Trabajo Práctico Especial

Autómatas, Teoría de Lenguajes y Compiladores

Manual de usuario

Alumnos:

Liu, Jonathan Daniel 62533
Vilamowski, Abril 62495
Wischñevsky, David 62494

Profesores:

Arias Roig, Ana María
Golmar, Mario Agustín

23 de Noviembre de 2023

Tabla de contenido

Tellurium

Tellurium es un *Domain Specific Language* basado en *JavaScript* y *Node Js.* Su objetivo es facilitar la creación de *tests* de unidad, y la utilización de la librería de *Selenium*. Por esto, su sintaxis está compuesta por un subconjunto de *JavaScript*, al que se agrega un conjunto de funcionalidades especiales.

Compilación

Para crear una *suite* o un *script* de *Tellurium*, se debe compilar el proyecto y ejecutar el siguiente comando:

```
$ ./start.sh {nombre del archivo} [path]
```

donde *path* es el directorio donde se quiere generar el archivo (si no es indicado, por default utiliza ./output)

Para correr el *script*, debe estar instalada la librería de *selenium-webdriver* con *npm* dentro de la carpeta elegida. Luego, simplemente se debe ejecutar el archivo con *Node Js* (versión probada en desarrollo: v18.15.0).

Ejemplo: (asumiendo que el directorio no existe, y el archivo *program* tiene código válido)

```
$ ./script/build.sh
$ ./script/start.sh program ./output
# Se instala el paquete de selenium-webdriver automáticamente...
$ cd output
$ node program
```

El *script test.sh* es similar, con la diferencia que genera todos *tests* en la carpeta *output*. En caso de querer ejecutar algunos de los *tests* generados, primero debe correrse

```
$ ./script/test.sh
$ cd output
$ npm install selenium-webdriver
$ node {nombre del test}
```

Hay que recordar que muchos de los tests fueron pensados a nivel léxico y sintáctico. Puede ocurrir que generen código válido de *JavaScript* que luego no pueda ser ejecutado por *Node* o que tenga escaso interés como *test*.

Consideraciones sobre JavaScript

Para aquellos programadores que se encuentren habituados a la sintaxis de *JavaScript*, se recomienda que tengan presente que las siguientes características no se encuentran implementadas.

- Strings *single quote* y *template literals*
- Operadores pre y post incrementales
- Clases y constructores
- El *spread operator*
- *Nullish coalescing* y *optional chaining operators*
- Las variantes del *for* y *do while*
- *Throw*
- Funciones *Lambda* sin llaves
- Declaración múltiple de variables usando comas “, ”
- *Scopes* de una sola línea sin llaves (con *if, for, while* por ejemplo)

Para más información, consultar el apartado de **Futuro** del informe.

Funcionalidades propias de Tellurium

Suites

Un archivo de *Tellurium* puede contener código *Tellurium* suelto, o contener únicamente una *suite*, que puede ser anónima o tener nombre.

```
Suite [suite_name]{
      //modules
}
```

Modules

Una suite solo puede contener módulos, pudiendo ser estos anónimos, con nombre, o los módulos especiales *BeforeAll*, *AfterAll*.

Sintaxis:

```
Module [module_name] {
      //code
}

BeforeAll {
      //code
}

AfterAll {
      //code
}
```

Estos pueden estar en cualquier orden, pero sus nombres no se pueden repetir y sólo puede haber hasta un módulo *BeforeAll* y un módulo *AfterAll*.

El módulo *BeforeAll* se ejecutará al inicio de cada módulo, todas las variables declaradas en *BeforeAll* se pueden acceder en cualquier módulo.

El módulo *AfterAll* se ejecutará al final de cada módulo, indistintamente de que si estos hayan pasado exitosamente o no.

Al ejecutarse una *suite*, se ejecutarán todos los módulos y al final, se mostrarán la cantidad total de módulos y la cantidad de los que pasaron exitosamente.

