12

December 2010

testing experience

The Magazine for Professional Testers

printed in Germany

print version 8,00 €

free digital version

www.testingexperience.com

ISSN 1866-5705

Open Source Tools

Tellurium Automated Testing Framework

by Haroon Rasheed

The Tellurium Automated Testing Framework for web applications began originally two years ago as a project with a different testing concept from that of the Selenium framework. While most web testing framework software focuses primarily on individual UI elements, Tellurium treats the whole UI element as a widget; calling the element a UI module.

Currently in 0.7.0 release, the core of the project has quickly generated multiple sub-projects as follows.

- **Tellurium UDL**: Tellurium UID Description Language (UDL) Parser.
- **Tellurium Engine**: Based on Selenium Core with UI module, CSS selector, command bundle, and exception hierarchy support.
- **Tellurium Core**: UI module, APIs, DSL, Object to Runtime Locator mapping (OLM), and test support.
- **Tellurium Extensions**: Dojo Javascript widgets and ExtJS Javascript widgets.
- **Tellurium UI Module Plugin (Trump)**: A Firefox plug-in to automatically generate the UI module after users select the UI elements from the web under testing.
- **Tellurium Maven Archetypes**: Maven archetypes to generate skeleton Tellurium JUnit and Tellurium TestNG projects using one Maven command.
- **Tellurium Reference Projects**: Use Tellurium project site as examples to illustrate how to use different features in Tellurium and how to create Tellurium test cases.
- **Tellurium IDE**: A Firefox plug-in that records user actions and generates Tellurium test scripts. This includes UI Module definitions, actions and assertions. The scripts are written in Groovy.
- **TelluriumWorks**: A standalone Java Swing application used to edit and run Tellurium test scripts. An IDE plug-in for IntelliJ IDEA is in development.

Features:

1. **Tellurium UI Module**. Represents the UI being tested. Object uids are used to reference UI elements that are expressive.
2. **UI attributes**. Used to describe the UI instead of fixed locators. The actual locators are generated at runtime. If the attributes are changed, new runtime locators are generated by the Tellurium Framework. Tellurium then self-adapts to UI changes, as necessary.
3. The Santa algorithm. Improves test robustness by locating the whole UI module in a single attempt. A UI module partial match mechanism is used to adapt to attribute changes up to a certain level.
4. The Tellurium UI templates and the Tellurium UID Description Language (UDL). Used to represent dynamic web content.
5. **Groovy Class separation. Example**: UI and corresponding methods are defined in a separate Groovy class which decouples the test code from the Tellurium UI Module.
6. **Tellurium Framework**. Enforces the separation of the UI Module from the test code allowing easy refactoring.
7. **Additional Tellurium Framework Features**:
   - Uses abstract UI objects to encapsulate web UI elements
   - Supports widgets for re-usability
   - Offers a DSL for UI definition, actions, and testing
   - Supports Group locating to locate a collection of UI components in one attempt
   - Includes CSS selector support to improve test speed in IE
   - Improves test speed by Locator caching and command bundling
   - Supports Data-driven test support

Why UI Module?

The UI Module is a collection of user interface elements grouped together. Usually, the UI Module represents the UI object in a format of nested basic UI elements. For example, the Google search UI Module can be expressed as follows.

```
ui.Container(uid: „GoogleSearchModule“, clocator:
[tag: „td“]){
       InputBox(uid: „Input“, clocator: [title:
„Google Search“])
       SubmitButton(uid: „Search“, clocator:
[name: „btnG“, value: „Google Search“])
       SubmitButton(uid: „ImFeelingLucky“, cloca-
tor: [value: „I‘m Feeling Lucky“])
}
```

Tellurium is built on the foundation of the UI module. Tellurium uses UI Module definitions to build Locators for the UI elements at runtime. This makes Tellurium more robust and responsive to changes of internal UI elements. It also makes Tellurium expressive. UI elements can be referred to simply by appending the names along with the path to the specific element. This enables Tellurium's Group Locating feature, making composite objects reusable and addressing dynamic web pages.

