podrobněji prozkoumal a porovnal, viz následující kapitola.

Tabulka 2.1: Přehled nástrojů

Název	**Licence/Cena**	**Skriptovací jazyk**	**Platforma**	**Jazykové omezení**
AutoIt[?]	Freeware	BASIC-like	Windows	-
AutoHotKey[?]	GNU GPLv2	AutoHotKey	Windows	-
AutoKey[?]	GNU GPLv3	Python	Linux	-
SikuliX[?] [?]	MIT License	Python, Ruby	Windows, Linux, Mac	-
Jubula[?]	EPL 1.0	Drag & Drop, Java	Windows, Linux, Mac	Java, HTML, .NET, iOS
Robot Framework[?]	Apache License 2.0	Natural-like	Windows, Linux, Mac	Podle pluginů (Java, web, Android, iOS, ...)
Squish[?]	cca 2400 €/osoba	Python, JavaScript, Ruby, Perl, Tcl	Windows, Linux, Mac	-
eggPlant[?]	nedostupná, vázaná na stroj	SmartTalk, Drag & Drop, pomocí rozhraní eggDrive např. Java, C#, Ruby	Windows, Linux, Mac	-
UFT[?]	nedostupná	VBScript, Drag & Drop	Windows, Linux, Mac	-
Rational Functional Tester [?]	3300 $/osoba	Nahrávání akcí	Windows, Linux	-
Ranorex[?]	690 €	C#, VisualBasic, nahrávání akcí	Windows	-
SilkTest[?]	nedostupná	C#, VisualBasic, Java	Windows	-
TestComplete [?]	889 €/stroj	Python, VBScript, JScript, C#Script, DelphiScript, C++Script, nahrávání akcí	Windows	-

3 Zvolené nástroje

Vzhledem k požadavkům na nástroje, které vyplývají z vazby na předmět KIV/OKS, jako je bezplatnost, schopnost fungování nezávisle na OS nebo podpora testování programů vytvořených technologií Java a webových aplikací, jsem z výše zmíněných vybral nástroje Jubula, SikuliX a Robot Framework. Každý z nástrojů bude stručně charakterizován a bude následovat podrobnější srovnání.

3.1 Jubula

Jubula je nástroj, který vznikl a je vyvíjen v rámci IDE Eclipse. Do projektu přispívá také firma BREDEX GmbH, která vytváří i tzv. standalone verzi, což je program, který je možné používat samostatně bez IDE Eclipse. Navíc obsahuje navíc některé nespecifikované funkce a nemusí být licencována pod EPL 1.0, jako je tomu u verze pro IDE Eclipse.

Pro tvorbu testovacích skriptů byla používána metoda Drag & Drop, popř. se akce určovaly klikáním na různé nabídky. V jedné z posledních verzí bylo vydáno Java API a skripty je tak možné psát pomocí jazyka Java. Mezi podporovaná testovaná rozhraní patří Java Swing, SWT, JavaFX, HTML a iOS. Výhodou této aplikace je také možnost její integrace do ostatních programů pro organizaci testování.

3.2 SikuliX

Sikuli (nověji SikuliX) je nástroj, který vznikl jako projekt skupiny User Interface Design Group na MIT, což odpovídá i jeho licenci - MIT License. Nyní jeho vývoj převzal Raimund Hock (aka RaiMan) společně s open-source komunitou.

Při tvorbě skriptů je možné využít pro SikuliX vlastní jazyk podobný přirozené angličtině, nebo některý ze zavedených, jako je Python, Ruby, Java, Jython, JRuby, Scala, Groovy, Clojure a další. Nástroj není omezený na určitá testovaná rozhraní, protože k identifikaci GUI používá rozpoznávání

obrazu podle vzoru[1], dokáže simulovat ovládání myši a klávesnice nebo rozpoznávat text v obrázcích[2]. Výhodou této aplikace je proto její nezávislost vůči testovanému rozhraní. Cenou za to je pravděpodobné snížení její rychlosti.

3.3 Robot Framework

Robot Framework je nástroj založený na pluginech a je open-source. Vývoj podporuje společnost Nokia Networks.

