Eusko Jaurlaritzaren Informatika Elkartea
Sociedad Informática del Gobierno Vasco

OTC-MAU-Selenium

Fecha: 02/12/2011

Referencia:

EJIE S.A.
Mediterráneo, 14
Tel. 945 01 73 00*
Fax. 945 01 73 01
01010 Vitoria-Gasteiz
Posta-kutxatila / Apartado: 809
01080 Vitoria-Gasteiz
www.ejie.es

Control de documentación

Título de documento: OTC-MAU-Selenium v0.4.doc

Histórico de versiones

Código:	Versión:	Fecha:	Resumen cambios:
	0.1	07/04/2011	Primera Versión.
	0.2	11/04/2011	Se añaden recomendaciones
	0.3	08/11/2011	Se recomienda tracear la prueba ante errores
	0.4	02/12/2011	Nuevo parámetro otc.proyecto.url.base con dominio de la aplicación
	0.5	12/03/2013	Uso de WebDriver en vez de Selenium Grid. Punto 6.

Cambios producidos desde la última versión

Uso de WebDriver en vez de Grid. Punto 6.

Control de difusión

Responsable: Ander Martínez

Aprobado por:

Firma: Fecha:

Distribución:

Referencias de archivo

Autor: Consultoría de Áreas del Conocimiento

Nombre archivo: OTC-MAU-Selenium v0.5.doc

Localización:

Contenido

1. Introducción

Selenium es un conjunto de herramientas que permiten registrar la navegación realizada en una aplicación para su ejecución automatizada, utilizando para ellos varios formatos.

- **Selenium IDE**: es un plugin de Firefox que permite grabar la navegación que constituye la prueba en formato JUnit (entre otros).
- **Selenium RC**: (remote control) es un proceso que escucha peticiones de prueba y las ejecuta utilizando navegadores disponibles en su entorno.
- **Selenium Hub**: es un proceso encargado de distribuir las peticiones de prueba entre los RC que controle.

La versión homologada para las aplicaciones albergadas en ejie es la 1.0.1.

Para más información se puede visitar la página oficial del producto:
http://Seleniumhq.org/

2. Selenium-IDE

2.1 Grabar pruebas funcionales

Gracias a esta herramienta se podrán implementar de manera sencilla pruebas funcionales de los aplicativos, permite grabar navegaciones simples y añadir otros comandos sobre un elemento de la interfaz web de usuario.

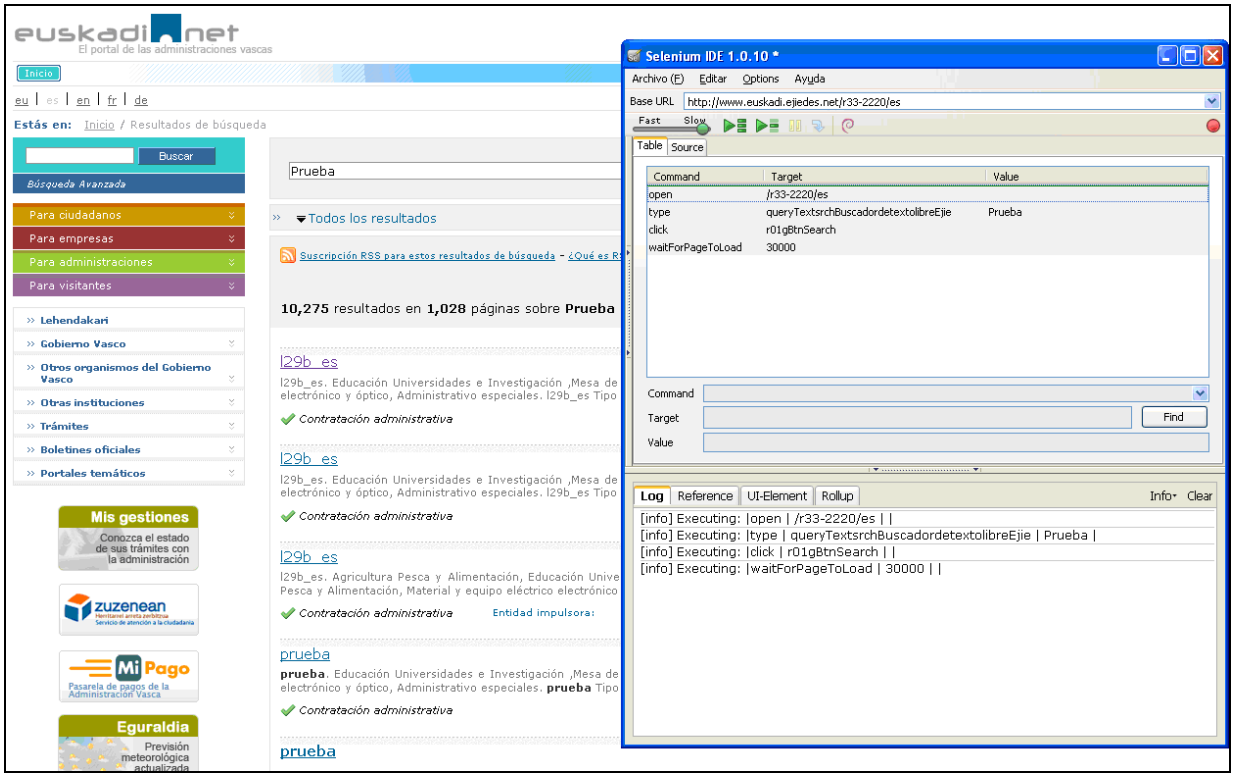


Figura 1: Grabando el script

No todas las acciones se capturan automáticamente, durante la grabación Selenium-IDE, entre otros, insertará comandos como:

- "Clicks" sobre enlaces → Comando *"Link"*.
- Entradas de valores → Comando *"Types"*
- Selección de combos → Comando *"Select"*
- "Clicks" sobre checkBoxes → Comando *"Check"*

Hay que tener en cuenta además que:

- El comando "Type" requiere seleccionar algún área de la interfaz de usuario.