The test cases are written in Java, Groovy or plain domain specific language scripts. The UI definition allows the tests to be more expressive and easy to understand within the context of the tests. For example, the tests for the Google Search UI Module defined above would be as follows.

```
type "GoogleSearchModule.Input", "Tellurium test"
click "GoogleSearchModule.Search"
```

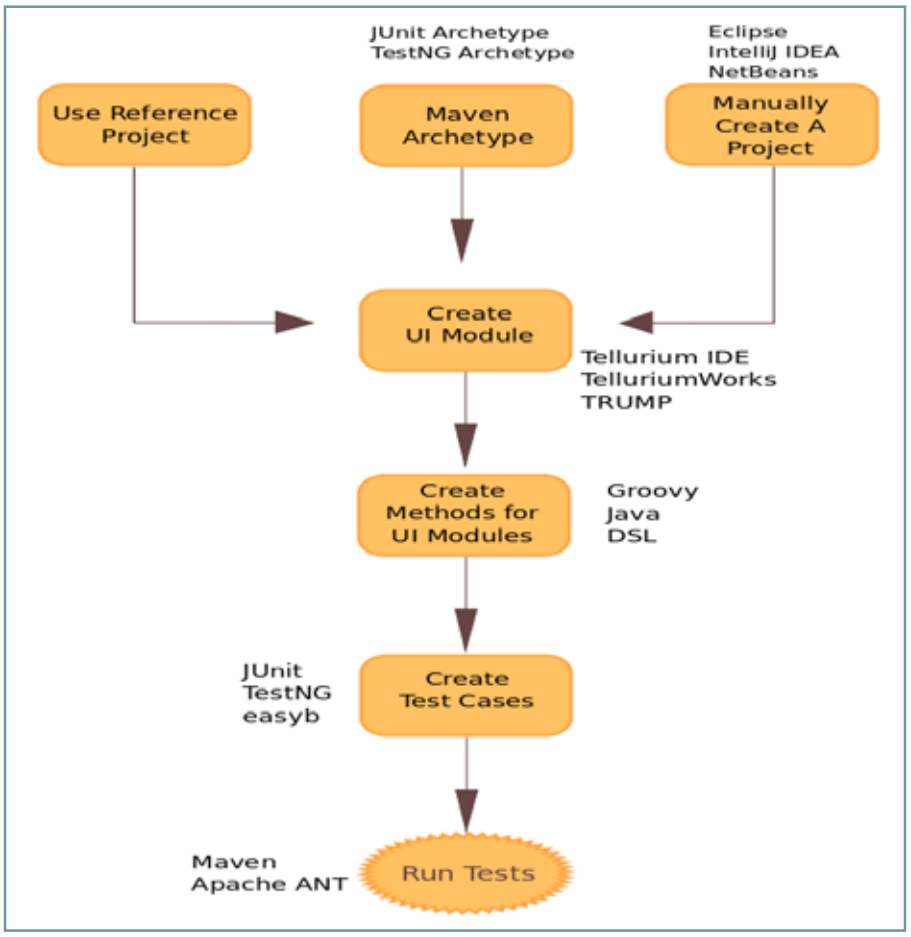
Tellurium sets the object to locator mapping automatically at runtime so that UI objects can be defined simply by their attributes using composite locators.

UI Module definition also decouples the tests from the UI locators, thus making it easier to refactor the tests.

Getting Started With Tellurium

There are several ways to get started with Tellurium. The following diagram illustrates all the possible ways to set up a functional project with Tellurium.

