Grado en Ing. del Software

E.T.S.I. Informática

Mantenimiento y Pruebas de Software
Pruebas de sistema, aceptación y regresión
Pruebas de interfaces de usuario

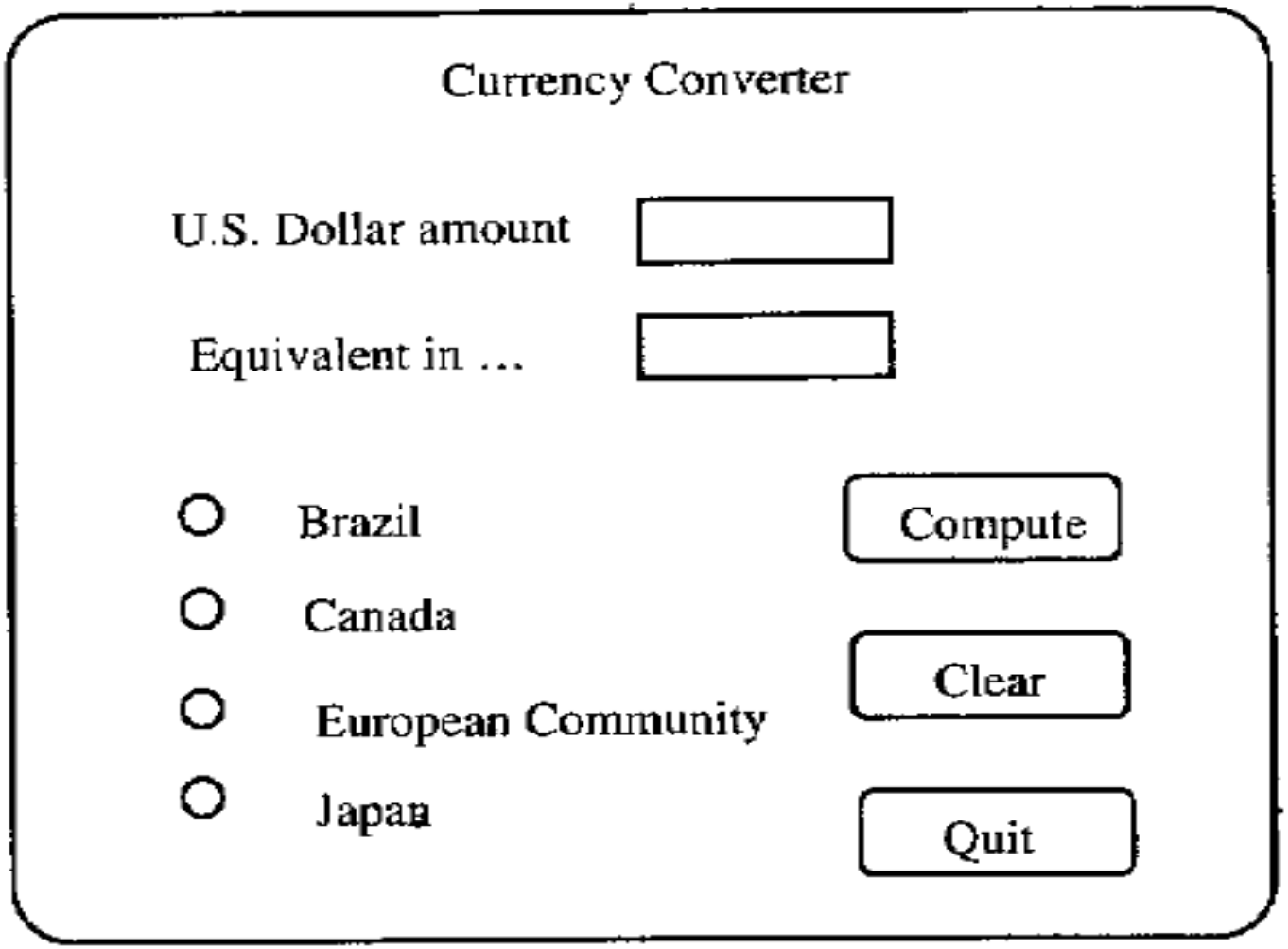
Francisco Javier Durán Muñoz

Lenguajes y Sistemas Informáticos

Pruebas de GUIs

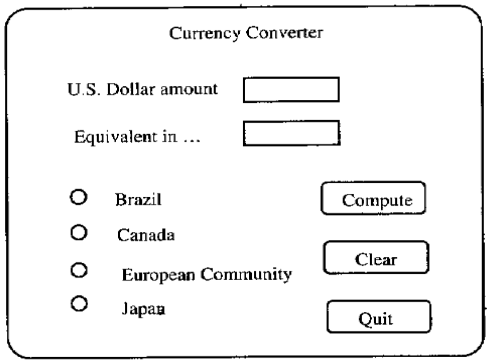
- La principal característica de cualquier aplicación con interfaz gráfica de usuario (GUI) es que es **guiada por eventos**.
- La esencia de las pruebas de sistema para aplicaciones con GUI es ejercitar esta naturaleza guiada por eventos.
- Los modelos UML proporcionan diagramas muy apropiados para ello
  - Diagramas de comportamiento:
    - Statecharts
    - máquinas de estados finitos

El programa de conversión de moneda



Jorgensen, capítulo 19

- Para probar una aplicación con GUI, lo primero será identificar
  - todos los eventos de entrada del usuario y
  - todos los eventos de salida del sistema (observables externamente)

