Transform tools

Tool suite for model inference and test generation

FITTEST Project

11/2013

Introduction

Transformtools is a suite of tools integrated in a standalone Eclipse plug-in that :

- infers models from FITTEST LOGS, FBK Trace Format, Selenium HTML test cases.
- generates CTE trees and abstract tests
- infers domain input classifications
- transforms abstract tests to SELENIUM test cases and ROBOTIUM test cases

This document describes:

- FSM model format
- Domain Input Specification Format
- User guide of the eclipse plugin

For more complete user guide, please refer the user guide of the ITE framework.

FSM Model Spec

FSM models can be specified either in DOT format using Graphviz graphical tool or our format. If you use DOT, use our utility tool dot2fsm to convert to our format. This tool is available in the Downloads page.

A FSM model includes Nodes (States) and Transitions (Events). Our format for FSM models has the following rules:

1. Each model is one .fsm file
2. Each line specify a Transition, ending with a ;
3. The first Node is the starting node, there can be many ending Nodes.

The format of each line is as follows:

S -> [Event] E;

In which:
* S is the source node of the transition.

- E is the sink node of the transition.
- Event is the name of the event that makes the system under consideration change its state, from S to E

Examples:

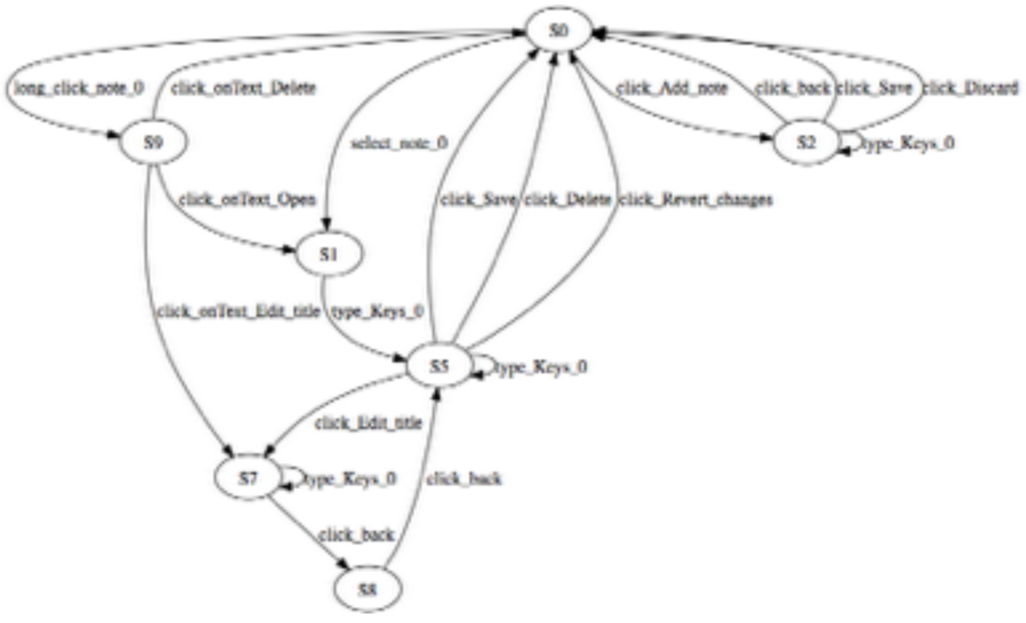
1. In DOT format:

```
digraph fsm {
        S0 -> S1 [label="select_note_0"];
        S0 -> S9 [label="long_click_note_0"];
        S9 -> S1 [label="click_onText_Open"];
        S9 -> S0 [label="click_onText_Delete"];
        S9 -> S7 [label="click_onText_Edit_title"];
        S0 -> S2 [label="click_Add_note"];
        S2 -> S0 [label="click_back"];
        S2 -> S2 [label="type_Keys_0"];
        S2 -> S0 [label="click_Save"];
        S2 -> S0 [label="click_Discard"];
        S1 -> S5 [label="type_Keys_0"];
        S5 -> S5 [label="type_Keys_0"];
        S5 -> S0 [label="click_Save"];
        S5 -> S0 [label="click_Delete"];
        S5 -> S0 [label="click_Revert_changes"];
        S5 -> S7 [label="click_Edit_title"];
        S7 -> S8 [label="click_back"];
        S8 -> S5 [label="click_back"];
        S7 -> S7 [label="type_Keys_0"];
        S7 -> S5 [label="click_OK"];
        }
```