Dentro de los módulos se puede hacer lo mismo que en una función, si se hace *return*, se corta su ejecución.

Ejemplo:

```
Suite actionSuite {

   BeforeAll {
       browserStart("chrome");
       navigate("http://pawserver.it.itba.edu.ar/paw-2023b-12/search");
   }


   AfterAll {
       browserQuit();
   }


   Module basicStringModule {
       var searchinput = $("//input[@id='searchNavInput']");
       await searchinput.waitUntil(Tellurium.enabled, 2);
       await searchinput.sendKeys("test");
       value = await searchinput.getAttribute("value");
       console.log("search input value: ", value);
       assertEquals value, "test";
   }


   Module keyTextnModule {
       var searchinput = $("//input[@id='searchNavInput']");
       await searchinput.waitUntil(Tellurium.enabled, 2);
       await searchinput.sendKeys(<|"te" +SHIFT "S" -SHIFT "t" |>);
       await sleep(1000);
       value = await searchinput.getAttribute("value");
       console.log("search input value: ", value);
       assertEquals value, "teSt";
   }
```

```
   Module keyActionsModule {
       var searchinput = $("//input[@id='searchNavInput']");
       await searchinput.waitUntil(Tellurium.enabled, 1);
       await searchinput.sendKeys(<|"test" +CONTROL LEFT -CONTROL DELETE
"r"|>);
       await sleep(1000);
       value = await searchinput.getAttribute("value");
       console.log("search input value: ", value);
       assertEquals value, "rest";
   }
}
```

Assertions

Tellurium provee funcionalidades de *assertions* para poder probar las predicciones que hacen en los *tests*.

Se proveen 4 sentencias para esto: *assertTrue, assertFalse, assertEquals, assertNotEquals*

Sintaxis:

```
assertTrue {actual};
assertFalse {actual};
assertEquals {expected}, {actual};
assertNotEquals {expected}, {actual};
```

Vale aclarar que internamente *assertTrue* y *assertFalse* comparan con "==" contra sus respectivos *expected*, mientras que *assertEquals* y *assertNotEquals* comparan con "===".

Ejemplos:

```
assertTrue 1 == 1;
assertFalse false;
assertEquals 1, a;
assertNotEquals "test", a.getText();
```

Retry

Con el fin de facilitar el uso de estructuras de control *try-catch*, dentro del contexto de las páginas *web*, donde los elementos pueden funcionar o fallar de manera inconsistente, se provee la estructura de control *retry*.

Sintaxis:

```
try {
      //code
} retry ({exception1} | {exception2} | … [, retry_attempts]) {
      //attempts exhausted
} catch ({exception}) {
      //unhandled exception code
} finally {
      //code
}
```

El *retry* lleva como argumentos:

- Los tipos de excepciones que permite que se haga un reintento, separados por '|'.
- Opcionalmente, la cantidad de veces que se reintenta. Por defecto lo reintenta 1 vez.

El código dentro del alcance del *retry* se ejecuta cuando se acabó la cantidad de intentos y no logró ejecutar el código del *try* exitosamente.

Si la excepción no es de ninguna de las clases manejadas explícitamente por el *retry*, saltará al *catch* y no habrá más reintentos.

Ejemplos:

```
var n = -1;
var arr;
try {
      arr = Array((n+=1));
} retry(RangeError) {
      console.log("failed second attempt");
      assertTrue false;
} catch(e) {
      console.log("failed for unexpected reasons");
      assertTrue false;
}
```

```
var button;
try {
      button = $("//button[@id='submitButton']");
      button.waitUntil(Tellurium.enabled, 5);
      button.click();
} retry(TelluriumExceptions.TimeoutError |
TelluriumExceptions.NoSuchElementException, 5) {
```

```
        console.log("failed 6th attempt");
} catch(e) {
        return;
}
```

Funcionalidades relacionadas con Selenium

Operadores XPath

Para poder hacer búsquedas dentro de la jerarquía *HTML*, *Tellurium* ofrece dos operadores que reciben un *XPath*:

- El operador de búsqueda individual: `$({xpath})`
- El operador de búsqueda múltiple: `#({xpath})`

El primer operador devuelve una promesa de un elemento que "*wrappea*" un objeto *Selenium.WebElement*. Nativamente se proveen los siguientes métodos sobre el mismo:

- `.sendKeys(sequence) : Promise<void>`
  Ejecuta una secuencia de actions sobre el elemento
- `.click() : Promise<void>`
  Clickea el elemento
- `.getText() : Promise<string>`
  Obtiene el texto del elemento
- `.getAttribute() : Promise<string>`
  - Obtiene un atributo del elemento
- `.isDisplayed() : Promise<boolean>`
  Indica si el elemento es visible
- `.isSelected(): Promise<boolean>`
  Indica si el elemento está siendo seleccionado
- `getTagName(): Promise<string>`
  Indica el nombre de la etiqueta del elemento
- `waitUntil(condition, timeout) : Promise<Selenium.WebElementPromise>`
  Espera *timeout* segundos hasta que se produzca un evento particular. Por defecto, *Tellurium* ofrece los objetos *Tellurium.visible*, *Tellurium.enabled*, *Tellurium.disabled*.

Para usuarios más avanzados, el objeto también cuenta con un atributo *"element"*, que contiene el auténtico *Selenium.WebElement*.

El segundo operador devuelve una lista de "*wrappers*", que funcionan de la misma manera.

Ejemplo:

```
var a = $("//a[@id='searchButton']");
await a.click()
var rows = #("//tr[contains(@class, 'note-row')]");
if ((await rows[0].getText()) === "test") {
  console.log("ok");
}
```

Action sequences

Los *sequences*, son una serie de acciones encerradas entre el conjunto de símbolos "<|" y "|>", que pueden ser ejecutadas sobre un elemento, utilizando el método *sendKeys*.

Pueden ser utilizadas como literales y asignadas a variables.

Pueden contener *strings* planos, que se traducen en la escritura secuencial de dichos caracteres; o teclas particulares. Estas últimas pueden, opcionalmente, estar precedidas de un '+' (*keydown*) o un '-' (*keyup*).

Las teclas soportadas por el léxico del lenguaje son:

- CONTROL
- SHIFT
- ALT
- LEFT
- RIGHT
- UP
- DOWN
- ENTER
- BACKSPACE
- DELETE
- TAB
- ESCAPE
- PAGEUP
- PAGEDOWN
- END
- HOME

Por otra parte, también se ofrecen las siguientes operaciones sobre secuencias:

- `.add(sequence) : wrapper`
  Permite concatenar dos secuencias para obtener una nueva
- `.multiply(n) : wrapper`
  Permite concatenar una secuencia consigo misma n veces

Sintaxis:

```
<| [texto] [(+/-) tecla] … |>
```

Ejemplos

```
<| |>
(<| "user" |>).multiply(2)
<| +SHIFT "upper" -SHIFT "lower" |>
(<| "user" TAB "1234" TAB |>).add(<| ENTER |>)
<| +SHIFT END -SHIFT +CONTROL "c" -CONTROL |>
```

Funciones auxiliares

Además de las funcionalidades mencionadas, *Tellurium* ofrece una librería estándar con algunas funciones de *Selenium* muy usadas, que pueden utilizarse libremente dentro de los módulos o *scripts*.

- `browserStart(browser) : void`
  Carga el *browser* (“firefox”, “chrome”, etc)
- `browserQuit() : Promise<void>`
  Cierra el *browser*
- `navigate(url) : Promise<void>`
  Abre el *browser* en la *URL* elegida. Requiere que se haya hecho un *browserStart* previamente
- `sleep(timeout): Promise<void>`
  Hace que la ejecución del programa se detenga {timeout} segundos

También se ofrece un alias al objeto de error de *Selenium* (*selenium-webdriver/lib/error*) mediante la constante *TelluriumExceptions*.