Základ nástroje, tzv. core framework, je vytvořený v jazyce Python. Knihovny je možné psát v jazyce Python nebo Java a samotné skripty pak v jazyce podobném přirozené angličtině. Díky dodržování jistého formátování je pro člověka velmi přehledný. Mezi podporovaná testovaná rozhraní patří např. Android, iOS, Java Swing, webové aplikace, databáze a aplikace vytvořené pro OS Windows. Výhodou této aplikace je možnost si chybějící modul pro testování určitého rozhraní vytvořit a používat.

[1] Pomocí OpenCV, http://opencv.org/
[2] Pomocí Tesseract OCR, https://github.com/tesseract-ocr

4 Srovnání nástrojů

Pro srovnání nástrojů jsem vytvořil návrh multikriteriálního hodnocení, který se snaží nástroje hodnotit z různých úhlů a vytvořit tak komplexní klasifikaci. Každé z hodnocených částí je možné přiřadit vlastní váhu. Ta určuje důležitost hodnotícího kritéria pro každého jedince a tím napomáhá výběru vhodného nástroje. V obrázku 4.1a je návrh ukázán a je vidět výsledek pro mnou zvolené váhy, viz obrázek 4.1b. Jako nejvhodnější se jeví použití nástroje SikuliX. Dále se budu věnovat jednotlivým hodnotícím kritériím.

Možnost vytváření skriptů je jedno z nejdůležitějších kritérií vzhledem k vazbě na předmět KIV/OKS. Hlavním požadavkem bylo, aby bylo možné skripty tvořit v jazyce Java. Dále jsem vybral několik skriptovacích jazyků a metod.

Podpora testovaných rozhraní byla dalším z rozhodujících kritérií. Hlavními platformami měly být aplikace vytvořené pomocí jazyka Java a webové aplikace. Opět jsem přidal některé další běžné platformy. Nástroj by měl být též multiplatformní, proto je jedním z kritérií podpora operačních systémů.

Reportování výsledků testů, složitost jejich tvorby a jejich přehlednost může napomoci vývojáři diagnostikovat případnou chybu. Také je přínosné znát stav obrazovky a to zajistí screenshot. Díky tomu se stává vývoj jednodušší, a proto jsem toto kritérium také zařadil do hodnocení.

Dále jsem přidal kritérium univerzálnosti nástroje s poněkud individuálním ohodnocením. To je zde myšleno tak, co obecně nástroj dokáže, ale co není podstatné z pohledu předmětu KIV/OKS. Například Jubula je čistě testovací nástroj, ale SikuliX se dá použít navíc pro automatizaci pracovních postupů.

Posledním kritériem je vhodnost nástroje pro účely předmětu KIV/OKS. Jedná se hlavně o to, jak zapadá do konceptu výuky, jak je práce s ním složitá a jaké má nároky na studentovy znalosti.

Vysvětlivky termínů z dále použitých tabulek:

- Natural-like – víceméně okleštěný přirozený jazyk založený na angličtině,
- Složitost tvorby – míní se tím složitost přípravy reportu a v tabulce

vyšší počet bodů znamená jednodušší tvorbu pro tvůrce skriptů

(a) Multikriteriální hodnocení

Kritéria multikriteriálního hodnocení			
Možnosti vytváření skriptů	**Jubula**	**RobotFramework**	**SikuliX**
Java	Ano	Ano	Ano
Python	Ne	Ano	Ano
Ruby	Ne	Ano	Ano
Drag & Drop	Ano	Ne	Ne
Natural-Like	Ne	Ano	Ano
Body	**2**	**4**	**4**
Podpora testovaných rozhraní			
SWT	Ano	Ano	Ano
Java Swing	Ano	Ano	Ano
JavaFX	Ano	Ne	Ano
.NET	Ano	Ano	Ano
HTML	Ano	Ano	Ano
Android	Ne	Ano	Ano
iOS	Ano	Ano	Ano
Body	**6**	**6**	**7**
Podporované operační systémy			
Linux/Unix	Ano	Ano	Ano
Windows	Ano	Ano	Ano
Mac	Ano	Ano	Ano
Body	**3**	**3**	**3**
Reportování výsledků testů			
HTML	Ano	Ano	Ano (pomocí JUnit a Ant, resp. HTML Test Runner a unittest)
XML	Ano	Ano	Ano (pomocí JUnit a Ant, resp. HTML Test Runner a unittest)
Složitost tvorby (čím vyšší, tím snazší)	4	4	1
Přehlednost reportu	4	4	3
Body	**10**	**10**	**6**
Screenshot při chybě	Ano	Ano	Ano
Body	**1**	**1**	**1**
Univerzálnost nástroje	**2**	**3**	**5**
Vhodnost nástroje pro účely KIV/OKS	**4**	**3**	**4**