2. In FSM format:

```
S0 -> [select_note_0] S1;
S0 -> [long_click_note_0] S9;
S9 -> [click_onText_Open] S1;
S9 -> [click_onText_Delete] S0;
S9 -> [click_onText_Edit_title] S7;
S0 -> [click_Add_note] S2;
S2 -> [click_back] S0;
S2 -> [type_Keys_0] S2;
S2 -> [click_Save] S0;
S2 -> [click_Discard] S0;
S1 -> [type_Keys_0] S5;
S5 -> [type_Keys_0] S5;
S5 -> [click_Save] S0;
S5 -> [click_Delete] S0;
S5 -> [click_Revert_changes] S0;
S5 -> [click_Edit_title] S7;
S7 -> [click_back] S8;
S8 -> [click_back] S5;
S7 -> [type_Keys_0] S7;
S7 -> [click_OK] S5;
```

Visually, the model is represented as follows:

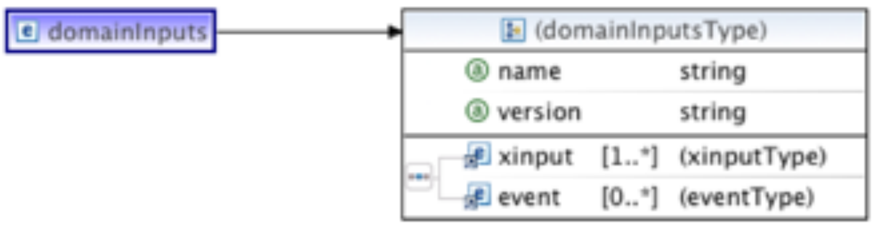


Domain Input Specification

Domain input partitioning and combinatorial testing have proven to be effective in detecting faults and dealing with domain input explosion. Such techniques require domain input values to be classified/grouped into representative classes; any value taken from a class has the same effect on the SUT. However, to the best of our knowledge, no unified format for the specification of input classification exists. Consequently, software engineers have to spend much time in redefining input classifications for different usages, maintaining consistency, and the like, which are costly.

This technical report aims at bridging the gap, introducing an XML structure for the unified specification of domain input classifications. The proposed format is very similar to XML Schema Definition (XSD), which is familiar with most of contemporary software developers.

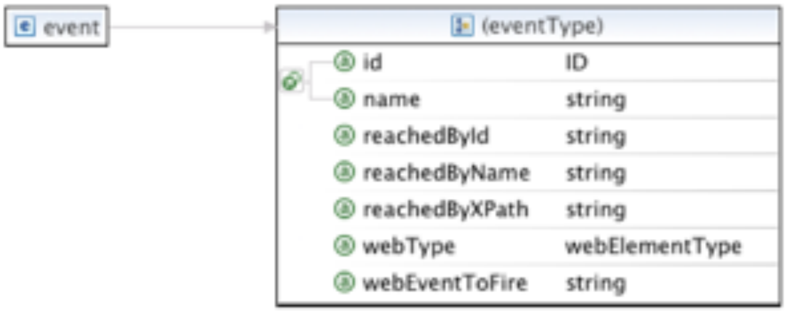
Domain input specification of a whole SUT or of some of its functionalities is stored together in an XML file. The root node of the file is DOMAININPUTS, it contains two lists: XINPUT and EVENT.



```
<xs:element name="domainInputs">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="tns:xinput" minOccurs="1" maxOccurs="unbounded" />
                        <xs:element ref="tns:event" minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
                <xs:attribute name="name" type="xs:string" />
                <xs:attribute name="version" type="xs:string" />
        </xs:complexType>
</xs:element>
```

EVENT

An event element specifies concrete information about an interaction between a component (e.g, a visual interface control) of the SUT and the outside world (a user, a testing tool).