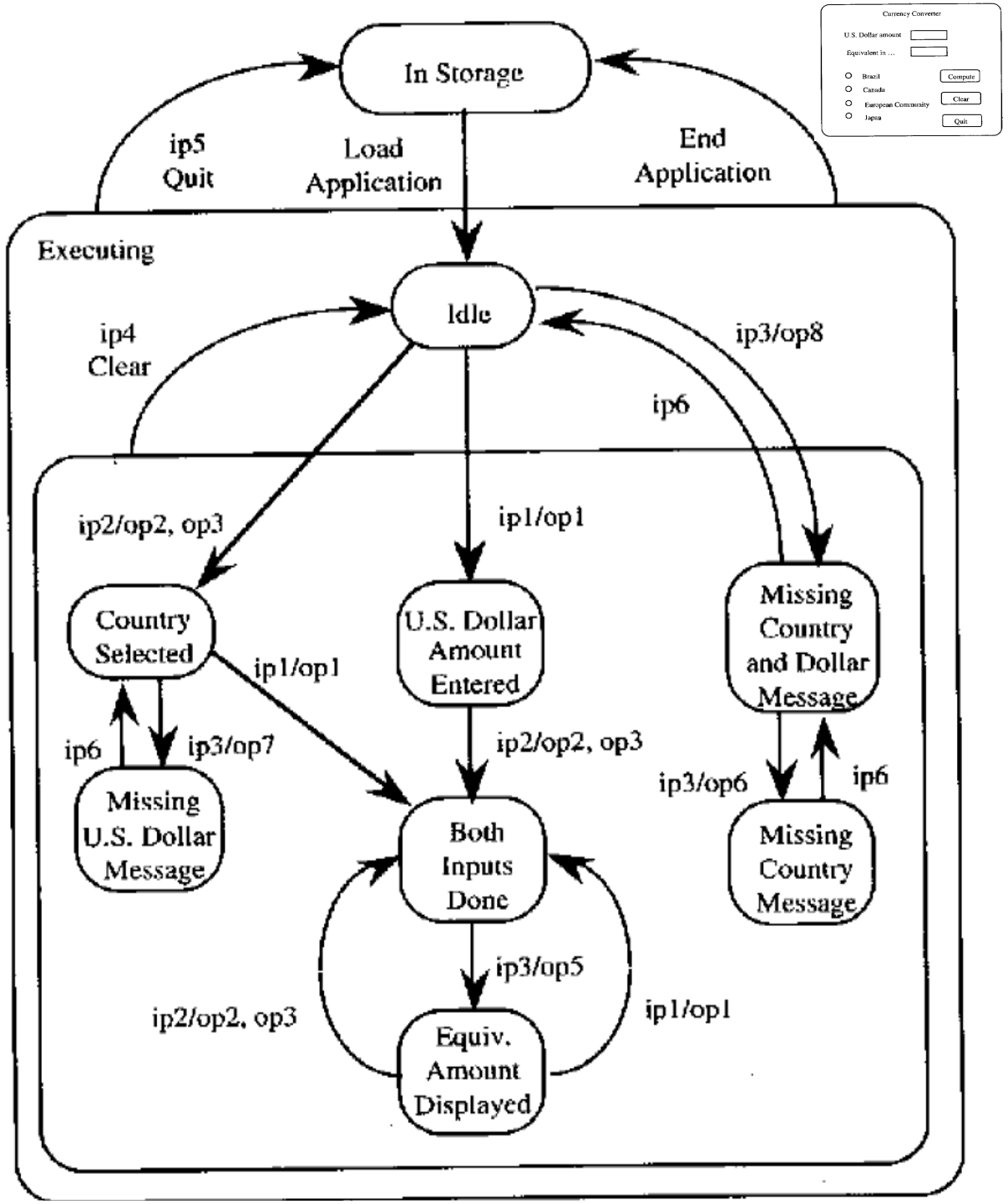


	Input Events		*Output Events*
ip1	Enter U.S. dollar amount	op2.5	Display ellipsis
ip2	Click on a country button	op3	Indicate selected country
ip2.1	Click on Brazil	op3.1	Indicate Brazil
ip2.2	Click on Canada	op3.2	Indicate Canada
ip2.3	Click on European Community	op3.3	Indicate European Community
ip2.4	Click on Japan	op3.4	Indicate Japan
ip3	Click on Compute button	op4	Reset selected country
ip4	Click on Clear button	op4.1	Reset Brazil
ip5	Click on Quit button	op4.2	Reset Canada
ip6	Click on OK in error message	op4.3	Reset European Community
	Output Events	op4.4	Reset Japan
op1	Display U.S. dollar amount	op5	Display foreign currency value
op2	Display currency name	op6	Error msg: must select a country
op2.1	Display Brazilian reals	op7	Error msg: Must enter U.S. dollar amount
op2.2	Display Canadian dollars	op8	Error msg: must select a country and enter U.S. dollar amount
op2.3	Display European Community euros	op9	Reset U.S. dollar amount
op2.4	Display Japanese yen	op10	Reset equivalent currency amount

	Input Events
ip1	Enter U.S. dollar amount
ip2	Click on a country button
ip2.1	Click on Brazil
ip2.2	Click on Canada
ip2.3	Click on European Community
ip2.4	Click on Japan
ip3	Click on Compute button
ip4	Click on Clear button
ip5	Click on Quit button
ip6	Click on OK in error message
	Output Events
op1	Display U.S. dollar amount
op2	Display currency name
op2.1	Display Brazilian reals
op2.2	Display Canadian dollars
op2.3	Display European Community euros
op2.4	Display Japanese yen
op2.5	Display ellipsis
op3	Indicate selected country
op3.1	Indicate Brazil
op3.2	Indicate Canada
op3.3	Indicate European Community
op3.4	Indicate Japan
op4	Reset selected country
op4.1	Reset Brazil
op4.2	Reset Canada
op4.3	Reset European Community
op4.4	Reset Japan
op5	Display foreign currency value
op6	Error msg: must select a country
op7	Error msg: Must enter U.S. dollar amount
op8	Error msg: must select a country and enter U.S. dollar amount
op9	Reset U.S. dollar amount
op10	Reset equivalent currency amount

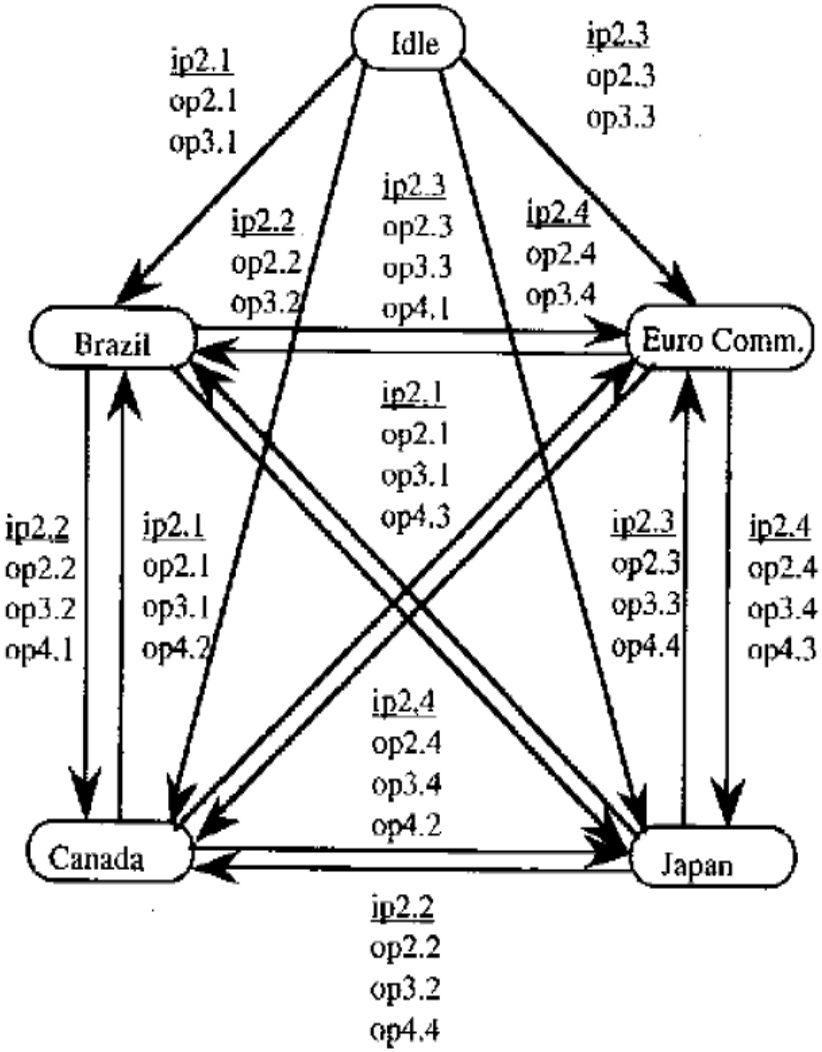
Máquina de estados finitos de alto nivel de la GUI

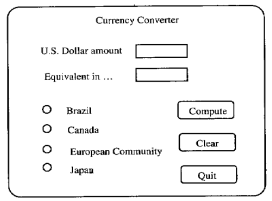
Los estados representan apariencias de la GUI visibles externamente.