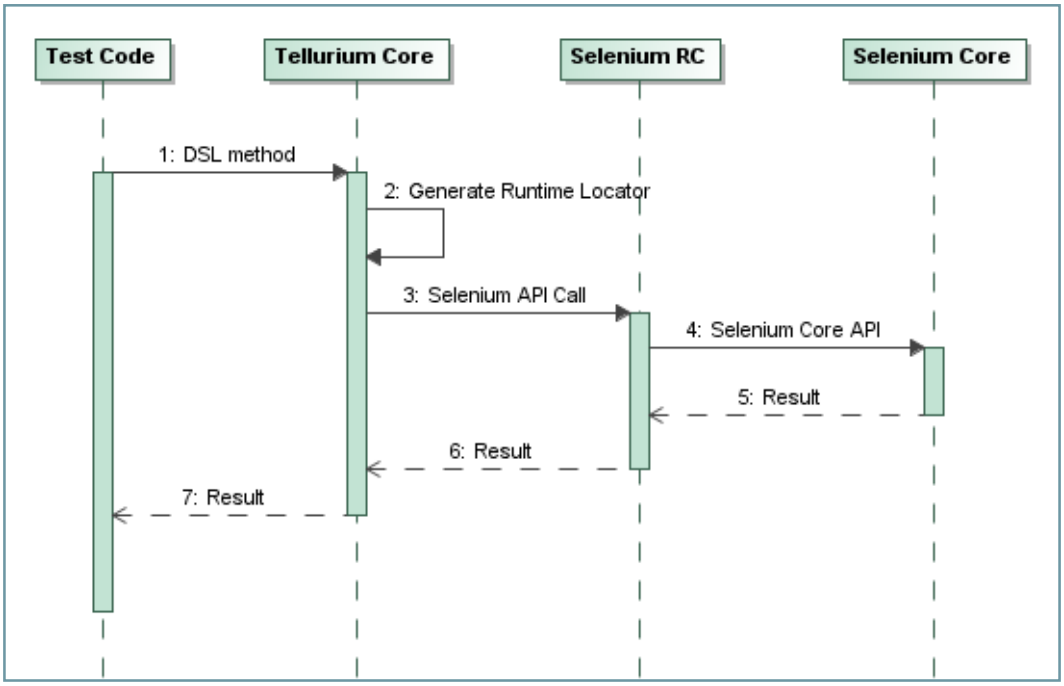
This also serves as the workflow to develop a typical Tellurium test case. A user would start with setting up a Tellurium project, create a UI module using any of the tools provided by Tellurium to identify the UI for the application under test, create re-usable methods for different operations on the UI module and then using these methods in the test case to develop the actual test case.



How Tellurium Works?

Tellurium works in two modes. The first one is to work as a wrapper of the Selenium framework. Tellurium core generates the runtime locator based on the attributes in a UI module and then pass the Selenium calls to Selenium core with Tellurium extensions.

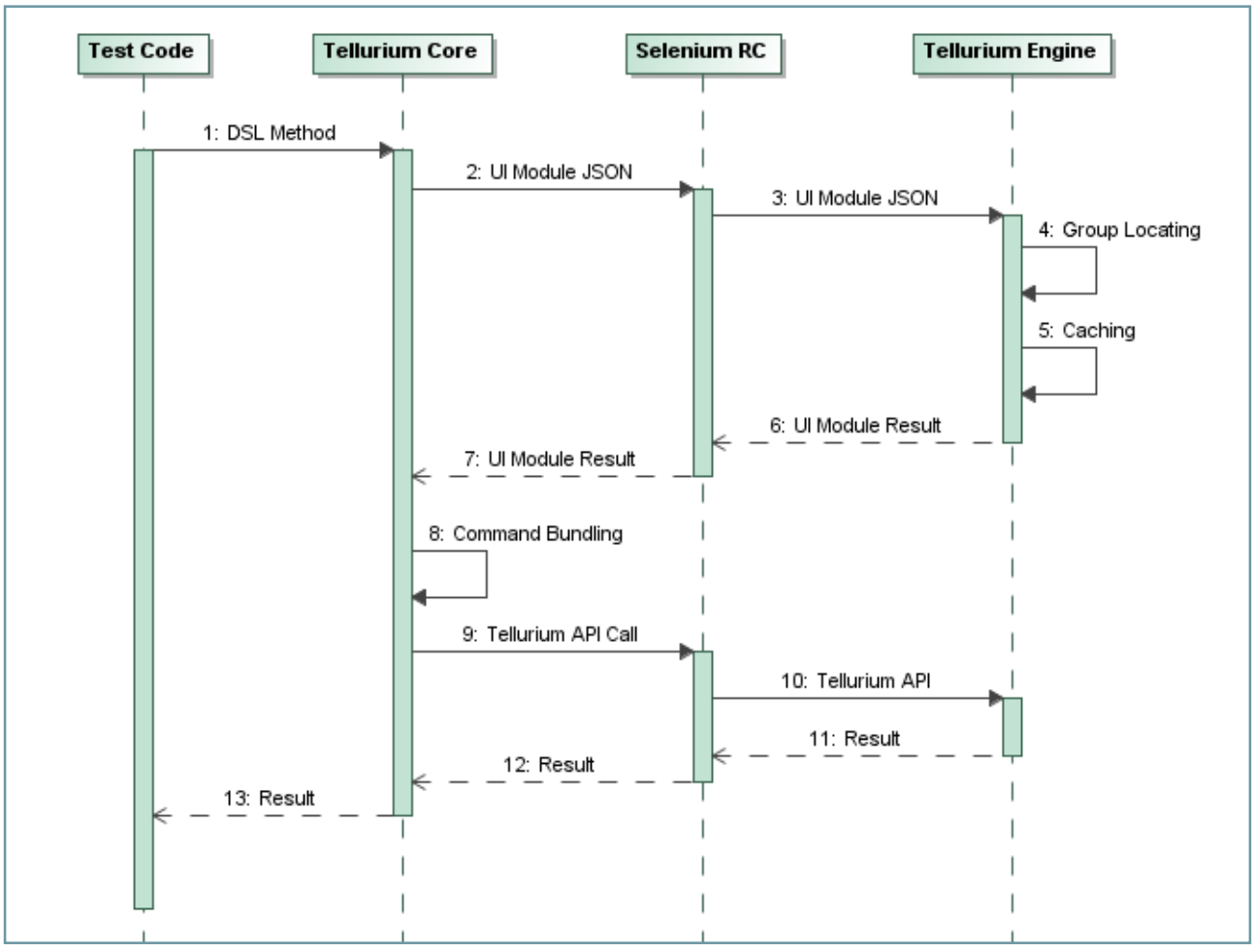
The following diagram illustrates the flow of a Tellurium test when it is run as a wrapper around a Selenium framework.