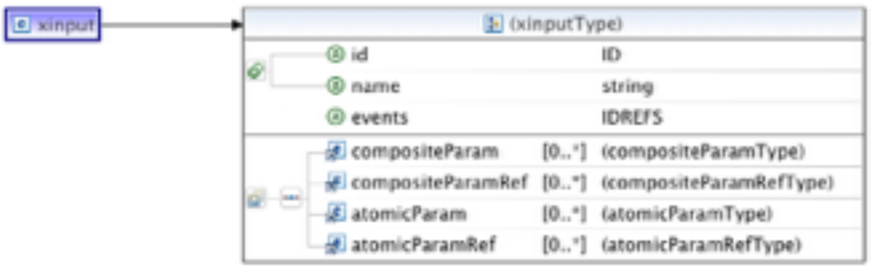


```
<xs:element name="event">
        <xs:complexType>
                <xs:attributeGroup ref="tns:commonAttGroup" />
                <xs:attribute name="reachedById" type="xs:string" />
                <xs:attribute name="reachedByName" type="xs:string" />
                <xs:attribute name="reachedByXPath" type="xs:string" />
                <xs:attribute name="webType" type="tns:webElementType" use="required" />
                <xs:attribute name="webEventToFire" type="xs:string" use="required"/>
        </xs:complexType>
</xs:element>
```

Element	Description
Id	Unique id of the element in the specification file
Name	Name of the element
ReachedByID	Id of corresponding GUI element (control), e.g. button id
ReachedByName	Name of the corresponding GUI element
ReachedByXPath	XPath specification of the GUI element that shows precisely how to reach the element
WebType	The actual type of the element, e.g. button, textbox, checkbox
WebEventToFire	The actual event to fire, can be a GUI event,e.g. click(), or a javascript invocation.

XINPUT

An xinput element specifies a set of parameters and their classifications for a set of events that have the same parameters.

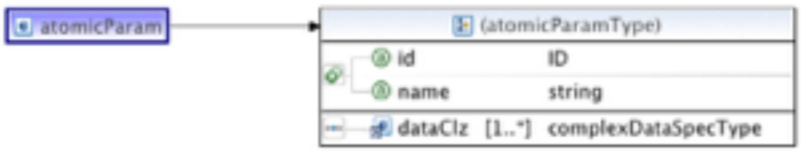

xinput
(xinputType)
id ID
name string
events IDREFS
compositeParam [0..*] (compositeParamType)
compositeParamRef [0..*] (compositeParamRefType)
atomicParam [0..*] (atomicParamType)
atomicParamRef [0..*] (atomicParamRefType)

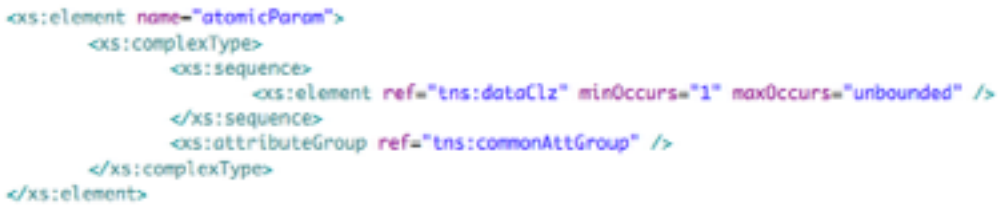
```
<xs:element name="xinput">
        <xs:complexType>
                <xs:group ref="tns:inputSet" />
                <xs:attributeGroup ref="tns:commonAttGroup"></xs:attributeGroup>
                <xs:attribute name="events" type="xs:IDREFS" use="required" />
        </xs:complexType>
</xs:element>
```

The event set is specified by the ids of the events that are available in the specification file.