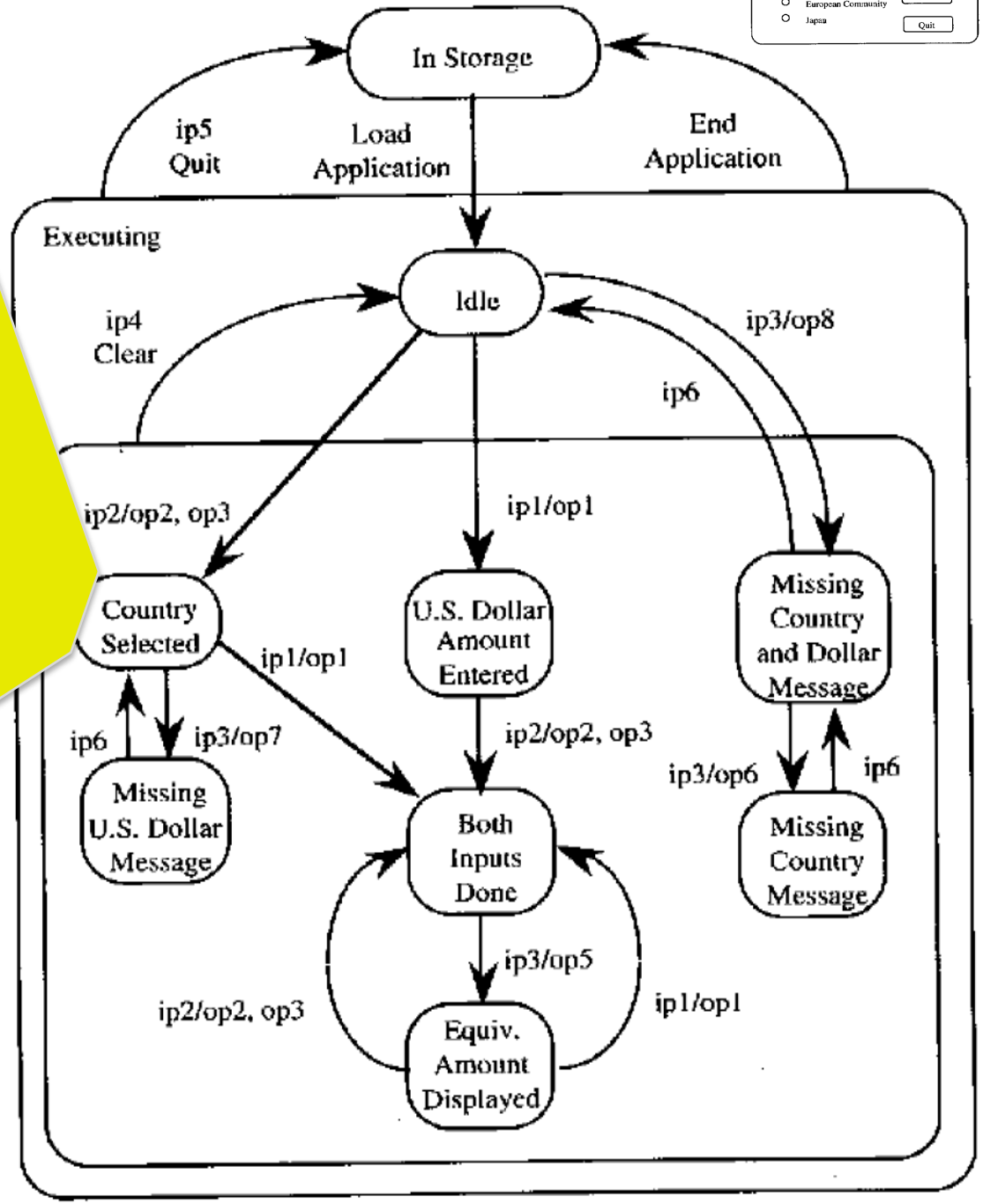


Faltaría el tratamiento de datos erróneos

FSM de la GUI







El estado Country Selected en realidad es un “macro estado” que se refiere a uno de los cuatro países seleccionados.
Los eventos (artificiales) ip2 y op2, op3 y op4 simplifican la máquina de estados

La pruebas de GUIs son complicadas

- Son **imprescindibles**, el usuario identifica la aplicación con su uso.
- Dependiendo de la complejidad (casos de uso, puntos de inicio y de final, etc.) las pruebas puede ser muy **numerosas**.
  - GUIs complicadas
  - Código de las GUIs denso, a menudo sensible a cambios de entorno
- Por distintas razones (falta de conocimiento, falta de tiempo, complejidad) las pruebas de GUIs normalmente se hacen **a mano, con efectos negativos**:
  - Las pruebas son repetitivas
  - Las pruebas manuales son extremadamente largas
  - Difícil conseguir alto nivel de cobertura
- Las pruebas, para ser efectivas, **han de automatizarse**
  - Las GUIs son diseñadas para ser usadas por personas, no por programas.
  - Las pruebas de unidad no encajan con las GUIs, estas normalmente implican más de una clase.
- Una GUI es guiada por eventos
  - El kit de pruebas debería simular eventos, esperar a que el evento llegue a un oyente de eventos, lo maneja y por último comprobar que la GUI ha respondido satisfactoriamente.
- Una GUI puede cambiar layout y apariencia sin cambiar su funcionalidad.

Pruebas de GUIs semi-automáticas

- La creación de pruebas automatizadas es más caro que las pruebas normales
- Los costes de mantenimiento son altos
- Los interfaces pueden ser bastante dinámicos
- Imponer la estabilidad del sistema para poder automatizar las pruebas puede hacerlo demasiado rígido
- Los beneficios de las pruebas de GUIs vienen tarde (cuando el sistema es estable)

Requisitos básicos para las pruebas de GUI

- Pruebas funcionales: probamos si el programa se comporta como se espera desde el punto de vista del usuario
  - Son la forma más fiable de garantizar que una GUI funciona como se espera
- No deberíamos tener que probar comportamiento de los componentes
  - Un botón debería comportarse como un botón, es trabajo del desarrollador de GUIs probar estas cosas

Construcción de GUIs

- Recomendaciones para construir GUIs “probables”:
  - Seguir el patrón de diseño Modelo-Vista-Controlador (MVC) para separar la lógica interna del aspecto visual
    - Así la lógica interna puede probarse por métodos de prueba normales independientemente de la GUI

Una buena herramienta para pruebas de GUIs

- Debe haber una forma fiable de acceder a los componentes de las GUIs
  - Cambios en posición, tamaño y layout no deben romper las pruebas
- Características que una buena biblioteca de pruebas de GUIs debería proporcionar:
  - Una API intuitiva. Debería permitir que tanto desarrolladores como probadores se centren en los escenarios de casos de uso a probar, en lugar de cómo usar las herramientas.
  - Una API concisa. Fácil de leer, de forma que las pruebas son cortas y legibles, y haciendo que el mantenimiento sea menos costoso.
  - Información útil. Debería proporcionar suficiente información sobre al GUI probada de forma que se garantice la localización de los fallos detectados.
- Extra:
  - Debería soportar pruebas
    - grabadas (scripts record/playback) y
    - programadas.

Herramienta	Comercial	Grabación	Orientación	Integrado con JUnit	Especificidades
UISpec4J	No		Programador	Sí	Enfoque basado en proxies (permite definir pruebas antes de implementación)
FEST	No		Programador	Sí	Permite usar mock objects (pruebas funcionales)
Abbot/Costello	No	Sí	Probador		Permite especificación de pruebas con XML
Pounder	No				
Marathon	No	Sí			
Jubula	+/-	Sí	Probador		BD para almacenamiento de pruebas
QF-Test	Sí				
Testing Anywhere	Sí				
TOSCA	Sí				
IBM Rational Functional Tester (RTF)	Sí	Sí	Probador		

Métodos para las pruebas de GUIs

- Scripts Record/playback
  - Las pruebas interaccionan con una GUI existente
  - Todos los eventos generados por el usuario son grabado en un script.
  - Los desarrolladores pueden repetir el script para reproducir las interacciones con el usuario.
  - Tiempo de desarrollo de pruebas pequeño.
  - Tiempo de mantenimiento de las pruebas grande. Requiere regrabar todos los escenarios de prueba afectados por el cambio.
  - Escritos en lenguajes propietarios.
- Pruebas de GUIs programadas
  - Escritos en lenguajes OO
    - más fáciles de mantener
    - más fácilmente ajustables a cambios en la aplicación
  - Utilización de IDEs
    - más productividad
    - menos coste de mantenimiento

Problema con el record/playback

- Localización
  - Idiomas
  - Formatos locales (fechas)
- Personalización
  - Esquemas de colores, imágenes, tipos de letra, tamaño y posición de pop-ups, ...
  - Formatos preferidos (primer día de la semana, un día/semana por página, ...
  - Short cuts, pasos de diálogo opcionales, ...
- Entorno
  - Qué navegador web o gestor de ventanas
  - Disponibilidad de aplicaciones de apoyo
  - Diferentes entornos pueden llevar a diferentes comportamientos de una GUI para la misma secuencia de gestos

Pruebas de GUIs programadas

- Más fáciles y baratas de mantener
- Más difíciles de desarrollar
  - Requiere una forma de encontrar los componentes de la GUI
  - Simula las interacciones con estas componentes
  - Verifica que la salida obtenida es la esperada

Abbot (http://abbot.sourceforge.net)

- Record/playback: script basados en XML
- Robusto: utiliza varios atributos para identificar componentes de la GUI dinámicamente sin depender de posiciones (cambios en layouts no suelen romper los scripts de prueba)
- Extensión de JUnit/TestNG: permite escribir pruebas programadas.
- ComponentFinder (encuentra componentes)
- Robot (simula acciones)

Pruebas de GUIs con java.awt.Robot

- java.awt.Robot fue introducida en JDK 1.3
- Útil para probar GUIs construidas con Swing y AWT
- Posibilidad de simular eventos de usuario
  - Robot puede simular acciones del usuario controlando el ratón y el teclado del entorno de la aplicación
- java.awt.Robot es una clase de muy bajo nivel, y necesita demasiado código para simular acciones de usuario
- Pruebas muy frágiles

Abbot

- Tests “robustos”
  - Tolerancia a cambios en las posiciones y layout de componentes
- API complicada de usar

FEST
(Fixtures for Easy Software Testing)

- Biblioteca de código abierto bajo licencia Apache 2.0
- Construido sobre Abbot y JUnit/TestNG
- Facilita la creación y mantenimiento de pruebas funcionales de GUIs
- Características que lo distinguen de otros proyectos:
  - API fácil de usar (*fluent interfaces*)

```
@Test public void shouldChangeTextInTextFieldWhenClickingButton() {
  naive.button("byeButton").click();
  naive.label("messageLabel").requireText("Bye!");
}
```

  - Métodos-aserto para comprobar el estado de los componentes de la GUI
  - Puede generar capturas de pantalla de pruebas fallidas
  - Integración con herramientas para mocking
  - Proporciona facilidades para acceso reflexivo
  - Extensiones a Groovy, Selenium, ...

