Testing Java code con EFSM

Corso di testing e verifica del software

Materiale

- Capitolo 5 del libro «practical model-based testing»
  - Pdf del capitolo su restricted
- Tool ModelJunit.jar

EFSM

- Per modellare e quindi testare programmi (java) useremo le Extended Finite State Machines
- An EFSM looks similar to an FSM (states and transitions), but it is more expressive because it has internal variables that can store more detailed state information.
- Avremo un insieme di variabili
- Il numero degli stati visibili è comunque finite
  - Alcune volte raggruppiamo tanti valori di variabili in uno stato

EFSM in ModelJUnit

- The basic philosophy of ModelJUnit is to take advantage of the expressive power of Java (procedures, parameters, inheritance, annotations, etc.) to make it easier to write EFSM models, and then provide a collection of common traversal algorithms for generating tests from those models. It is typically used for online testing, which means that the tests are executed while they are being generated.
- The EFSM usually plays a dual role:
  - it defines the possible states and transitions that can be tested,
  - it acts as the adaptor that connects the model to the SUT (which is usually another Java class).

Write an EFSM with ModelJUnit

- Each EFSM model is written as a Java class, which must implement the interface **FsmModel**
- and have at least the following **public** methods:
- **Object getState():**
  This method returns the current visible state of the EFSM. So this method defines an abstraction function that maps the internal state of the EFSM to the visible states of the EFSM graph. Typically, the result is a string, but it is possible to return any type of object.

```
@Override
public Object getState() {
      String result =  car1.toString() + "-" + car2.toString();
      if ((car1.row == 1) && (car1.col == 2)) {
            result += "EXIT";
      }…
```

reset

- **public void reset(boolean):**
  This method resets the EFSM to its initial state.

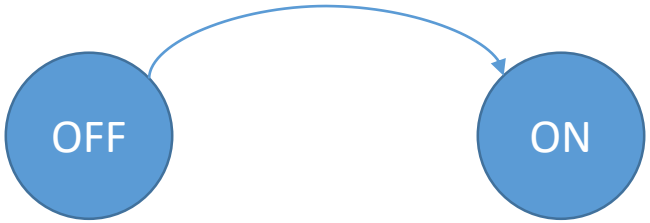
EFSM actions

- **@Action void *namei*():**
  The EFSM must define several of these action methods, each marked with an @Action annotation. These action methods define the transitions of the EFSM. They can change the current state of the EFSM.
- **boolean *namei*Guard():**
  Each action method can optionally have a guard, which is a boolean method with the same name as the action method but with "Guard" added to the end of the name. When the guard returns true, then the action is enabled (so may be called), and when the guard returns false, the action is disabled (so will not be called). Any action method that does not have a corresponding guard method is considered to have an implicit guard that is always true.

Esempio

```
@Action
public void up1() {
      car1.row--;
}
public boolean up1Guard() {
      return (car1.row > 0)
      && (!(car2.col == car1.col && car2.row == car1.row - 1));
}
```

Esempio – interruttore ON - OFF



Versione 1: con una sola azione change

- Scrittura della EFSM in Java

Generazione dei casi di test – valutazione copertura e stampa dei test

- In questo caso generiamo i casi di test utilizzando il seguente codice Java:

```
public static void main(String args[]) {
    OnOffMachine model = new OnOffMachine();
    Tester tester = new RandomTester(model);
    tester.generate(10);
}
```

Generazione dei casi di test

- In questo caso generiamo i casi di test utilizzando il seguente codice Java:

```
public static void main(String args[]) {
        OnOffMachine model = new OnOffMachine();
        Tester tester = new RandomTester(model);
        GraphListener graph = tester.buildGraph();
        CoverageMetric stateCoverage = new StateCoverage();
        tester.addCoverageMetric(stateCoverage);
        CoverageMetric transitionCoverage = new TransitionCoverage();
        tester.addCoverageMetric(transitionCoverage);
        tester.addListener(new VerboseListener());
        tester.generate(10);
        System.out.println("State coverage =" + stateCoverage.toString());
        System.out.println("Transition
coverage="+transitionCoverage.toString());
}
```

Drawing the EFSM

- Dal modello è possibile ottenere il disegno:

1. Esportare in jar
2. Caricarlo in ModelJUnit

Demo + (esercitazione) + video

ATTENZIONE: prima di farlo disegnare assicurarsi che sia corretto (come visto prima) almeno sintatticamente.

Testare del codice senza modificarlo

- Alcune volte NON posso modificare il codice del System Under Testing (SUT)
  - Il codice fa parte di una libreria
- Alcune volte voglio testarere solo una parte del comportamento
  - La classe SUT fa anche altre cose
- Alcune volte non riesco a trovare le azioni semplici del SUT
  - I metodi sono complicati

- In questi casi devo connettere la mia SUT con la EFSM

Connecting with the sut for online testing

- Se voglio usare ModelJunit per online testing della mia SUT devo connetterla:
- Creo uno istanza della sut nel costruttore
- Invio al SUT gli input chiamando I metodi e poi controllo che il comportamento sia corretto.

- Ad esempio:

```
public class RushHourFSM implements FsmModel {
	private int c1_row, c1_col, c2_row, c2_col;
	// system under test
	RushHour sut;
	public RushHourFSM() {
		c1_row = 1;c1_col = 1;c2_row = 1; c2_col = 2;
		// creo sut
		sut = new RushHour();
	}
```

- ..}

Connecting with the sut for online testing

- **@Action void *namei*():**
  The EFSM must define several of these action methods, each marked with an @Action annotation. These action methods define the transitions of the EFSM. They can change the current state of the EFSM. and when online testing is being used, they also send test inputs to the SUT and check the correctness of its responses.
- **reset**: When online testing is being used, it should also reset the SUT or create a new instance of the SUT class. The boolean parameter can be ignored for most unit testing applications.

Connecting with the sut for online testing

- Each action method typically defines a short, straight-line sequence of Junit code that tests one aspect of the SUT by calling one or more SUT methods and checking the correctness of their results.
- The effect of applying model based testing to the EFSM is to make a traversal through the EFSM graph, and this weaves those short sequences of test code into longer sequences of more sophisticated tests that dynamically explore many aspects of the SUT
- Esempio

```
@Action
public void down2() {
      sut.moveCar(car2.row, car2.col, 3);
      car2.row++;
      assertTrue(sut.griglia[car2.row][car2.col] == 2);
}
```