- Es probable que en algunos casos deba cambiarse un comando “click” generado por otro de tipo “clickandwait” para evitar que se produzcan errores inesperados al intentar acceder a un elemento que todavía no se haya cargado,

2.2 Exportar de Selenium-IDE a pruebas java para Junit

Selenium-IDE permite además exportar los casos de prueba a distintos lenguajes y estándares definidos, de esta manera, para las aplicaciones albergadas en Ejie se exportarán los resultados en Junit 3 para WL8.1 y JUnit 4 para wl11.

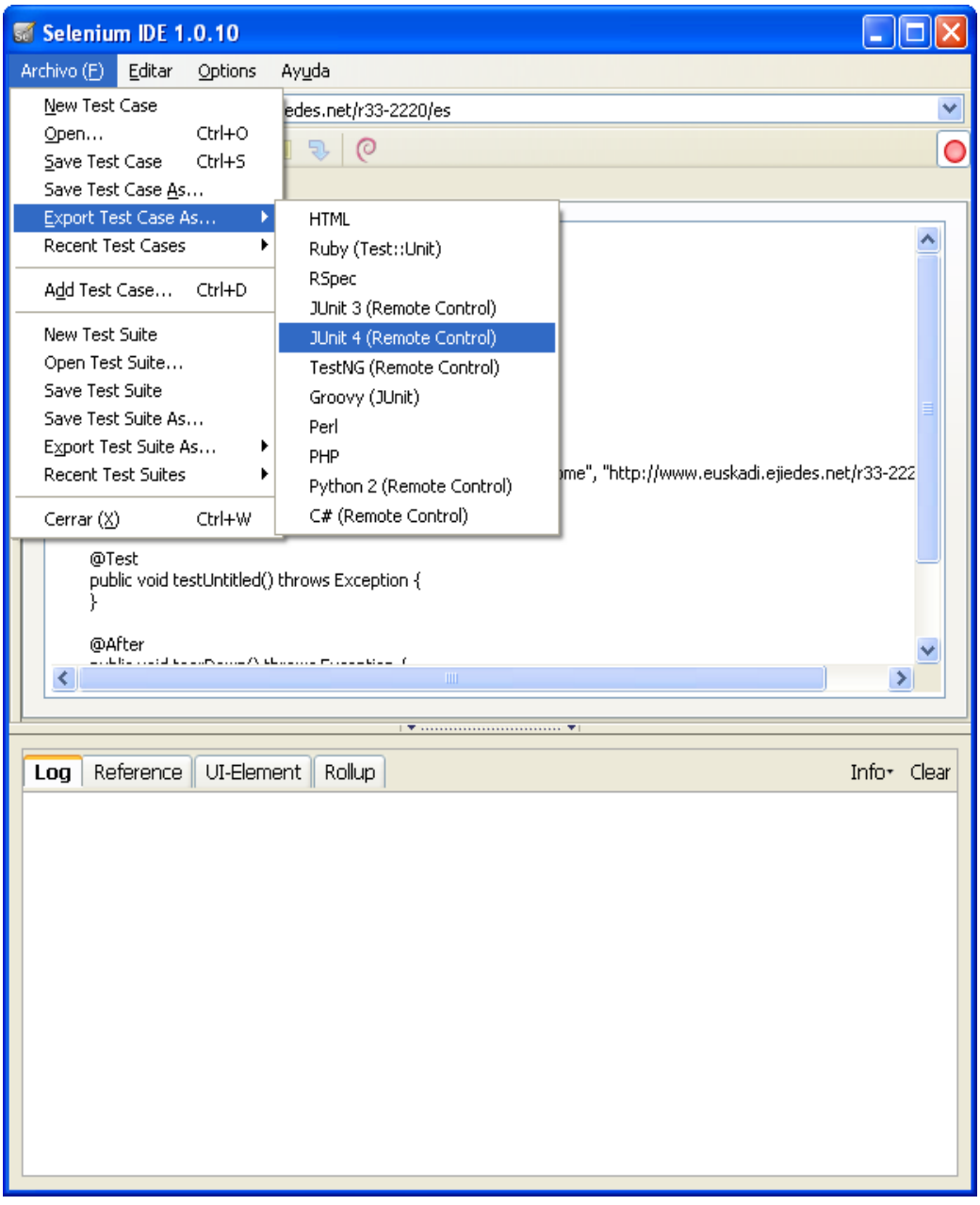


Figura 2: Exportar a java JUnit

2.3 Reutilizar navegación para grabar VUgen.

En algunos casos puede ser interesante grabar una misma navegación para las dos herramientas definidas para la ejecución de pruebas funcionales, Selenium en desarrollo y LoadRunner en pruebas.