Localización de componentes

- Se necesitan identificadores únicos para cada componente
  - El texto mostrado en un componente no debería utilizarse para identificarlo (internacionalización, evolución, ...)
  - Aunque complicado, sería posible identificarlo por tipo.
- FEST tiene tres formas de localizar componentes:
  - Por nombre

```
JButton okButton = new JButton("OK");
okButton.setName("ok");
```

```
JButtonFixture button = frame.button(”ok");
```

  - Por tipo

```
JButtonFixture button = frame.button();
```

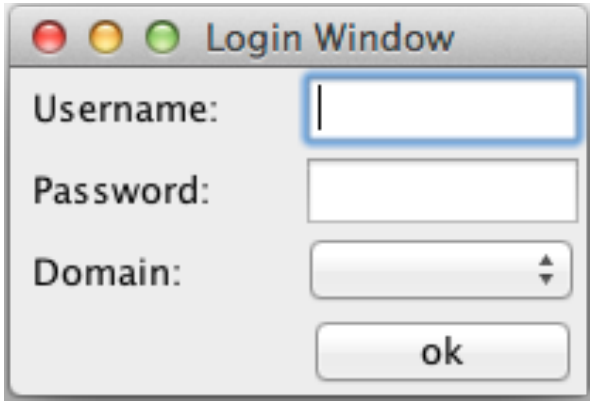
  - Específica

```
GenericTypeMatcher<JButton> textMatcher =
   new GenericTypeMatcher<JButton>(JButton.class) {
     @Override protected boolean isMatching(JButton button) {
       return "OK".equals(button.getText());
   }
};
```

```
frame.button(withName("cancel").andText("Cancel")).click();
```

Desarrollo de GUIs guiado por pruebas con FEST

- Supongamos que queremos desarrollar una GUI con el siguiente aspecto:



- Comportamiento esperado:
  - El usuario introduce su username y password
  - El usuario selecciona el dominio al que quiere conectarse de entre los ofrecidos en el desplegable
  - Si uno de los campos se deja en blanco se abre una ventana de diálogo notificando al usuario de que la información que falta es necesaria

Creamos nuestras pruebas con FEST

- Primero creamos nuestra clase de pruebas

```
public class LoginWindowTest {

}
```

Creamos nuestras pruebas con FEST

- Activamos la comprobación automática de que todos los componentes Swing son actualizados a través de la EDT de Swing (Event Dispatcher Thread)

```
@BeforeClass
public static void setUpOnce() {
    FailOnThreadViolationRepaintManager.install();
}
```

Con esto FEST comprobará que la EDT será usada correctamente.

Creamos nuestras pruebas con FEST

- Como JUnit se ejecuta en su propio thread, debemos lanzar nuestro frame o dialog desde FEST, para asegurarnos de que se usa bien la EDT.

```
@Before
public void setUp() {
    frame = new FrameFixture(
                GuiActionRunner.execute(
                    new GuiQuery<JFrame>() {
                        protected JFrame executeInEDT() {
                            return new LoginFrame();
                        }
                    }));
    frame.show();
    // testFrame.robot.settings().delayBetweenEvents(500);
}
```

Creamos nuestras pruebas con FEST

- Al acabar nos aseguramos de descargar el frame y los recursos utilizados.

```
@After
public void tearDown() {
    frame.cleanUp();
}
```

Creamos nuestras pruebas con FEST

- Ya estamos listos para crear nuestras pruebas.
- Comprobemos que al pulsar el botón ok con el campo username vacío se abre una ventana de diálogo indicando la situación.

```
@Test
public void shouldShowErrorIfUsernameIsMissing() {
    frame.textBox("username").deleteText();
    frame.textBox("password").enterText("secret");
    frame.comboBox("domain").selectItem("USERS");
    frame.button("ok").click();
    frame.optionPane().requireErrorMessage()
            .requireMessage("Please  enter your username");
}
```

Creamos nuestras pruebas con FEST

- Ahora podemos ejecutar nuestras pruebas como siempre.
- Es aconsejable no hacer nada en el ordenador mientras se hacen las pruebas.

Selenium 2.0: Breve historia

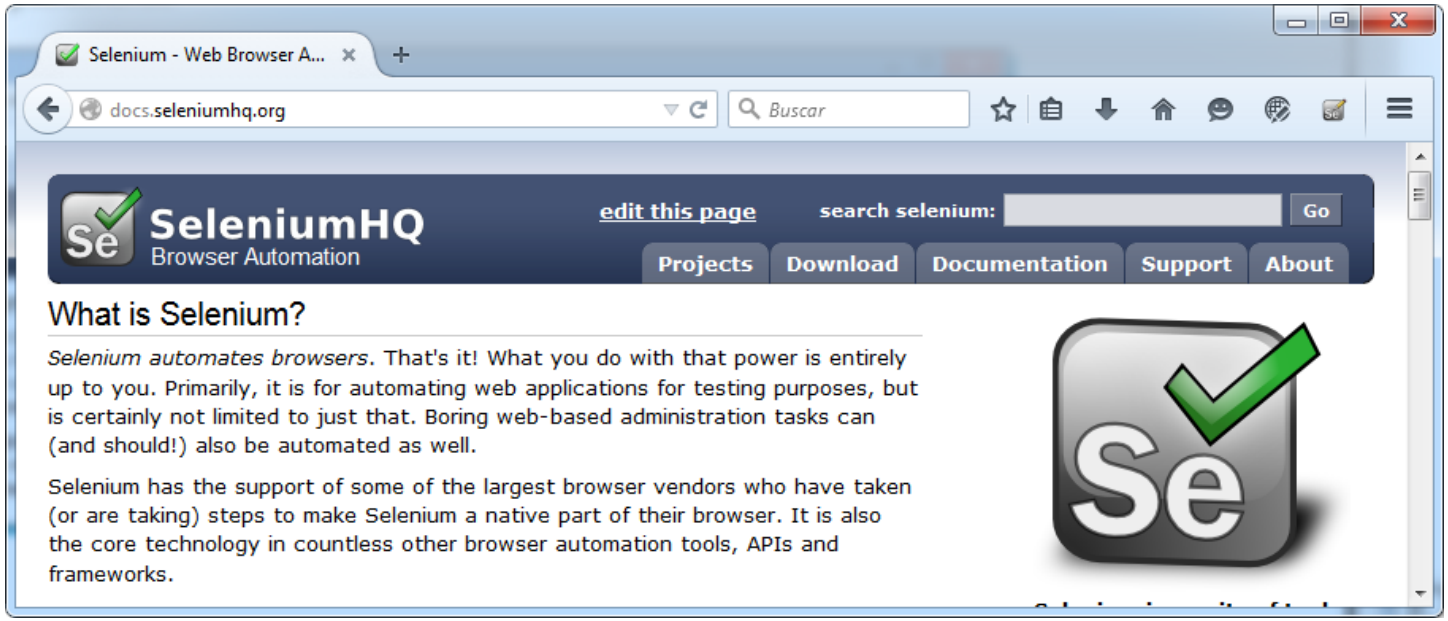
- **Selenium** es un conjunto de diferentes herramientas cada una con una aproximación distinta para soportar la automatización de las pruebas.
- Selenium aparece por primera vez en **2004** cuando Jason Huggins estaba testeando una aplicación interna. Desarrolló una librería javascript que le permitía reejecutar automáticamente los tests en múltiples navegadores.
- Debido a su motor de automatización **Javascript** y las limitaciones de seguridad que los navegadores aplican a Javascript, había algunas acciones que Selenium no podía ejecutar.
- Aunque en Google se utilizó intensivamente Selenium, los testers tenían que utilizar **"work arounds"** para evitar las limitaciones del producto.
- En 2006, Simon Stewart, un ingeniero de Google, comenzó a trabajar en un proyecto que él llamó **"WebDriver"**. Simon quería construir una herramienta de testing que se comunicara directamente con el navegador, evitando las restricciones de un entorno Javascript.
- En 2008, ambos proyectos, Selenium y WebDriver, se unieron. Selenium tenía una enorme comunidad de usuarios y ofrecía soporte comercial pero se apostó a que WebDriver era la herramienta del futuro.