(b) Výsledek multikriteriálního hodnocení

Váhy	**Vlastní váha**	**Celková váha**	**Váha v %**
Možnosti vytváření skriptů	4	0,10	10,3
Podpora testovaných rozhraní	8	0,21	20,5
Podporované operační systémy	3	0,08	7,69
Reportování výsledků testů	5	0,13	12,8
Screenshot při chybě	4	0,10	10,3
Univerzálnost nástroje	7	0,18	17,9
Vhodnost nástroje pro účely KIV/OKS	8	0,21	20,5
Součet	**39**	**1**	**100**
Celkové zhodnocení nástrojů	**Jubula**	**RobotFramework**	**SikuliX**
Možnosti vytváření skriptů	0,21	0,41	0,41
Podpora testovaných rozhraní	1,23	1,23	1,44
Podporované operační systémy	0,23	0,23	0,23
Reportování výsledků testů	1,28	1,28	0,77
Screenshot při chybě	0,10	0,10	0,10
Univerzálnost nástroje	0,36	0,54	0,90
Vhodnost nástroje pro účely KIV/OKS	0,82	0,62	0,82
Celkové skóre	**4,23**	**4,41**	**4,67**

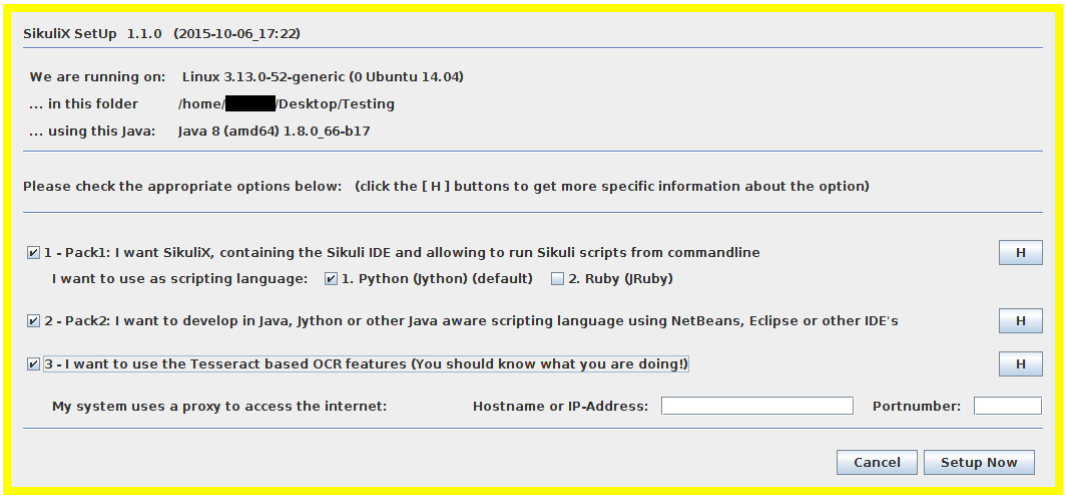
Z uvedeného multikriteriálního hodnocení je zřejmé, že vybrané nástroje jsou v podstatě vyrovnané. To ostatně potvrzuje i jejich poměrné zastoupení mezi uživateli. SikuliX byl zvolen též po diskuzích s vedoucím práce a to pro jeho vlastnost naprosté nezávislosti na testovaném rozhraní. Tato vlastnost se významně hodí v předmětu KIV/OKS, protože je možné dát nástroj do kontrastu s nástrojem Selenium. Jinými slovy řečeno SikuliX je principiálně odlišný nástroj, což není možné říci o např. Jubule.