En este apartado se explicarán los pasos para reutilizar esa misma navegación

1. Accedemos al VuGen y creamos un nuevo script

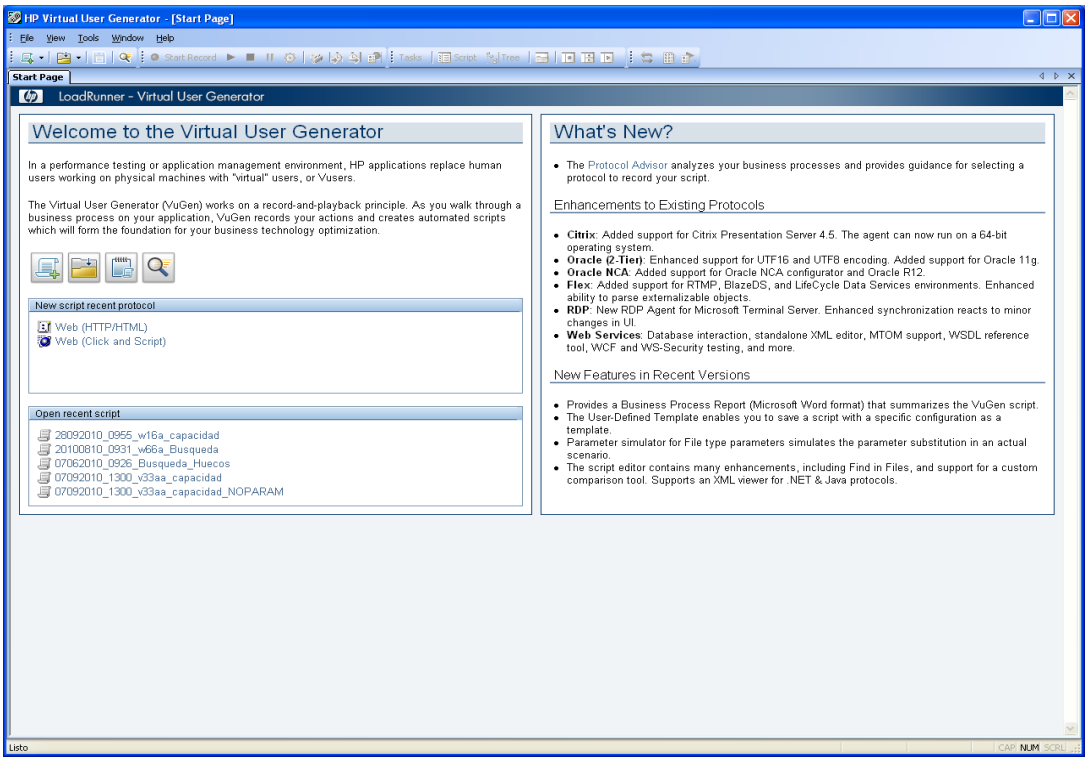


Figura 3: Inicio VuGen

2. En las opciones disponibles, seleccionamos un script de Web (HTTP/HTML) y pulsamos create

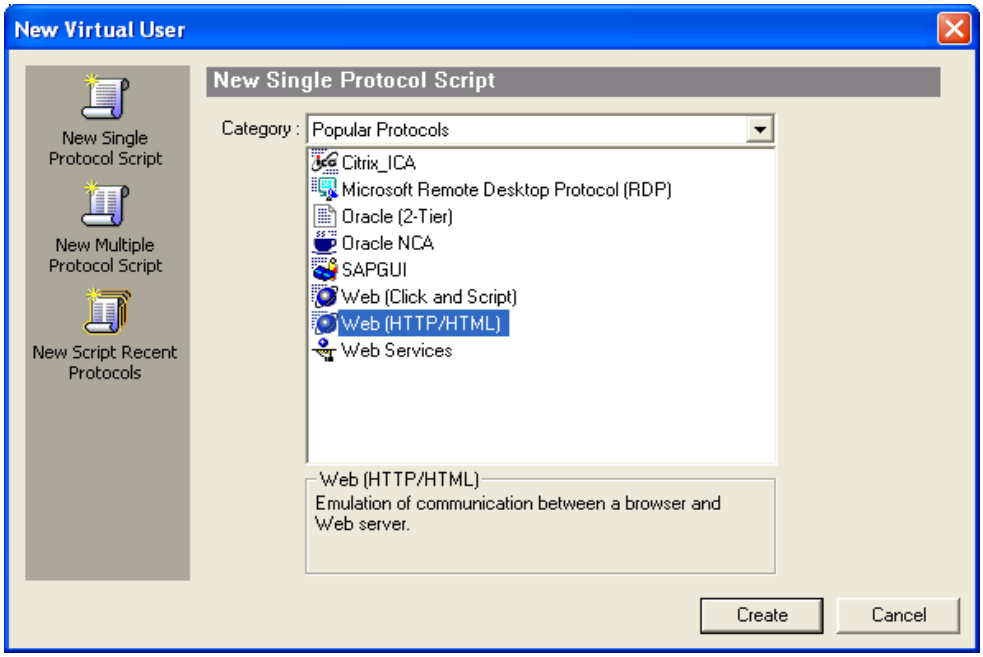


Figura 4: New virtual user

3. En la siguiente ventana, en el campo program to record, pulsamos en el botón de la derecha y buscamos el ejecutable de Firefox (el plugin solo esta disponible para firefox)