Selenium Tool Suite: http://seleniumhq.org

Selenium IDE

Selenium WebDriver

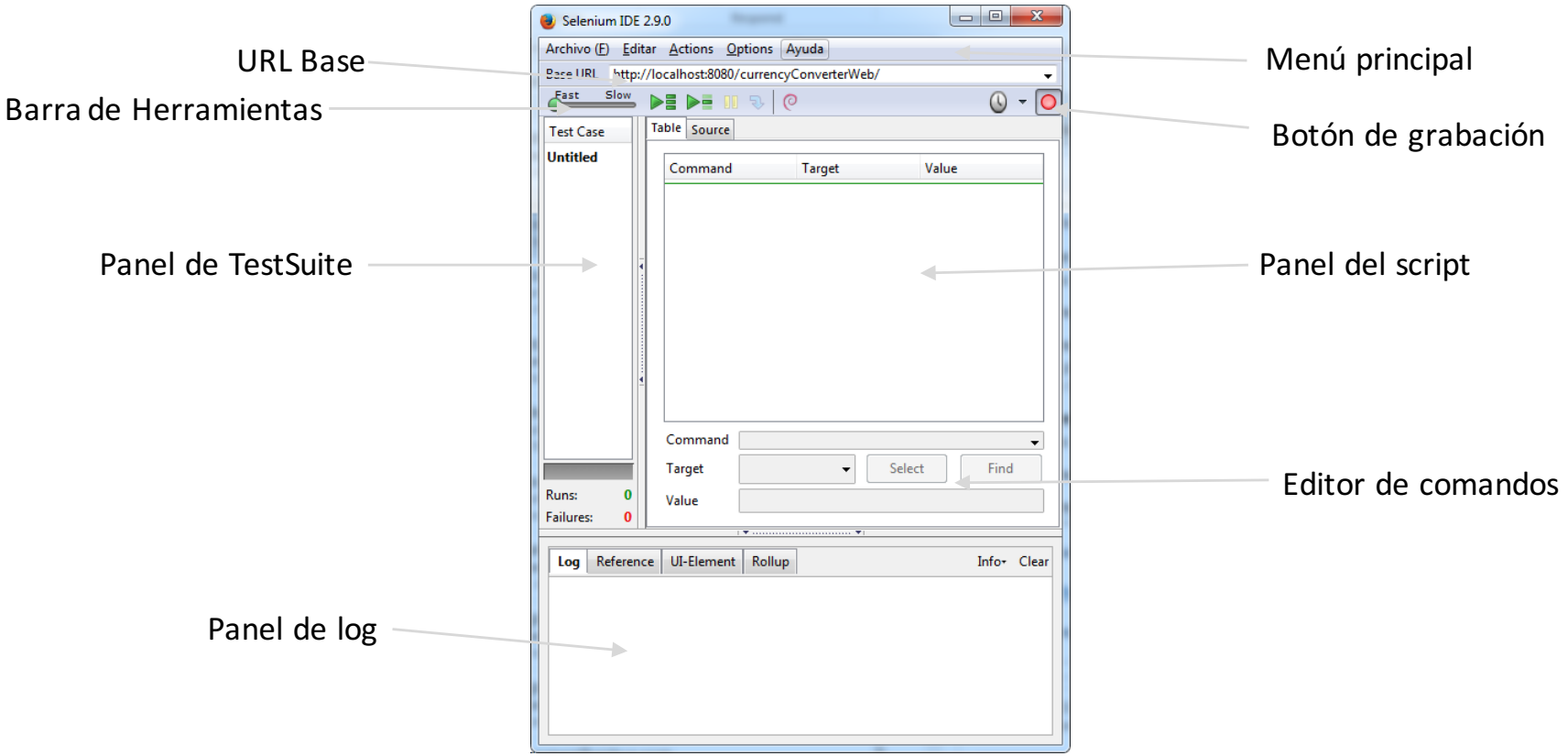
Selenium Grid



Selenium IDE

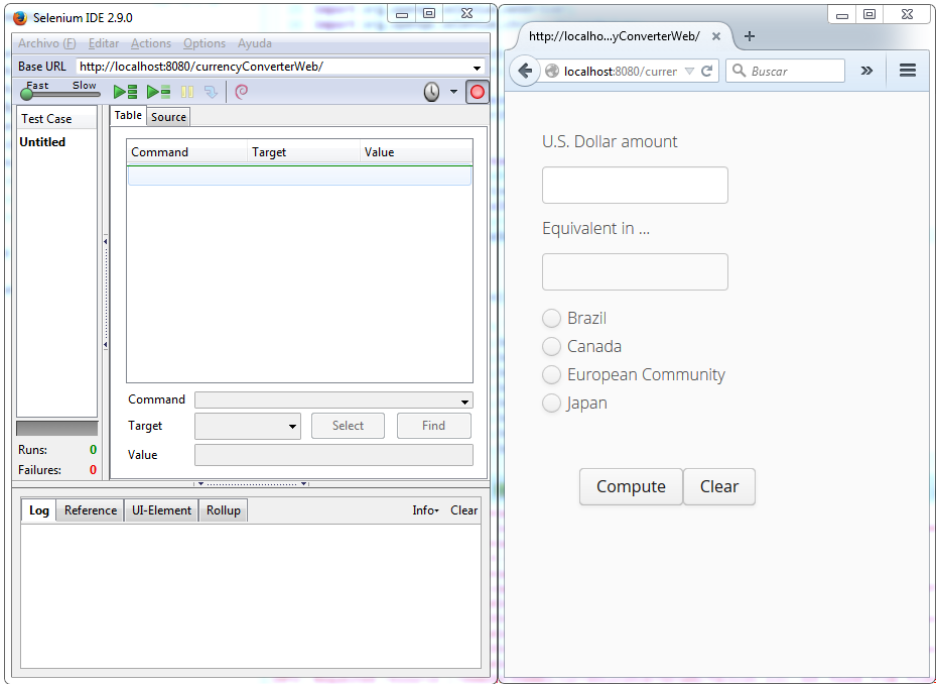
- Se trata de un plugin para Firefox para construir scripts de prueba.
- Tiene una funcionalidad de grabación que registra las acciones del usuario.
- Puede grabar las acciones registadas para volver a ejecutarlas.
- También se pueden crear los scripts con el asistente que tiene integrado.
- Puede exportar las pruebas a varios lenguajes para que sean ejecutados con Selenium WebDriver.

Selenium IDE | Selenium WebDriver | Selenium Grid



Ejecución

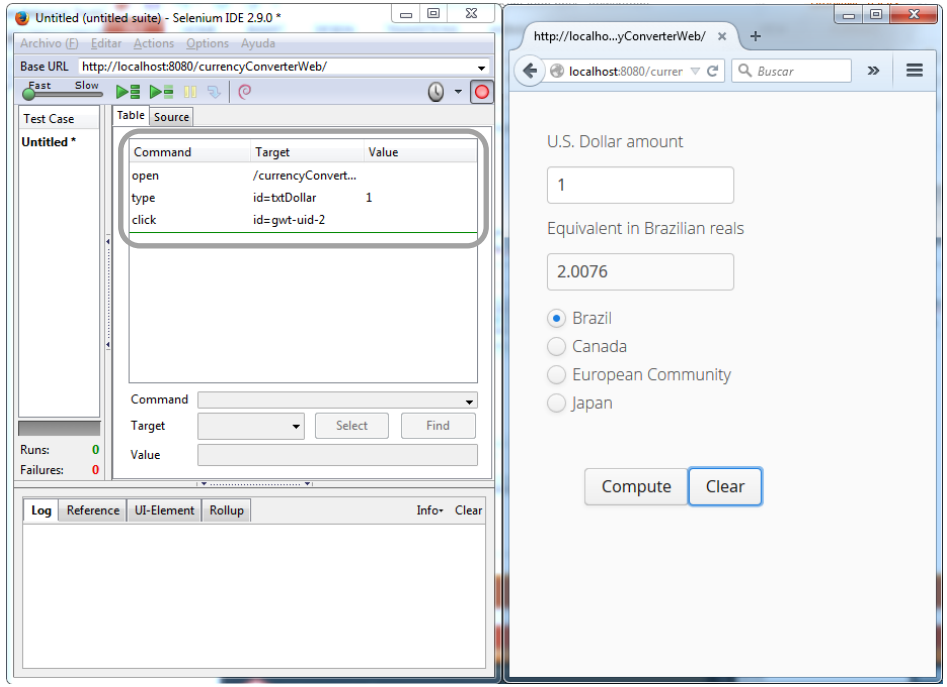
Selenium IDE | Selenium WebDriver | Selenium Grid



Abrimos Selenium IDE y la aplicación web que queremos probar.