Tellurium is evolving to its own test driving Engine and the work mode is different as compared to the mode when it is run as a wrapper around Selenium framework.

The Tellurium Core will convert the UI module into a JSON representation and pass it to the Tellurium Engine for the first time when the UI module is used. The Tellurium Engine uses the Santa algorithm to locate the whole UI module and put it into a cache. For the sequent calls, the cached UI module will be used instead of locating them again. Another new feature in Tellurium 0.7.0 is the Macro Command, which combines multiple commands into one batch and then sends them to Tellurium engine in one call to reduce the round trip latency.

This mode is different as shown in the following sequence diagram.



Testing Support

For test support, the following Tellurium Frameworks are used to run Tellurium test cases:

- JUnit . By extending TelluriumJUnitTestCase
- TestNG . By extending TelluriumTestNGTestCase
- easyb . By extending TelluriumEasybTestCase
- TelluriumMockJUnitTestCase and TelluriumMockTestNGTestCase. Incorporated to mock the HTTP server. An HTML web page can then be loaded and tested against it with minimal configuration.

Internationalization Support

Tellurium provides support for internationalization of String and Exception messages. Any software system should have support for regional language settings and options to be effective.

Internationalization and localization provides the support, with locals defining the language and the country. Every locale will have a language code followed by the country. For example ft_FR represents the French language in the country of France.

Internationalized Strings for each locale is provided through a MessageBundle engineered for specific locales and the format is:

```
<MessageBundleName>_<language-code>_<country_code>.properties
```

Future Plans

Tellurium is a new and innovative framework with many novel ideas derived from both the development team and the user community. There are many areas Tellurium is considering for future development, including:

1. Tellurium 0.7.0 has implemented a new test driving engine using **jQuery**. Tellurium will continue to develop the new Engine to its maturity.
2. The **Tellurium IDE** release candidate is out to record and generate test scripts. They are key to the success of Tellurium and they will continue to be improved upon.
3. Tellurium as a cloud **testing tool** is another very important Tellurium future development.

Biography

I have worked as Test Automation Engineer for more than 8 years. During this time, I have worked on different projects ranging from web based CRM applications to Solaris based legacy products for the Oil & Gas industry.
I am currently working as an independent test consultant helping clients to set up maintainable and effective test automation frameworks.