5 SikuliX

5.1 Instalace

Po stažení balíčku započne instalace jeho spuštěním[1]. Je ukázána instalace v Linuxu, avšak instalace ve Windows je obdobná. V průběhu máme na výběr různé možnosti, jak chceme nástroj používat, viz obrázek 5.1. Např. zda chceme používat SikuliX-IDE a Python nebo Ruby, jestli budeme používat jiné IDE a Javu a zda chceme používat OCR funkce. Zaškrtneme všechna políčka kromě *Ruby (JRuby)* a klikneme na *Setup Now*. Jsme dotázáni, zda chceme balíčky stáhnout, nebo ukončit instalaci. Zvolíme *Yes*. Další dotaz je na verzi Jythonu, kterou chceme použít, s upozorněním, že může nastat problém se znaky v kódování UTF-8. Opět zvolíme *Yes*. Začne vytváření souborů a měla by se otevřít dvě okna jako na obrázu 5.2, obě potvrdíme tlačítkem *OK*. Pokud vše proběhne v pořádku, vzniknou v adresáři soubory podobné těmto[2] *runsikulix, SetupStuff, SikuliX-1.1.0-SetupLog.txt, sikulixapi.jar, sikulix.jar*.

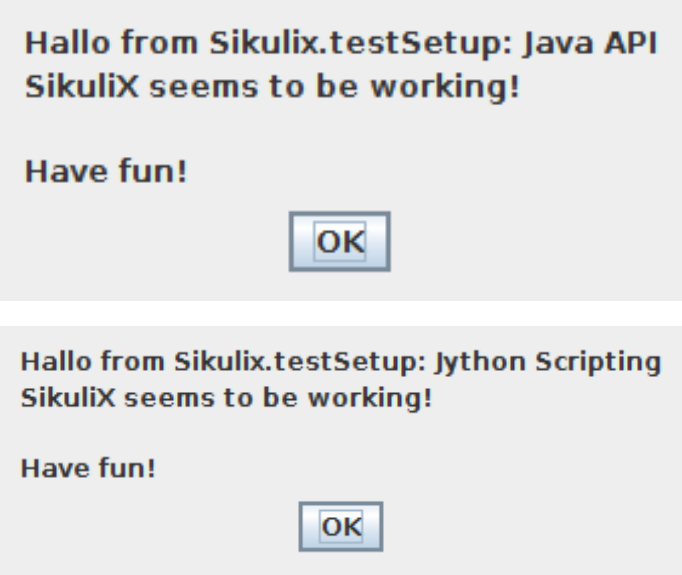
Obrázek 5.1: Instalace SikuliX



[1] Je potřeba instalace JRE nebo JDK 6 a vyšší, v linuxové distribuci balíky *libopencv-core2.4, libopencv-imgproc2.4, libopencv-highgui2.4, libtesseract3* a *wmctrl* [**?**]

[2] Může se lišit na různých OS

Obrázek 5.2: Test instalace



5.2 SikuliX-IDE

Spustit SikuliX-IDE je možné různými způsoby [**?**].

1. Spuštěním souboru SikuliX.app (Mac) nebo SikuliX.exe (Windows),
2. dvojklikem na soubor runsikulix (Linux) nebo runsikulix.cmd (Windows),
3. z příkazové řádky příkazem `java -jar cesta/k/sikulix.jar [volitelne parametry]`.