Ejecución

Selenium IDE | Selenium WebDriver | Selenium Grid

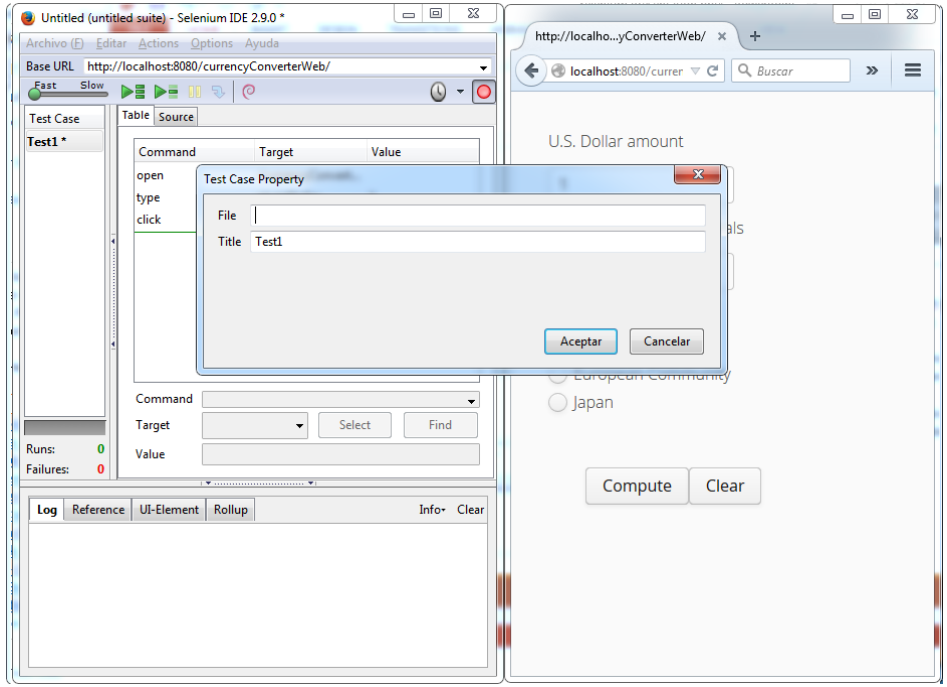


Abrimos Selenium IDE y la aplicación web que queremos probar.

Cuando empezamos a realizar acciones Selenium IDE las va registrando.

Ejecución

Selenium IDE | Selenium WebDriver | Selenium Grid



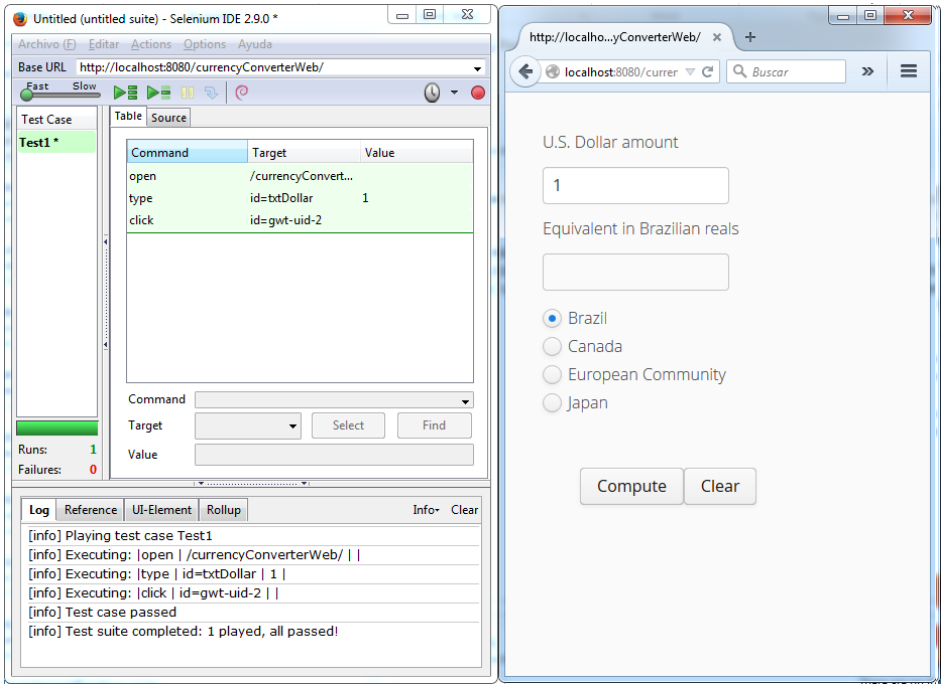
Abrimos Selenium IDE y la aplicación web que queremos probar.

Cuando empezamos a realizar acciones Selenium IDE las va registrando.

Podemos renombrar el test pulsando con el botón derecho sobre el nombre y accediendo a propiedades.

Verificación

Selenium IDE | Selenium WebDriver | Selenium Grid



Abrimos Selenium IDE y la aplicación web que queremos probar.

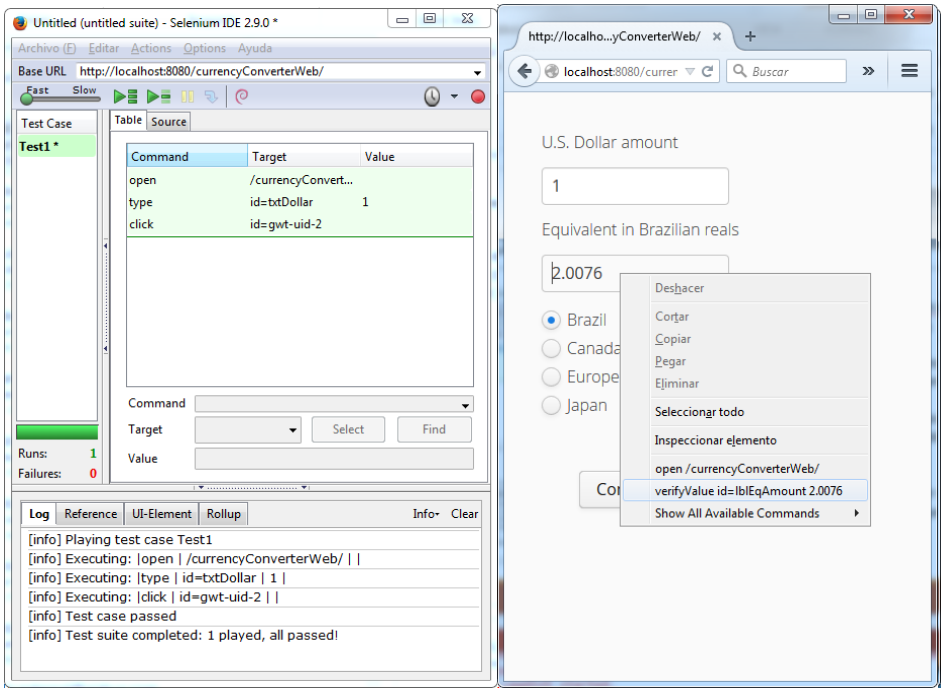
Cuando empezamos a realizar acciones Selenium IDE las va registrando.

Podemos renombrar el test pulsando con el botón derecho sobre el nombre y accediendo a propiedades.

Podemos ejecutar el script que hemos creado todas las veces que queramos.