One xinput element can contain atomic parameters, references of the atomic parameters that have been specified in other xinput elements, composite parameters, and references to composite parameters. Atomic parameters are used to specify primitive data types, e.g. integer, double, string. A composite parameter can contain other composite parameters as well as atomic parameters.

ATOMIC PARAMETER





```
<xs:element name="atomicParam">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="tns:dataClz" minOccurs="1" maxOccurs="unbounded" />
                </xs:sequence>
                <xs:attributeGroup ref="tns:commonAttGroup" />
        </xs:complexType>
</xs:element>
```

An atomic parameter element is the place for the classification specification, it contains a set of data classes (dataClz). Each dataClz element has a base attribute that tells about the data type (any of the supported XSD primitive data type). Then the specification of the dataClz is detailed by its boundary (min, max), its regular expression (pattern), its length, or its limited set of permissive values (enumeration).

EXAMPLE OF A DOMAIN INPUT SPECIFICATION

```
<?xml version="1.0" encoding="UTF-8"?>
<domainInputs name="shoppingCart" version="0.1" xmlns="http://www.fbk.eu/xinput"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.fbk.eu/xinput ../xsd/xinput.xsd ">
    <xinput id="remove_input" events="rem">
        <atomicParamRef paramRef="itemid" />
</xinput>
    <xinput id="empty_input" events="empty"/>
    <xinput id="add_input" events="add">
        <atomicParam id="itemid" name="itemid">
            <dataClz base="string" name="itemid">
                <enumeration value="hat001"/>
                <enumeration value="dog001"/>
                <enumeration value="sou001"/>
                <enumeration value="cha001"/>
                <enumeration value="str001"/>
                <enumeration value="qua001"/>
            </dataClz>
        </atomicParam>
</xinput>
    <xinput id="add_input_cplx" events="noevent">
        <compositeParam id="bookdata">
            <atomicParam id="title">
                <dataClz base="string" name="fulltitle">
                    <minLength value="3"/>
                    <maxLength value="1024"/>
                </dataClz>
            </atomicParam>
            <atomicParam id="price">
                <dataClz base="double" name="expensive">
                    <minInclusive value="10.0"/>
                </dataClz>
                <dataClz base="double" name="cheap">
                    <minInclusive value="0.0"/>
                    <maxExclusive value="10.0"/>
                </dataClz>
            </atomicParam>
            <atomicParam id="date">
<dataClz base="string" name="date">
<pattern value="^(0[1-9]|1[012])[- /.](0[1-9]|[12][0-
9]|3[01])[- /.](19|20)\d\d$"/>
                </dataClz>
            </atomicParam>
        </compositeParam>
</xinput>
    <event id="add" name="addToCart" webType="button" webEventToFire="onclick" reachedBy="/
/button[@id='btnAdd']"/>
    <event id="rem" name="removeFromCart" webType="button" webEventToFire="onclick" reachedBy="/
/button[@id='btnRemove']"/>
    <event id="empty" name="emptyCart" webType="button" webEventToFire="onclick" reachedBy="/
/button[@id='btnEmpty']"/>
</domainInputs>
```

Classification tree

The classification trees generated by the tool suite use the B&M CTE XL professional format. More information is available here: http://www.berner-mattner.com/en/berner-mattner-home/products/cte-xl/