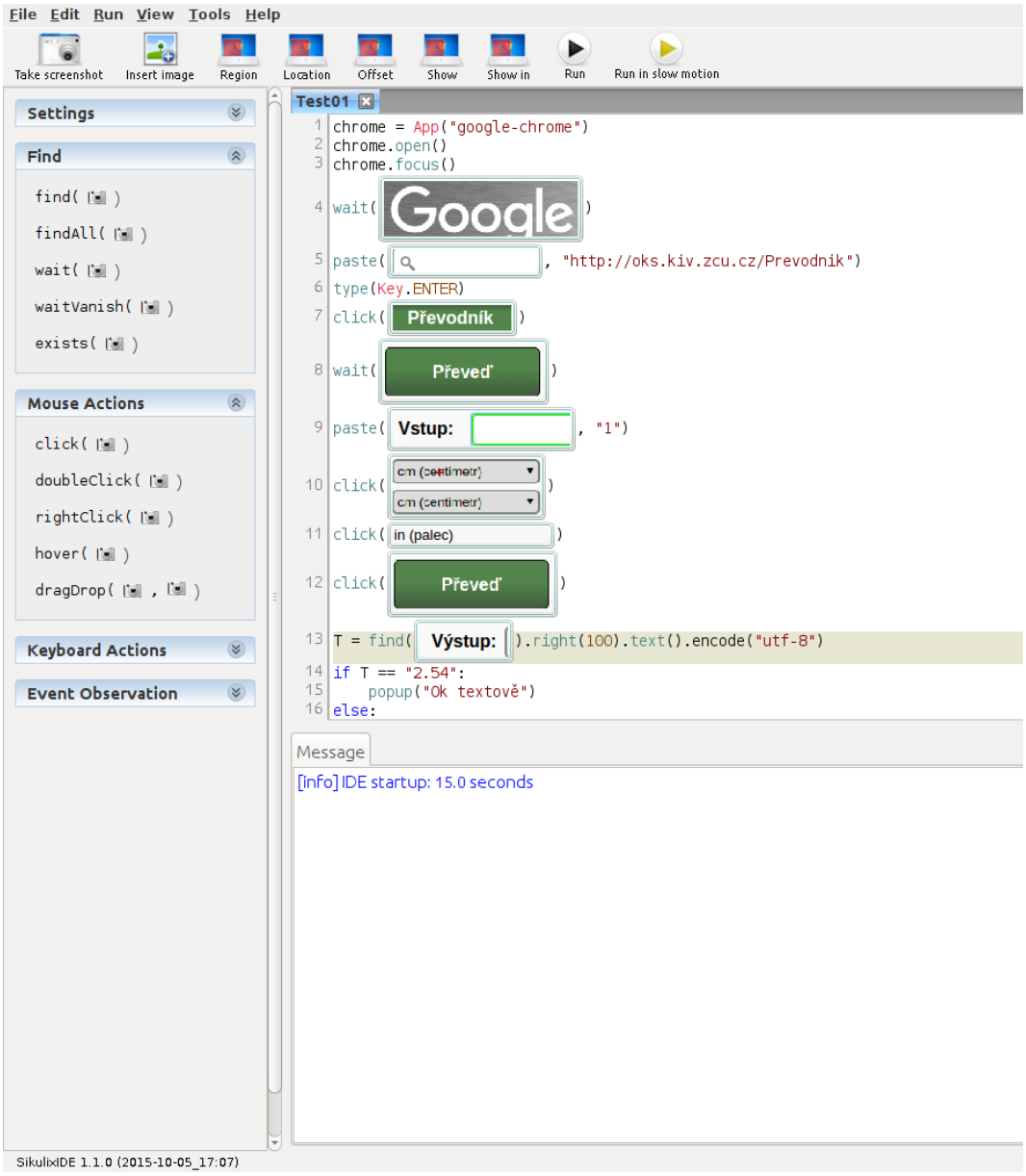
Po spuštění vypadá IDE jako na obrázku 5.3. Jako parametry se v metodách, ve kterých je to možné, ukazují obrázky vzorů, podle kterých se na obrazovce nástroj orientuje.

5.2.1 První skript

Skript se připravuje v SikuliX-IDE, které je vidět na obrázku 5.3. Kód, který je vidět v 5.1, není v IDE identický, ale cesta k obrázku je vždy nahrazena jeho náhledem. K tvorbě jsou v IDE užitečné pomůcky, které se nacházejí v levém a v horním panelu.

Skript pracuje tak, že se otevře prohlížeč, který přejde na adresu `http://oks.kiv.zcu.cz/Prevodnik`. Klikne na odkaz *Převodník*, do vstupního

Obrázek 5.3: SikuliX-IDE



pole vloží *1* a stiskne *Převeď*. Z pole s výsledkem přečte text a porovná jej s předpokládanou hodnotou *2,54*. Pokud si odpovídají, objeví se vyskakovací okno s potvrzením, jestliže ne, zobrazí se chybová hláška. Obdobně je tomu v následující části, kde se pouze kontroluje existence obrázku.