Verificación

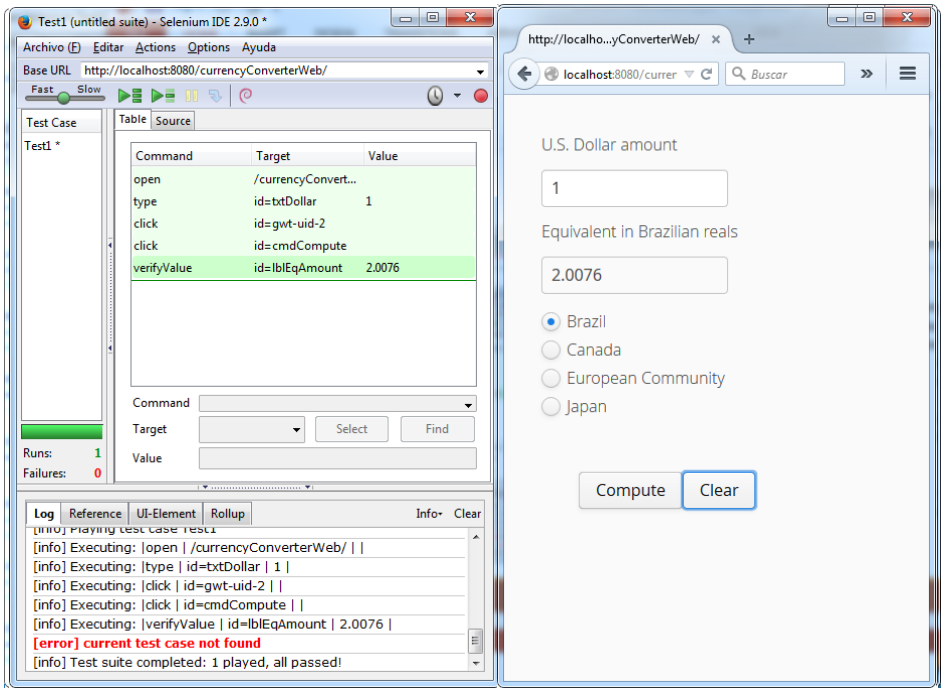
Selenium IDE | Selenium WebDriver | Selenium Grid



Podemos añadir verificaciones al script simplemente pulsando con el botón derecho en el campo a verificar.

Verificación

Selenium IDE | Selenium WebDriver | Selenium Grid

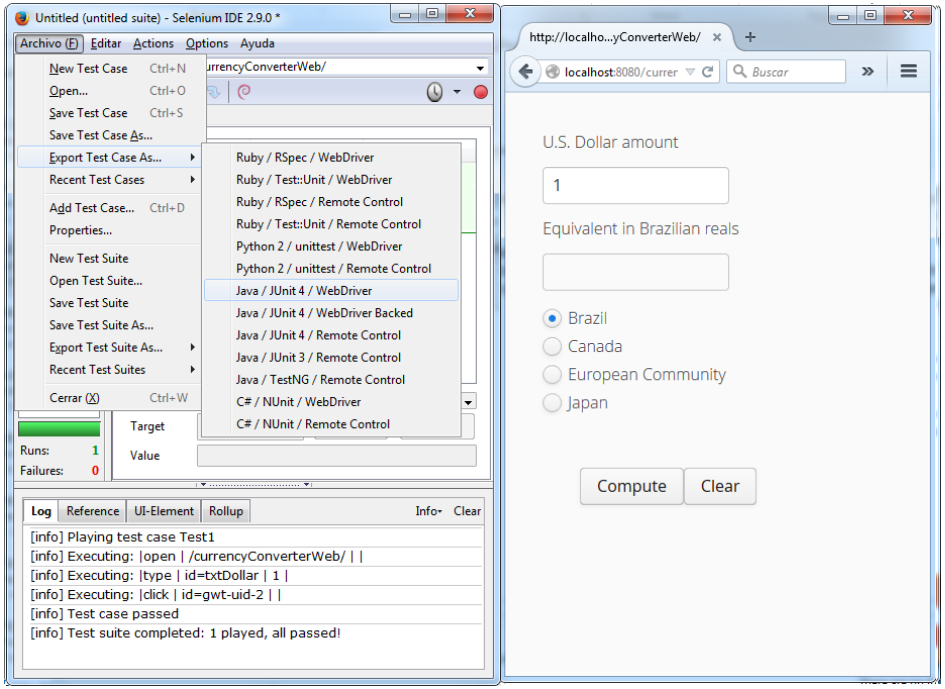


Podemos añadir verificaciones al script simplemente pulsando con el botón derecho en el campo a verificar.

Al ejecutar el script marcará si la verificación se ha realizado correctamente.

Exportación

Selenium IDE | Selenium WebDriver | Selenium Grid



Desde el menú de archivo podemos exportar el Test Case (o todo el Test Suite) al lenguaje que nos interese.

Comandos

Selenium IDE | Selenium WebDriver | Selenium Grid

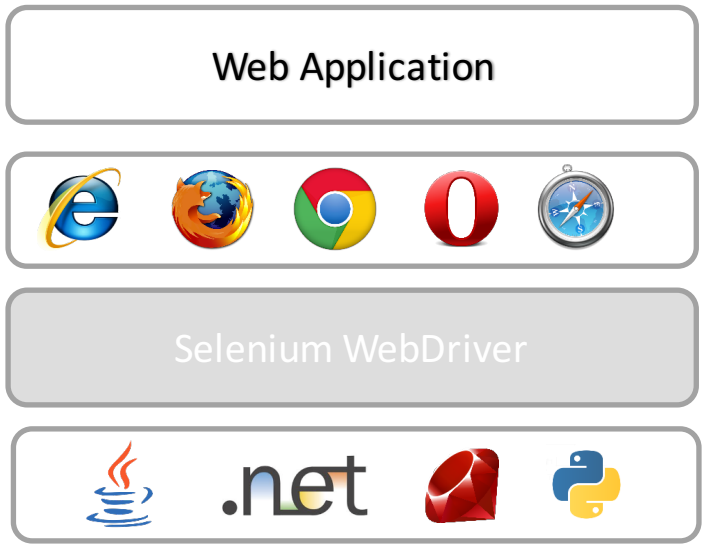
- **open:** Abre una página usando una URL.
- **click/clickAndWait:** Ejecuta un click, y opcionalmente espera a que la nueva página se cargue.
- **verifyTitle/assertTitle:** Verifica el título de la página esperado.
- **verifyElementPresent:** Verifica que un element de la UI está presente, tal y como está definido en su tag HTML.
- **waitForPageToLoad:** Pausa la ejecución hasta que la nueva página estña cargada. Se llama automáticamente cuando usamos clickAndWait.
- **waitForElementPresent:** Pausa la ejecución hasta que el element del UI indicado está presente.

Selenium IDE | **Selenium WebDriver** | Selenium Grid

- Es el resultado de la unión de los proyectos Selenium y WebDriver.
- Proporciona una API de programación para desarrollar scripts de prueba usando diferentes lenguajes de programación.
- Se pueden ejecutar los test en los distintos navegadores soportados por Selenium.
- Los scripts se pueden generar en Java, .Net, Ruby o Python.
- Soporta los navegadores Mozilla Firefox, Google Chrome, MS Internet Explorer, Safari y Opera.

Arquitectura

Selenium IDE | **Selenium WebDriver** | Selenium Grid



- WebDriver proporciona una API orientada a objetos para realizar pruebas de aplicaciones web.
- Se comunica con cada uno de los navegadores mediante drivers específicos.
- WebDriver hace llamadas directas al navegador, utilizando soporte nativo de este.
- Se apoya en los distintos lenguajes que soporta para la generación de los scripts de prueba.

Drivers

Selenium IDE | **Selenium WebDriver** | Selenium Grid

- **HtmlUnit Driver:** Es la implementación más rápida y ligera de WebDriver. Se basa en HtmlUnit (una implementación Java de un navegador sin GUI).
- **Firefox Driver:** Controla el navegador Firefox usando un plugin. Funciona en Windows, Mac y Linux.
- **Internet Explorer Driver:** Se controla mediante .dll y solo está disponible para Windows.
- **Chrome Driver:** Se interactúa con Chrome a través de chromedriver, mantenido y soportado por el Proyecto Chromium.
- **Otros drivers:** Opera Driver, iPhone Driver y Android Driver.

```
WebDriver driver = new FirefoxDriver();

driver.quit();
```

Cargar una página

Selenium IDE | **Selenium WebDriver** | Selenium Grid

```
driver.get("http://www.google.es");
```

- El comportamiento de get depende del SO y del navegador utilizados.
- WebDriver puede devolver el control antes de que la página haya sido cargada, siendo en algunos casos necesario introducir esperas explícitas o implícitas.