Tool User Guide

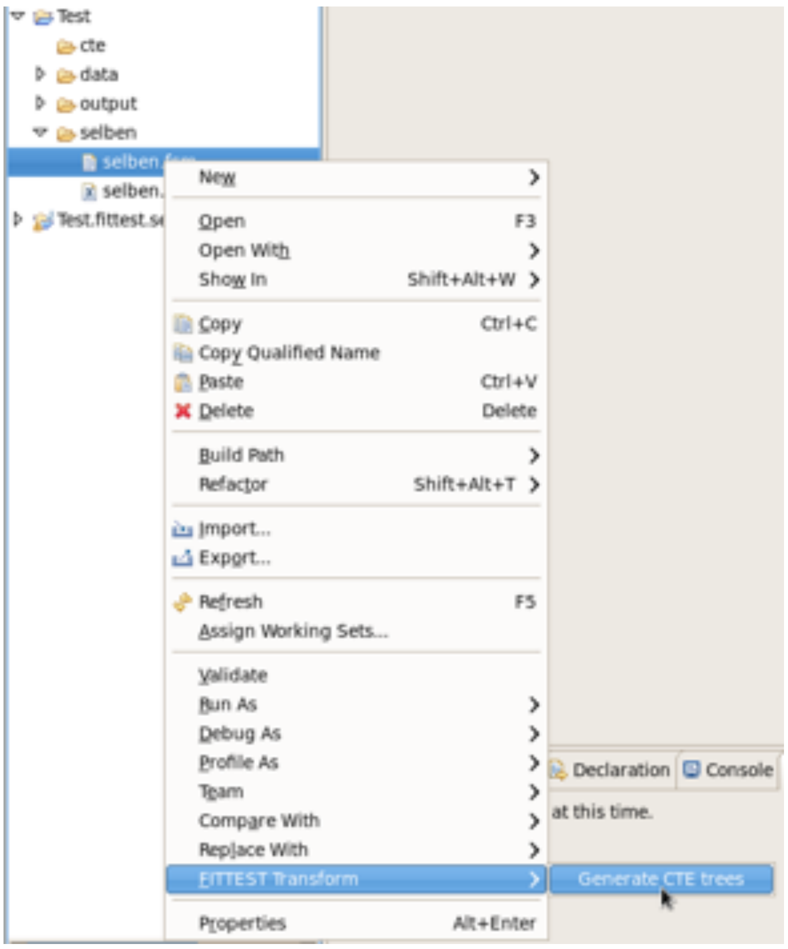
The installation of the plugin is similar to any other eclipse plugin, just drop the package inside the Eclipse/plugins folder and restart Eclipse.

A typical use of the plugin consists of two steps:

1. Generate CTE trees from a model and an input domain specification
2. Transform the generated to executable test cases (Selenium Webdriver, or Robotium for Android)

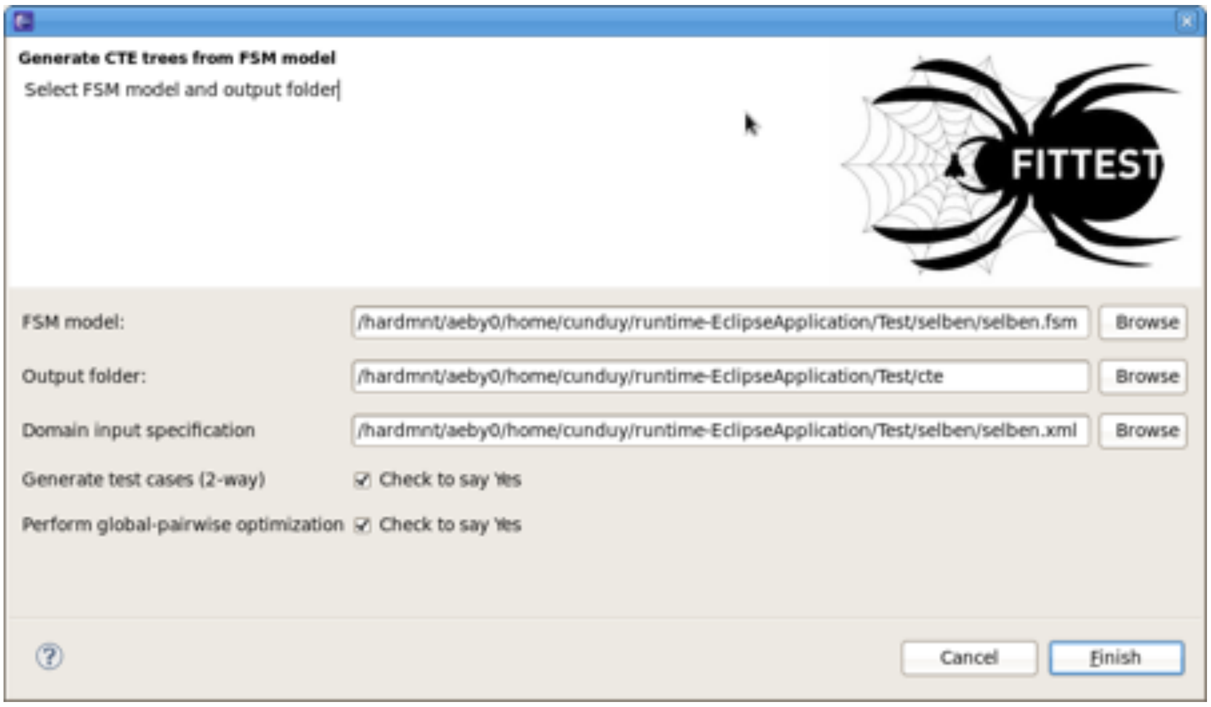
Once CTE trees have been generated, the user can use CTE XL Professional to enhance the trees. For example, constraints can be added to the trees, and CTE XL can help checking for validity of the test case.