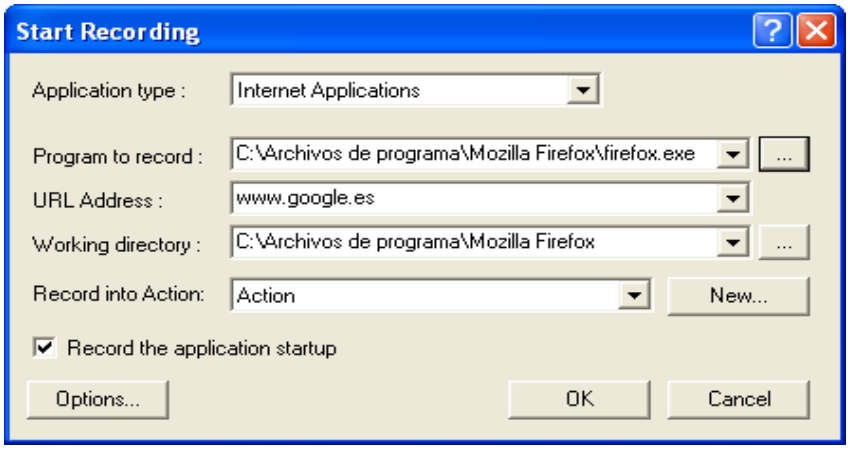


Figura 5: selección carpeta

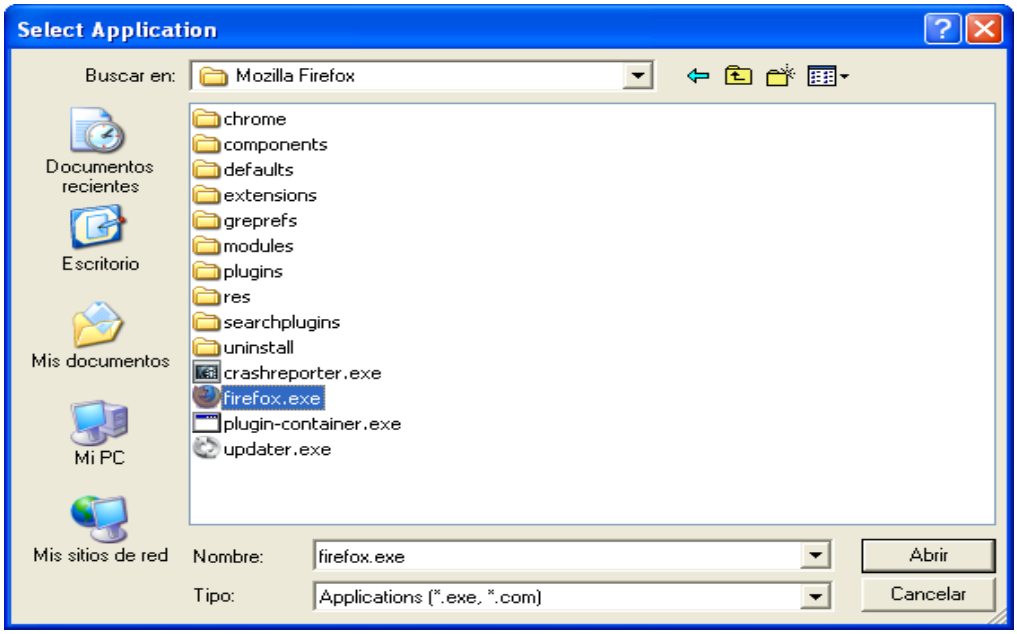


Figura 6: Selección carpeta

4. Después pulsamos ok y hacemos la grabación con VuGen normalmente, se nos abrirá el firefox en lugar del Explorer, y en la parte izquierda tenemos Selenium IDE (para que salga, previamente habrá que haberlo seleccionado en ver –> panel lateral). Como se ve en la imagen, a medida que vugen va grabando sus eventos, Selenium IDE hace lo mismo.

Figura 7: Verificación de grabación

3. Desarrollo y ejecución de casos de pruebas

Este punto está deprecado, y se recomienda usar Web Driver (ver punto 6 de este mismo documento).

Para aplicaciones desarrolladas con lenguaje java, Selenium proporciona un cliente con el que implementar la navegación que se define en el caso de prueba.

Se podrá partir de código generado por Selenium-IDE, teniendo en cuenta que la herramienta solo graba alguno de los pasos y que con toda probabilidad haya que hacer modificaciones posteriores del código.

El cliente implementado envía al servidor de Selenium las acciones a través de comando que forman parte de su api:

http://release.Seleniumhq.org/Selenium-core/1.0.1/reference.html

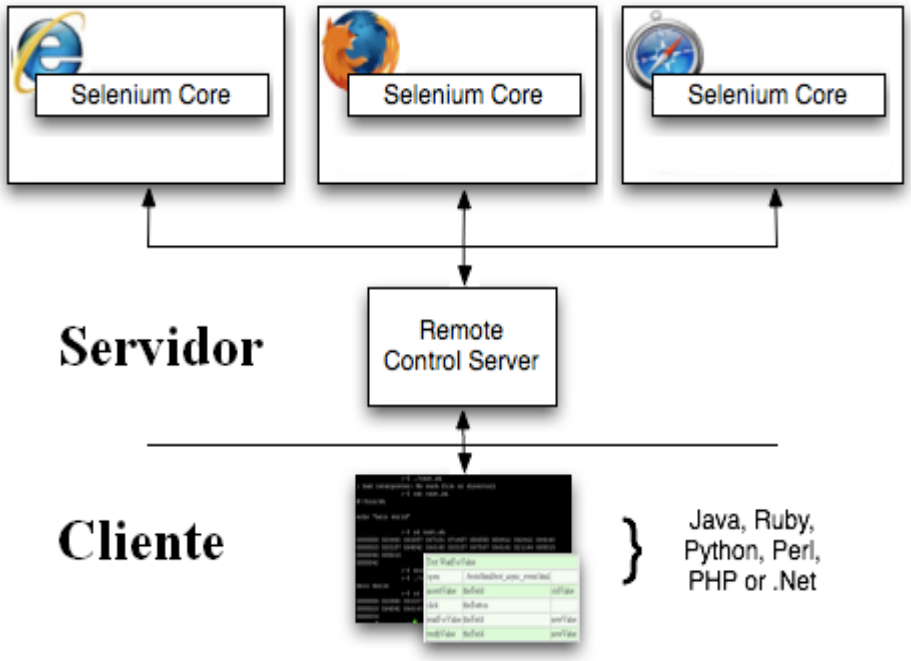


Figura 8: Arquitectura Selennium

Las pruebas de sistemas se lanzarán en lo servidores de aplicación como pruebas desarrolladas en JUnit. De esta manera podemos ejecutar las pruebas registradas por el equipo de desarrollo, y tener al mismo tiempo los resultados en un formato igual al de las pruebas unitarias.

La clase de Junit que implementa las pruebas de sistemas, recibe 4 parámetros:

- **otc.selenium.host** → Host de Selenium configurado en la herramienta Jenkins
- **otc.selenium.port** → Puerto de Selenium configurado en la herramienta Jenkins
- **otc.selenium.browser** → Navegador con el que se desea ejecutar la prueba configurado en la herramienta Jenkins

- **otc.proyecto.url.base:** Dominio de la aplicación contra el que se ejecuta la prueba
- **otc.proyecto.entorno** → Entorno en el que se ejecuta la prueba (*Desarrollo/Pruebas*).

Estas variables se guardan a nivel de propiedad del sistema y se accederá a las mismas de la siguiente manera:

```
System.getProperty("otc.selenium.host"),
```

Las tres primeras propiedades permitirán modificar el contexto de ejecución del servidor Selenium de una manera dinámica y la cuarta, posibilita diferenciar la lógica dependiente del entorno, por ejemplo para indicar la url de inicio de la aplicación.

Un ejemplo de una prueba de sistema podría ser el siguiente:

```
package tests;

import com.thoughtworks.Selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
public class pruebaTest2 extends SeleneseTestCase {
	@Before
	public void setUp() throws Exception {
		String host = System.getProperty("otc.selenium.host", "localhost");
		int port = Integer.parseInt(System.getProperty("otc.selenium.port",
		    "4444"));
		String browser = System.getProperty("otc.selenium.browser",
		    "*firefox");
		String url = System.getProperty("otc.proyecto.url.base",
		    "http://www.jakina.ejiedes.net/");
		selenium = new DefaultSelenium(host, port, browser,url);
		selenium.start();
	}

	@Test
	public void testPrueba5() throws Exception {
		selenium.open("/r33-2220/es");
		selenium.type("buscadorEjie", "prueba");
		selenium.click("r01gBtnSearch");
		selenium.waitForPageToLoad("30000");
		assertTrue(selenium.isTextPresent("prueba"));
	@After
	public void tearDown() throws Exception {
		selenium.stop();
	}
}
```

En el test del ejemplo se accede a la página de inicio de euskadi.net, se escribe “prueba” en el buscador y se aprieta el botón, después de esperar a que se cargue la página nos aseguramos (*assert*) de que existe un texto en la página que sea “prueba”.

Cuando un assert no se cumple, el test se detiene no ejecutándose ninguna de las secuencias siguientes. Para los casos en los que interese que la prueba continúe, se puede utilizar el comando verify que permite verificar sin parar la ejecución.

Nota: Es recomendable tracear y capturar las excepciones que se puedan lanzar, para tener información en un posible error de la prueba.

4. Casos especiales

4.1 Alerts / pop-ups / Prompts / Advertencias navegador

El servidor de Selenium intenta salvar estos diálogos para evitar que se pare la ejecución de la prueba. No obstante, si estos diálogos se ejecutan antes de que la página se haya cargado, Selenium no será capaz de tratarlo y se parará la ejecución. Será necesaria en este caso una intervención manual del usuario.

Por su parte cada navegador genera advertencias de seguridad, que tampoco es capaz de capturar por si solo.