Localización de componentes (WebElements)

Selenium IDE | **Selenium WebDriver** | Selenium Grid

- Podemos localizar elementos en un WebDriver o en un WebElement
- Podemos localizar (findElement / findElements)
  - Un objeto WebElement (lanza una excepción si no se localiza)
  - Una lista de WebElements (vacía si no se encuentra ningún elemento)

Selenium IDE | Selenium WebDriver | Selenium Grid

Localizadores

- **Por ID:**
  - ...
  - WebElement element = driver.findElement(**By.id("id1")**);
- **Por TagName:**
  - <iframe src="..."></iframe>
  - WebElement frame = driver.findElement(**By.tagName("iframe")**);
- **Por nombre:**
  - <input name="cheese" type="text"/>
  - driver.findElement(**By.name("cheese")**);
- **Por XPath:**
  - <input type="text" name="example" />
    <input type="text" name="other" />
  - List<WebElement> inputs = driver.findElements(**By.xpath("//input")**);
  - WebDriver usa las capacidades Xpath nativas del navegador cuando es posible.
  - Para los navegadores que no tienen soporte de XPath usa su propia implementación.
- **Usando Javascript:**
  - WebElement element = (WebElement) ((JavascriptExecutor)driver).executeScript("return $('.cheese')[0]");
  - findElement(By.name("cheese"));
- **Por CSS:**
  - Usando CSS Selectors (http://www.w3.org/TR/CSS/#selectors)
  -
    milk
    cheese

  - WebElement cheese = driver.findElement(**By.cssSelector("#food span.dairy.aged")**);
- **Por Link Text.**
  - WebElement cheese = driver.findElement(**By.linkText("Go to")**);
  - WebElement cheese = driver.findElement(**By.partialLinkText("Go")**);

Acciones sobre WebElements

Selenium IDE | **Selenium WebDriver** | Selenium Grid

- sendKeys(String) — `f.sendKeys("15");`
  - introduce texto en un área o campo de texto
- click() — `f.click();`
- getText() — `f.getText();`
- getAttribute() / setAttribute()
- submit()
  - para enviar formularios (forms), en cualquier elemento del formulario
- switch()
  - para cambiarnos entre frames, windows o pop-up windows
- navigate()
  - para movernos entre páginas to(String), forward(), backward()
- gestionar cookies, profiles, drag and drop, ...

Seleccionar una opción de un desplegable

Selenium IDE | Selenium WebDriver | Selenium Grid

```
WebElement units = driver
      .findElement(By.xpath("//*[@id=\"aspnetForm\"]/div[5]/table/tbody/tr[1]/td[2]/select"));
Iterator<WebElement> it = units.findElements(By.tagName("option")).iterator();
boolean found = false;
WebElement option = null;
while (!found && it.hasNext()) {
      option = it.next();
      found = option.getAttribute("value").equals("km");
}
if (found) option.click();
```

o

```
Select select = new
Select(driver.findElement(By.name("ctl00$PlaceHolderMain$g_6b565c00_7104_4062_860f_70b4737c5346$ctl00")));
select.selectByVisibleText("s");
```

Esperas explícitas

Selenium IDE | **Selenium WebDriver** | Selenium Grid

- WebDriverWait(driver, timeout)
  - driver que se pasa al gestor de condiciones
  - timeout en segundos
- ExpectedConditions
  - modela condiciones booleanas
    - presenceOfElementLocated, alertIsPresent, elementToBeClickable, ...

```
WebDriverWait webDriver = new WebDriverWait(driver, 10);
WebElement elt = webDriver.until(
        ExpectedConditions.presenceOfElementLocated(
            By.id("myDynamicElement")));
WebElement elt = webDriver.until(
        ExpectedConditions.elementToBeClickable(
            By.id("myDynamicElement")
```

Devuelve el elemento buscado si lo encuentra en menos de 10 segundos, o lanza una TimeoutException en caso contrario

Esperas implícitas

Selenium IDE | **Selenium WebDriver** | Selenium Grid

- Espera un tiempo al buscar elementos si no están disponibles.
- Por defecto la espera es 0.

```
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

Java Project

Selenium IDE | Selenium WebDriver | Selenium Grid

- Para poder trabajar con las clases generadas desde Selenium IDE tenemos que añadir las librerías de selenium y de JUnit.
- Por ejemplo en Ivy añadiríamos las siguientes líneas en ivy.xml

```
<dependency org="org.seleniumhq.selenium" name="selenium-java" rev="2.45.0"/>
<dependency org="junit" name="junit" rev="4.12"/>
```

Java Project

Selenium IDE | Selenium WebDriver | Selenium Grid

```
@Before
public void setUp() throws Exception {
  driver = new FirefoxDriver();
  baseUrl = "http://localhost:8080/currencyConverterWeb/";
  driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
}
```

- Configuramos el navegador que vamos a usar en el método @:
  - Para cada navegador tenemos un Driver diferente (ChromeDriver, InternetExplorerDriver, …).
  - Los drivers pueden instalarse desde http://docs.seleniumhq.org/download/.
  - Si queremos ejecutar con otro navegador solo hay que seleccionar el driver adecuado para nuestra prueba.

Java Project

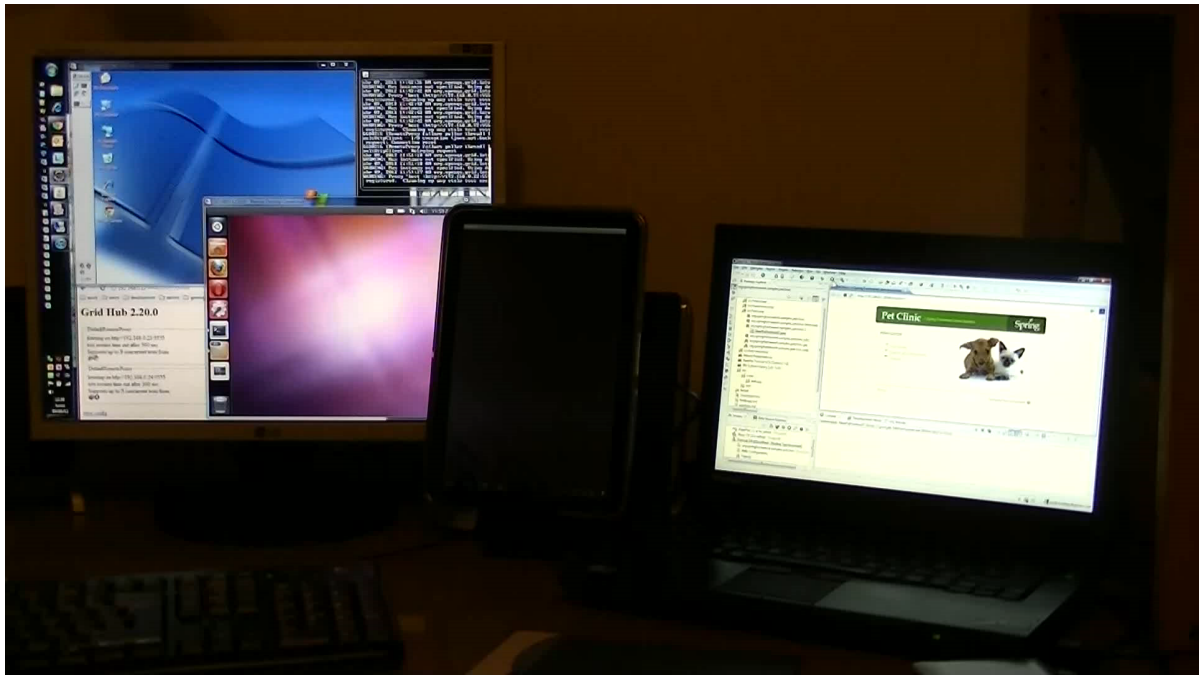
Selenium IDE | Selenium WebDriver | Selenium Grid

```
@Test
public void test1() throws Exception {
  driver.get(baseUrl + "/currencyConverterWeb/");
  driver.findElement(By.id("txtDollar")).clear();
  driver.findElement(By.id("txtDollar")).sendKeys("1");
  driver.findElement(By.id("gwt-uid-2")).click();
  driver.findElement(By.id("cmdCompute")).click();
  try {
    assertEquals("2.0076", driver.findElement(By.id("lblEqAmount")).getAttribute("value"));
  } catch (Error e) {
    verificationErrors.append(e.toString());
  }
}
```

- Permite la ejecución remota y distribuida de scripts de Selenium.
- Se pueden ejecutar scripts en paralelo y en plataformas móviles como Android o iOS.
- Se pueden ejecutar con diferentes combinaciones de Sistema Operativo y navegador.
- Es muy buena para pruebas de compatibilidad con diferentes navegadores.
- Se utiliza ampliamente para pruebas de regresión o para automatizar tareas repetitivas en aplicaciones web.

Selenium Grid

Selenium IDE | Selenium WebDriver | Selenium Grid



Referencias

- Instant Selenium Testing Tools Starter: http://it-ebooks.info/book/3123/
- http://seleniumhq.org
- https://github.com/seleniumhq/selenium
- https://seleniumhq.wordpress.com