Kód 5.1: První skript

```
prohlizec = App("google-chrome")
prohlizec.open()          #otevre aplikaci definovanou
                          #vyse
prohlizec.focus()         #vybere do popredi jeji okno
#ceka, dokud na obrazovce nenajde obrazek
wait("obr1.png")
#najde na obrazovce obrazek a vlozi do neho text
paste("obr2.png", "http://oks.kiv.zcu.cz/Prevodnik")
type(Key.ENTER) #simuluje stisk klavesy ENTER
#najde na obrazovce obrazek a klikne na nej
click("obr3.png")
wait("obr4.png")
paste("obr5.png", "1")
#klikne o 27px vyse a 18px vlevo od nalezeneho
#obrazku
click(Pattern("obr6.png").targetOffset(-27,-18))
click("obr7.png")
click("obr4.png")
#precte text z casti, ktera je 100px vpravo od
#nalezeneho obrazku
T = find("obr8.png").right(100).text()
if T == "2.54":
        #pokud rozpoznany text souhlasi se zadanym,
        #otevre se vyskakovaci okno
    popup("Ok textove")
else:
    popError("Chyba")     #jinak se zobrazi chybove
                          #okno

if exists("obr9.png"):
        #pokud na obrazovce existuje obrazek, otevre
        #se vyskakovaci okno
    popup("Ok obrazove")
else:
    popError("Chyba")
prohlizec.close()    #ukonci aplikaci
```

5.3 Java API

Dále bylo zkoumáno Java API, které SikuliX poskytuje. Pro jeho použití je potřeba mít při překladu a spuštění nastavený v classpath *sikulixapi.jar*. Toho docílíme např. tak, že použijeme v příkazové řádce dvou příkazů
`javac -cp sikulixapi.jar:. Test01.java`
`java -cp sikulixapi.jar:. Test01`
Syntaxe, kterou SikuliX v Java API využívá, je velmi podobná té v SikuliX-IDE.

5.3.1 První test

První test s použitím Java API, viz kód 5.2, je identický s tím, který byl vytvořen pomocí SikuliX-IDE.

Kód 5.2: První test Java API

```
import org.sikuli.basics.Settings;
import org.sikuli.script.*;
import javax.swing.*;

public class Test01 {

  static Screen s;
  static App prohlizec;

  public static void main(String [] args) {
    Settings.OcrTextSearch = true;
    Settings.OcrTextRead = true;

    s= new Screen();
    prohlizec = new App("google-chrome");
    prohlizec.open();
    prohlizec.focus();

    try {
      s.wait("obr1.png");
      s.paste("obr2.png");
      s.type(Key.ENTER);
      s.click("obr3.png");
```

```
        s.wait("obr4.png");
        s.paste("obr5.png", "1");
        s.click(new Pattern("obr6.png").targetOffset(
          -27,-18));
        s.click("obr7.png");
        s.click("obr4.png");
        String t = s.find("obr8.png").right(
          100).text();
        if (Double.parseDouble(t) == 2.54) {
          JOptionPane.showMessageDialog(null, "Ok" +
            " textove");
        } else {
          JOptionPane.showMessageDialog(null, "Chyba");
        }
        if (s.exists("obr9.png") != null) {
          JOptionPane.showMessageDialog(null, "Ok" +
            " obrazove");
        } else {
          JOptionPane.showMessageDialog(null, "Chyba");
        }
        prohlizec.close();
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
}
```

5.3.2 Sofistikovanější testy

S využitím knihoven *JUnit* a *Log4j* (ani jedna z těchto knihoven není pro běh SikuliX bezprostředně nutná) byly vytvořeny čtyři testy, viz kód 5.3. První test skončí negativně, druhý pozitivně, třetí pozitivně a čtvrtý negativně.

Kód 5.3: Další testy Java API

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
```

```
import org.junit.Test;
import org.junit.rules.ErrorCollector;
import org.sikuli.basics.Debug;
import org.sikuli.basics.Settings;
import org.sikuli.script.*;
import javax.swing.*;
import java.time.LocalDateTime;
import static org.junit.Assert.*;

public class Test01 {

  static Logger logger;
  static ErrorCollector collector;
  static Screen s;
  static App prohlizec;
  static boolean run;

  static {
    System.setProperty("log4j.configurationFile",
      "log-konfigurace.xml");
  }

  private String nazevScreenshotu() {
    LocalDateTime l = LocalDateTime.now();
      return l.getYear() + "" + l.getMonthValue() +
        "" + l.getDayOfMonth() + "" + l.getHour() +
        "" + (l.getMinute() < 10 ? "0" + l.
        getMinute() : l.getMinute()) + "" + l.
        getSecond() + "";
  }