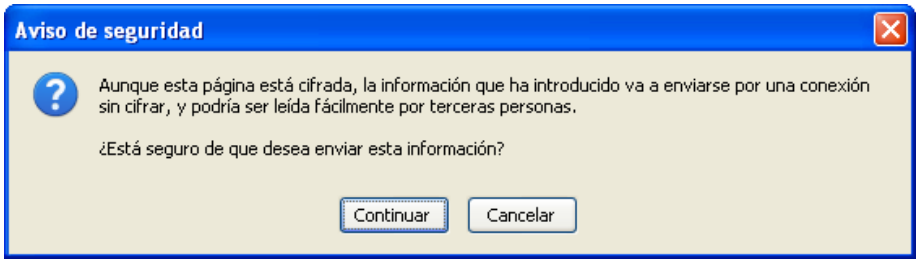


Figura 9: Aviso de seguridad

En el siguiente ejemplo se hace uso de una funcionalidad que ofrece la herramienta, que lanza un evento a nivel de hardware de la presión de una tecla del teclado, simulando de esta manera, la intervención del usuario.

```
for (int second = 0; second<30; second++) {
     try {
            if (!"Consultoría de Áreas de Conocimiento".equals(selenium.getTitle())){
                  selenium.keyPressNative(""+java.awt.event.KeyEvent.VK_ENTER);
            }else{
                  break;
            }
     } catch (Exception e) { }
                  Thread.sleep(1000);
     }
 }
```

El código genera un evento del teclado cada segundo hasta que el title de la página es el de la página de inicio de nuestra aplicación, de esta manera se salvan todos los mensajes del navegador que salgan por la interfaz de usuario y que no puedan ser capturados por el servidor de Selenium.

Para que esta solución funcione, ***el foco deberá estar en el mensaje*** que se quiere aceptar.

4.2 Aplicaciones RIA

Selenium soporta aplicaciones con contenido javascript, sin embargo al implementar los test hay que tener en cuenta las peticiones ajax al servidor.

Ajax nos permite realizar llamadas al servidor de manera asíncrona sin obligatoriamente recargarse la página, al desarrollar el test se debe contar con el tiempo para que se complete la transacción.

Un ejemplo para comprobar que la petición ya se ha ejecutado podría ser el siguiente:

```
boolean peticionAjax=false;

for (int second = 0; second<30; second++) {
        try {
                if (selenium.isElementPresent("link=ajaxLink")) {
                        peticionAjax=true;
                        break;
                }
            } catch (Exception e) { }
        Thread.sleep(1000);
}
```

En el ejemplo ilustrado, se espera hasta un tiempo máximo de 30 segundos comprobando que se ha cargado el elemento *"ajaxLink".*

En Selenium-IDE existen los comandos que cubren esta misma función como son por ejemplo *waitForTextPresent()*, *waitForBodyText()* y en general *waitForXXXXX()*. Se utilizarán las distintas soluciones dependiendo de las necesidades de la lógica.

4.3 Conexiones seguras

La herramienta Selenium soporta también la navegación por páginas seguras bajo el protocolo https, sin embargo, ciertos navegadores por motivos de seguridad muestran una advertencia que requiere confirmación por parte del usuario, esto provoca que la ejecución se detenga sin poder finalizar el test sin una intervención manual.

Para remediar esto, existen diferentes soluciones dependientes del navegador, por ejemplo:

FireFox: Se soluciona con perfiles firefox. Se crea un nuevo perfil (firefox –P) y se añade una excepción con el certificado.

IExplorer: Se soluciona confiando en el certificado.

Los clientes web instalados en los servidores de Ejie, ya confían en los certificados que se utilizan normalmente por las aplicaciones albergadas en sus servidores. Si fuera necesario uno distinto, se debe pedir que se incluya.

4.4 XLNets usuario/contraseña

Será habitual que las aplicaciones albergadas en Ejie utilicen como sistema de autenticación y autorización XLNets.

Tal y como es la implementación de este sistema, la sesión se mantiene en un cliente Windows, si no está instalado este cliente o si utilizamos un navegador que no sea Internet Explorer, por ejemplo, firefox, aparecen los siguientes mensajes de advertencia.

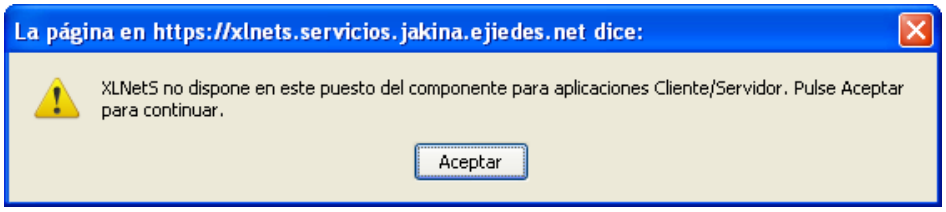


Figura 10: Aviso XLNets

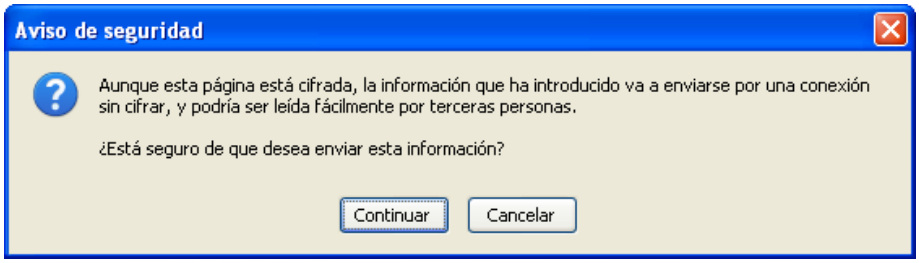


Figura 11: Aviso de seguridad

El primero es una alerta generada por el producto que se ejecuta antes de cargarse la página, Selenium por tanto no puede tratarla, y el segundo es un aviso de seguridad del propio navegador que tampoco se puede tratar.

Una posible solución sería generarnos para las pruebas un mock del sistema XLNets o en su defecto, intentar salvar los mensajes por otros medios.