Beside, the plugin can infer FSM models from execution traces, more info can be found here.



Generate CTE trees

In Eclipse, select a FSM file, right click and choose FITTEST Transform > Generate CTE trees



Generate CTE trees from FSM model

Select FSM model and output folder

FSM model:	/hardmnt/aeby0/home/cunduy/runtime-EclipseApplication/Test/selben/selben.fsm	Browse
Output folder:	/hardmnt/aeby0/home/cunduy/runtime-EclipseApplication/Test/cte	Browse
Domain input specification	/hardmnt/aeby0/home/cunduy/runtime-EclipseApplication/Test/selben/selben.xml	Browse
Generate test cases (2-way)	☑ Check to say Yes	
Perform global-pairwise optimization	☑ Check to say Yes	

Cancel Finish

There are required inputs:

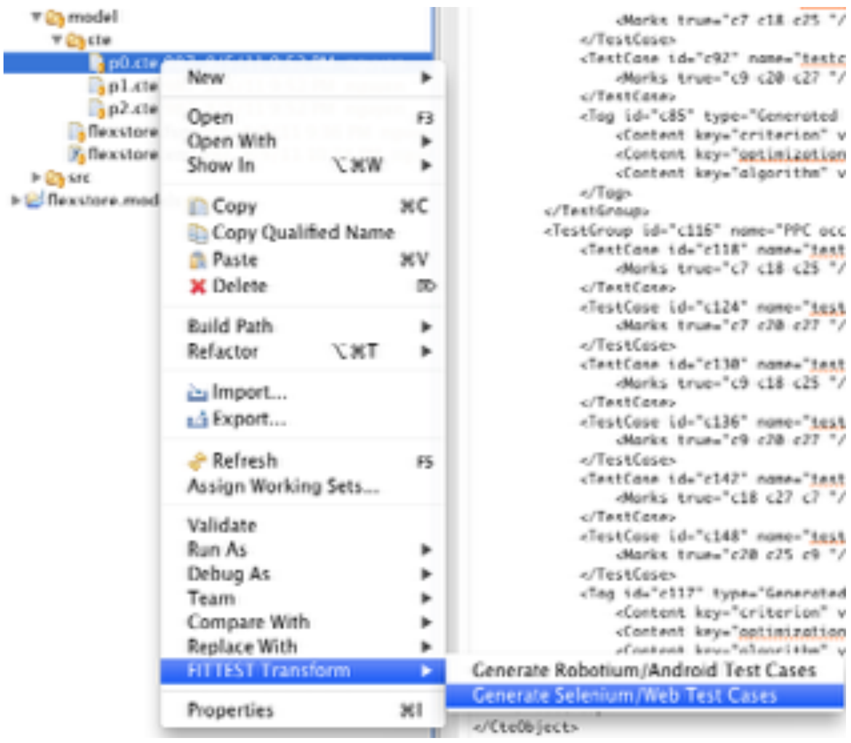
1. A folder to store CTE trees
2. A domain input specification file
3. Check the check boxes to generate pairwise test cases and optimize the generated trees (reduce the number of test cases)

CTE Trees enhancement

Once cte trees are generated, you can use CTE XL Professional to generate more test cases (abstract, not executable) using its generation tools. Moreover, you can also specify path constraints and ask the program to check for validity of the generated test cases.