  @BeforeClass
  public static void setUpBeforeClass() {
    logger = LogManager.getLogger();

    Settings.OcrTextSearch = true;
    Settings.OcrTextRead = true;
    Debug.setLogger(logger);
    Debug.setLoggerAll("info");

    collector = new ErrorCollector();
```

```
    s = new Screen();
    prohlizec = new App("google-chrome");
    prohlizec.open();
    prohlizec.focus();
    run = true;
    }

  @AfterClass
  public static void tearDownAfterClass() {
    JOptionPane.showMessageDialog(null, "Script" +
      " dokoncen");
    prohlizec.close();
  }

  @Before
  public void setUp() {
    try {
      s.wait("png/addressBar.png", 10);
      s.click(new Pattern("png/addressBar.png").
        targetOffset(100, 0));
      s.paste("http://oks.kiv.zcu.cz/Prevodnik");
      s.type(Key.ENTER);
      s.wait("png/zalozkaPrevodnik.png", 5);
    } catch (Exception e) {
      run = false;
      s.capture().save("errors", nazevScreenshotu());
      logger.error(e.getMessage());
    }
  }

  @Test
  public void testPorovnejText() {
    if (run) {
      try {
        s.click("png/zalozkaPrevodnik.png");
        s.wait("png/tlacitkoPreved.png", 5);
        s.paste("png/vstup.png", "1");
        Match m = s.find("png/jednotky.png");
        m.setTargetOffset(-27, -18);
        m.click();
        s.findText("(metr)").click();
```

```
        s.click(new Pattern("png/jednotky.png").
          targetOffset(-27, 18));
        s.find("png/dm.png").click();
        s.click("png/tlacitkoPreved.png");
        String t = s.find("png/vystup.png").right(
          100).text();
        assertEquals(10, Double.parseDouble(t),
          0.01);
      } catch (FindFailed | AssertionError e) {
        s.capture().save("errors",
          nazevScreenshotu());
        logger.error(e.getMessage());
        fail(e.getMessage());
      }
    } else {
      run = true;
      logger.error("setUp neuspesny");
      fail("setUp neuspesny");
    }
  }

  @Test
  public void testPorovnejObraz() {
    if (run) {
      try {
        s.click("png/zalozkaPrevodnik.png");
        s.wait("png/tlacitkoPreved.png", 5);
        s.paste("png/vstup.png", "1");
        s.click(new Pattern("png/jednotky.png").
          targetOffset(-27, -18));
        s.click("png/inch.png");
        s.click("png/tlacitkoPreved.png");
        assertTrue(s.exists("png/vysledek.png") !=
          null);
      } catch (FindFailed | AssertionError e) {
        s.capture().save("errors",
          nazevScreenshotu());
        logger.error(e.getMessage());
        fail(e.getMessage());
      }
    } else {
```

```
      run = true;
      logger.error("setUp neuspesny");
      fail("setUp neuspesny");
    }
  }

  @Test
  public void testZkontrolujOdkazObrazekKiv() {
    if (run) {
      try {
        s.click("png/logoKiv.png");
        assertTrue(s.exists("png/zahlaviKiv.png") !=
          null);
      } catch (FindFailed | AssertionError e) {
        s.capture().save("errors",
          nazevScreenshotu());
        logger.error(e.getMessage());
        fail(e.getMessage());
      }
    } else {
      run = true;
      logger.error("setUp neuspesny");
      fail("setUp neuspesny");
    }
  }

  @Test
  public void testChyba() {
    if (run) {
      try {
        s.wait("png/tlacitkoPreved.png", 5);
        s.paste("png/vstup.png", "1");
        s.click(new Pattern("png/jednotky.png").
          targetOffset(-27, -18));
        s.findText("(metr)").click();
        s.click("png/tlacitkoPreved.png");
        String t = s.find("png/vystup.png").right(
          100).text();
        assertEquals(100, Double.parseDouble(t),
          0.01);
      } catch (FindFailed | AssertionError e) {
```