En el siguiente ejemplo se hace uso de la solución propuesta en el apartado 4.1 de este documento para salvar ambos mensajes.

```
        selenium.setSpeed("1000");
        selenium.open("/s73bTramitacionWar/clienteJSP/s73binicio.do");
        selenium.waitForPageToLoad("30000");
        String cookies = selenium.getCookie();
        if (!cookies.contains("n38UidSesion")){
          System.out.println("No está conectado a XLNets");
          selenium.runScript("cambiaLoginUsu('true');");
          selenium.type("idUsuario", uss);
          selenium.type("idPassword", pass);
           for (int second = 0; second<30; second++) {
                try {
                        if (!"Consultoría de Áreas de Conocimiento".equals(selenium.getTitle())){
                                selenium.keyPressNative(""+java.awt.event.KeyEvent.VK_ENTER);
                        }else{
                                break;
                        }
                } catch (Exception e) { }
                Thread.sleep(1000);
            }
      }
```

4.5 Navegadores soportados

Selenium soporta los siguientes modos de ejecución:

*firefox	*iexplore	*safari
*mock	*firefox3	*piiexplore
*firefoxproxy	*safariproxy	*firefoxchrome
*pifirefox	*googlechrome	*opera
*chrome	*konqueror	*iehta
*iexploreproxy	*firefox2	*custom

"iexplore" y "iehta" son sinónimos de Inernet Explorer y "firefox" y "chrome" son sinónimos de Firefox. Usaremos "googlechrome" para identificar inequívocamente a Google Chrome.

En lo servidores de Ejie están instalados un cliente firefox para el entorno de desarrollo y un grid en el de pruebas con un nodo para Internet Explorer y otro para firefox.

5. Recomendaciones de diseño

En el siguiente apartado se resumen distintas recomendaciones sobre el diseño de los test de prueba de la aplicación.

El la página oficial, se puede encontrar más información sobre las mismas:

http://seleniumhq.org/docs/06_test_design_considerations.html

5.1 Detectar las pruebas automáticas

No todos los casos de prueba pueden ejecutarse de manera automática, es necesario por tanto dentro del análisis y planificación de las pruebas, estudiar qué pruebas deben ejecutarse de manera manual.

Otros casos necesitarán además de una ejecución automática, una verificación posterior que evidencie el correcto funcionamiento de la aplicación. Sería el ejemplo de un caso de prueba que verifique la usabilidad de una página determinada comprobando los colores que se muestran en pantalla. Esta verificación no se puede hacer de manera automática, pero sí la navegación hasta la página en cuestión.

Utilizando la funcionalidad de Selenium que nos permite capturar la pantalla por la que se está navegando, se pueden generar informes que ayuden a verificar estos casos de prueba.

5.2 Page Object Design Pattern

Este patrón busca diseñar los test de pruebas generando una estructura de clases que representen las distintas páginas de la aplicación y las acciones que se pueden realizar sobre las mismas.

Tendríamos por ejemplo la página de inicio siguiente:

```
public class paginaLoginTest{
        private Selenium selenium;
        private static String usernameID="usernamefield";
        private static String passID ="passfield";
        private static String botonSubmit="sign-in";

        public paginaLoginTest(Selenium selenium) {
                this.selenium = selenium;
                if(!selenium.getTitle().equals("Consultoría - Página de login")) {
                        throw new IllegalStateException("No es la página de login: "
                                        +selenium.getLocation());
                }
        }


        public paginaLoginTest loginValidUser(String userName, String password) {
                selenium.type(usernameID, userName);
                selenium.type(passID, password);
                selenium.click(botonSubmit);
                selenium.waitForPageToLoad("waitPeriod");

                return new paginaLoginTest(selenium);
        }
}
```

Siguiendo este patrón, se centralizan en una clase todas las acciones sobre una misma página, facilitando el mantenimiento de los casos de prueba. Sin embargo, requiere un esfuerzo considerable inicial de diseño e implementación de toda la estructura.

5.3 Localizar los objetos

Existen multiples maneras para seleccionar el objeto sobre el que queremos realizar la acción:

- Por su id
- Por su nombre
- Con una sentencia XPath
- …

Se debe elegir qué estrategia utilizar en cada caso, una búsqueda por id es más eficiente en temas de rendimiento, pero puede que en algunos casos no sea conveniente. Por ejemplo, hay objetos que se generar con distintos Ids en instancias de páginas diferentes, en este caso, es recomendable localizar el objeto por una sentencia XPath

6. Pruebas con Web Driver

El proyecto “Selenium” ha deprecado el uso del “Control Remoto”, que era parte de la arquitectura del “Selenium Grid”, aunque es posible tener funcionalidades de “Grid” que ahora están en el “Selenium Server”.