For example, one can specify a constraint like: if at event A, parameter PA has a value in class C1 (e.g. MONTH), then at the subsequence event B, parameter PB must have a value in class C2 (e.g. 1 <= PB <= 12)

Generate executable test cases (Selenium, Robotium)

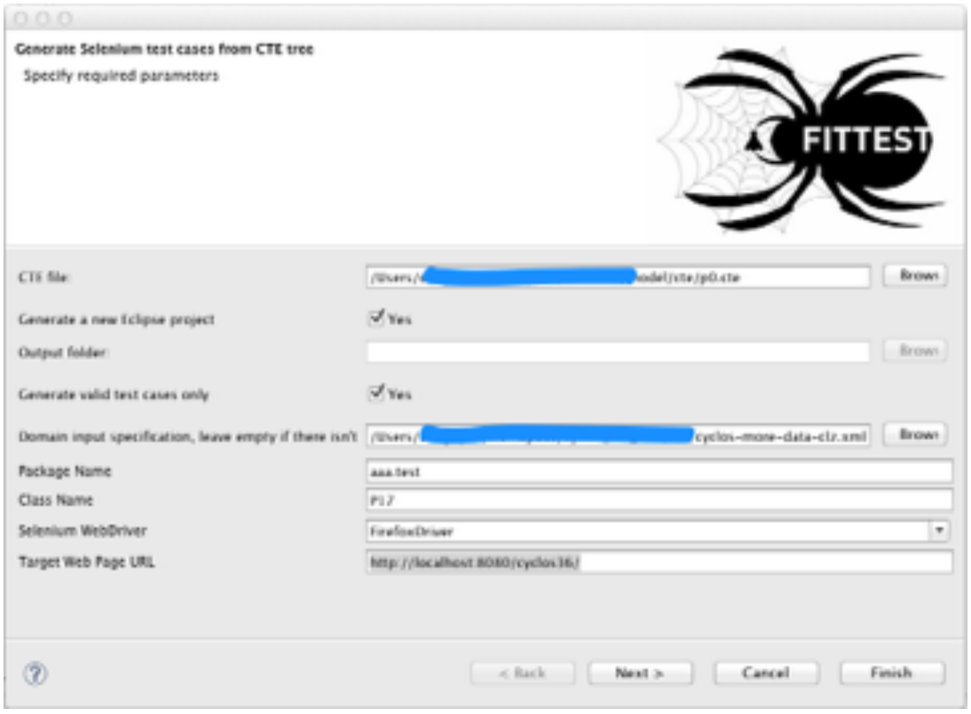


```
        <Marks true="c7 c18 c25 "/
      </TestCase>
      <TestCase id="c92" name="testc
        <Marks true="c9 c20 c27 "/
      </TestCase>
      <Tag id="c85" type="Generated
        <Content key="criterion" v
        <Content key="optimization
        <Content key="algorithm" v
      </Tag>
    </TestGroup>
    <TestGroup id="c116" name="PPC occ
      <TestCase id="c118" name="test
        <Marks true="c7 c18 c25 "/
      </TestCase>
      <TestCase id="c124" name="test
        <Marks true="c7 c20 c27 "/
      </TestCase>
      <TestCase id="c130" name="test
        <Marks true="c9 c18 c25 "/
      </TestCase>
      <TestCase id="c136" name="test
        <Marks true="c9 c20 c27 "/
      </TestCase>
      <TestCase id="c142" name="test
        <Marks true="c18 c27 c7 "/
      </TestCase>
      <TestCase id="c148" name="test
        <Marks true="c20 c25 c9 "/
      </TestCase>
      <Tag id="c117" type="Generated
        <Content key="criterion" v
        <Content key="optimization
        <Content key="algorithm" v
  </CteObject>
```

After having cte trees with abstract test cases (with or without validity checked, see previous step), the plugin can concretize them to executable test cases in JUnit, using Selenium or Robotium.