```
            s.capture().save("errors",
              nazevScreenshotu());
            logger.error(e.getMessage());
            fail(e.getMessage());
          }
        } else {
          run = true;
          logger.error("setUp neuspesny");
          fail("setUp neuspesny");
        }
      }
}
```

6 Závěr

Seznámil jsem se s některými metodami testování grafického uživatelského rozhraní a zjistil jsem některé důvody používání těchto metod. Dále jsem prozkoumal, které nástroje je k testování možné využít.

Vybral jsem tři programy, které jsem stručně popsal. Navrhl jsem multikriteriální hodnocení a provedl jejich podrobné porovnání. Výsledkem byl výběr jednoho programu, který použiji jako hlavní nástroj v této práci.

S aplikací SikuliX, kterou jsem zvolil předchozí činností, jsem se zběžně seznámil a vytvořil jeden test v prostředí SikuliX-IDE za použití vlastního jazyka SikuliX. Další čtyři testy jsem zhotovil pomocí Java API, které SikuliX nabízí a které budu využívat z důvodu vazby na předmět KIV/OKS.

Dále se hodlám zaměřit na podrobnější zkoumání používání nástroje. Také připravím sadu ukázkových testů a funkční scénáře.

Literatura

[AutoIT(2015)] AUTOIT. *AutoIt* [online]. AutoIt Consulting Ltd, 2015. [cit. 4.10.2015]. Dostupné z: https://www.autoitscript.com/site/autoit/.

[Daniels(2011)] DANIELS, K. W. *AutoHotKey* [online]. 2011. [cit. 4.10.2015]. Dostupné z: https://code.google.com/p/autokey/.

[eggPlant(2015)] EGGPLANT. *eggPlant Functional* [online]. TestPlant, 2015. [cit. 4.10.2015]. Dostupné z: http://www.testplant.com/eggplant/testing-tools/eggplant-developer/.

[Framework(2015)] FRAMEWORK, R. *Robot Framework, Generic test automation framework for acceptance testing and ATDD* [online]. Robot Framework and Nokia Networks, 2015. [cit. 4.10.2015]. Dostupné z: http://robotframework.org/.

[Hocke(2015)] HOCKE, R. *SikuliX* [online]. 2015. [cit. 4.10.2015]. Dostupné z: http://www.sikulix.com/.

[Jubula(2015)] JUBULA. *Jubula, Automated Functional Testing* [online]. Eclipse Foundation and Bredex GmbH, 2015. [cit. 4.10.2015]. Dostupné z: http://www.eclipse.org/jubula/.

[Mallet(2015)] MALLET, C. *AutoHotKey* [online]. 2015. [cit. 4.10.2015]. Dostupné z: https://www.autohotkey.com/.

[Ranorex(2015)] RANOREX. *Ranorex* [online]. Ranorex GmbH, 2015. [cit. 4.10.2015]. Dostupné z: http://www.ranorex.com/.

[RFT(2015)] RFT. *Rational Functional Tester* [online]. IBM, 2015. [cit. 4.10.2015]. Dostupné z: http://www-03.ibm.com/software/products/cs/functional.

[Sikuli(2015)] SIKULI. *Sikuli* [online]. User Interface Design Group at MIT, 2015. [cit. 4.10.2015]. Dostupné z: http://www.sikuli.org/.

[SilkTest(2015)] SILKTEST. *SilkTest* [online]. Micro Focus, 2015. [cit. 4.10.2015]. Dostupné z: http://www.borland.com/en-GB/Products/Software-Testing/Automated-Testing/Silk-Test.

[Squish(2015)] SQUISH. *Squish, GUI Tester* [online]. froglogic GmbH, 2015. [cit. 4.10.2015]. Dostupné z: http://robotframework.org/.

[TestComplete(2015)] TESTCOMPLETE. *TestComplete* [online]. SmartBear Software, 2015. [cit. 4.10.2015]. Dostupné z: http://smartbear.com/product/testcomplete/overview/.

[UFT(2015)] UFT. *Unified Functional Testing* [online]. Hewlett Packard Enterprise Development LP, 2015. [cit. 4.10.2015]. Dostupné z: http://www8.hp.com/cz/cs/software-solutions/unified-functional-automated-testing/.