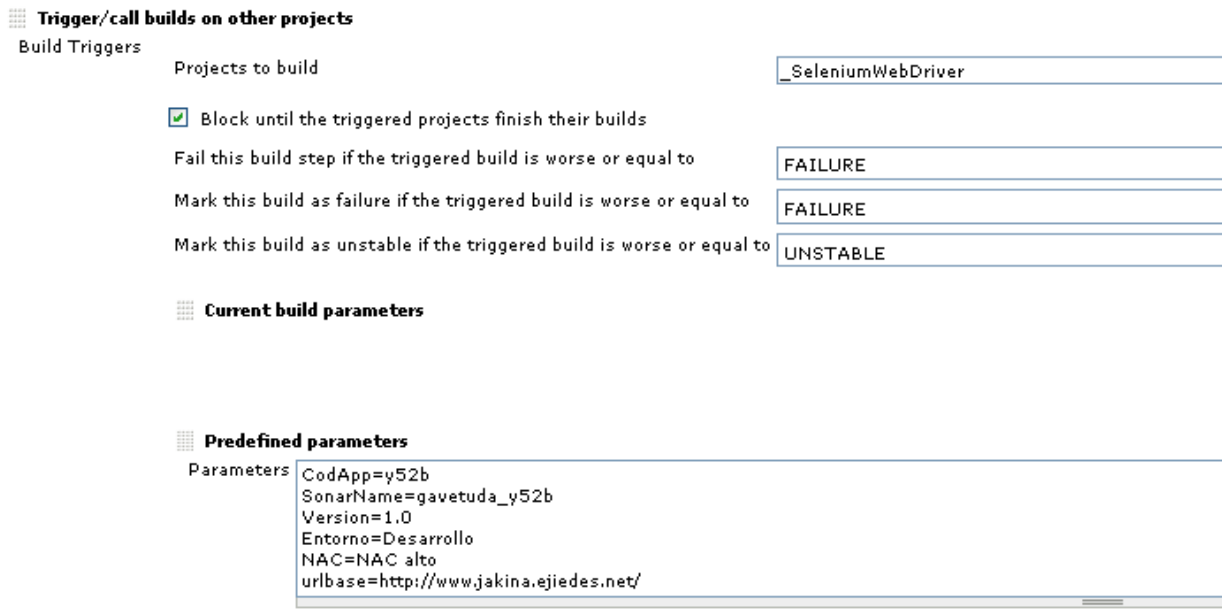
Con todos estos cambios, y teniendo en cuenta que el mantenimiento de un grid es más costoso, nos planetamos ahora usar el Web Driver directamente con Junit, lanzando la prueba en un PC a través de Jenkins.

Para poder usar WebDriver, necesitamos tener en cuenta:

- Las clases de prueba de selenium WebDriver deberán llamarse: **xxx*WebDriver**.class.** Es la manera que tenemos de distinguir las pruebas unitarias de servidor de las que se lanzarán con WebDriver, en un PC; e incluso con clases de utilidades. Todas las clases de prueba se empaquetan en uno de estos jar:
    - test-system.jar
    - test-integartion.jar
    - test-unit.jar

  Sólo el jar de pruebas de sistema es transportado a los PC donde se ejecutan las pruebas de sistema.
- Son pruebas de nivel “Sistema”, según ProbaMet, por lo que las colocaremos dentro de las capretas “test-system”, dentro del módulo más significativo de la prueba (xxxEarClasses es un buen sitio).
- Las librerías de libtest estarán disponibles en el classpath en tiempo de ejecución.
- Se permiten, por el momento, el InternetExplorerDriver y el FirefoxDriver.

Para lanzar la tarea desde Jenkis, se propone añadir a la tarea de tests el proyect “_SeleniumWebDriver” parametrizandolo, teniendo en cuenta que urlbase se pasará como parámetro “otc.proyecto.url.base” al código:



Ejemplo de login en XLNets de desarrollo con iexplorer:

```
import java.util.regex.Pattern;
import java.util.concurrent.TimeUnit;
import org.junit.*;
```

```
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.ie.*;
import org.openqa.selenium.support.ui.Select;

public class w84bWebDriverXlnetsSimpleLoginwithJunit {
	private WebDriver driver;
	private String baseUrl;
	private StringBuffer verificationErrors = new StringBuffer();
	@Before
	public void setUp() throws Exception {
		//driver = new FirefoxDriver();
		driver = new InternetExplorerDriver();
		baseUrl = System.getProperty("otc.proyecto.url.base", "
https://xlnets.servicios.jakina.ejiedes.net/");
		driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
	}

	@Test
	public void testXlnets() throws Exception {
		driver.get(baseUrl + "/xlnets/n38login.htm");
		for (int second = 0;; second++) {
			if (second >= 60) fail("timeout");
			try { if ("Egin klik hemen erabiltzaile-kodea eta pasahitzaren bitartez
identifikatzeko.".equals(driver.findElement(By.linkText("Egin klik hemen erabiltzaile-kodea eta
pasahitzaren bitartez identifikatzeko.")).getText())) break; } catch (Exception e) {}
			Thread.sleep(1000);
		}

		driver.findElement(By.linkText("Egin klik hemen erabiltzaile-kodea eta
pasahitzaren bitartez identifikatzeko.")).click();
		driver.findElement(By.name("idUsuario")).clear();
		driver.findElement(By.name("idUsuario")).sendKeys("jriobell");
		driver.findElement(By.name("idPassword")).clear();
		driver.findElement(By.name("idPassword")).sendKeys("jriobell");
		driver.findElement(By.name("aceptar")).click();
	}

	@After
	public void tearDown() throws Exception {
		driver.quit();
		String verificationErrorString = verificationErrors.toString();
		if (!"".equals(verificationErrorString)) {
			fail(verificationErrorString);
		}
	}

	private boolean isElementPresent(By by) {
		try {
			driver.findElement(by);
			return true;
		} catch (NoSuchElementException e) {
			return false;
		}
	}
}
```

Los resultados de la prueba se mandarán automáticamente a Testlink y al cuadro de mando de calidad.