To generate executable test cases, select a cte tree, right click, and select "FITTEST Transform > Generate Robotium/Android Test Cases", or select "FITTEST Transform > Generate Selenium/Web Test Cases".

Follow the wizard dialogs to specify required information.



Generate Selenium test cases from CTE tree

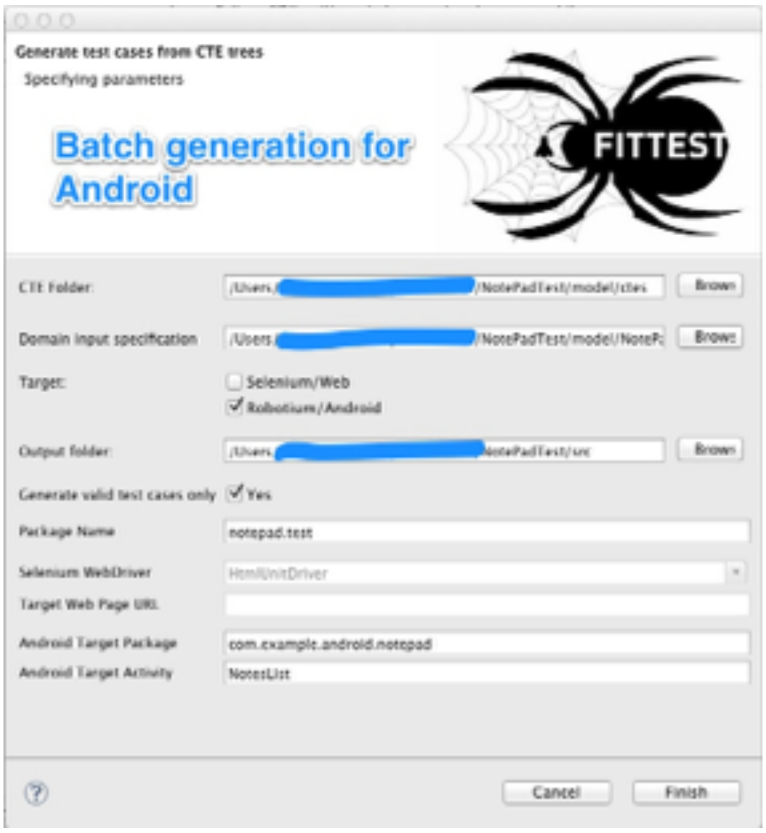
Specify required parameters

CTE file:	/Users/[redacted]odel/cte/p0.cte	Brows
Generate a new Eclipse project	✓ Yes	
Output folder:		Brows
Generate valid test cases only	✓ Yes	
Domain input specification, leave empty if there isn't	/Users/[redacted]cyclos-more-data-clz.xml	Brows
Package Name	aaa.test	
Class Name	P17	
Selenium WebDriver	FirefoxDriver	
Target Web Page URL	http://localhost:8080/cyclos36/	

If you generate test cases for Selenium/Web, you can choose generate a complete Eclipse project for testing purpose that have everything ready, i.e. library.

Batch generation

In case you want to generate executable test cases from multiple cte trees, you can use the batch generation feature of the plugin. In Eclipse, select a folder that contain cte trees, then choose "FITTEST Transform > Batch Generation of Test Cases"



Generate test cases from CTE trees
Specifying parameters

CTE Folder:	/Users/[redacted]/NotePadTest/model/ctes
Domain input specification	/Users/[redacted]/NotePadTest/model/NoteP
Target:	☐ Selenium/Web ✓ Robotium/Android
Output folder:	/Users/[redacted]NotePadTest/src
Generate valid test cases only	✓ Yes
Package Name	notepad.test
Selenium WebDriver	HtmlUnitDriver
Target Web Page URL	
Android Target Package	com.example.android.notepad
Android Target Activity	NotesList

Then, specify the required inputs.