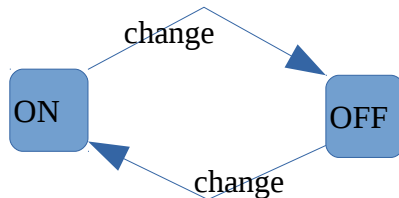


Testing FSM con ModelJunit

Si vuole testare una applicazione java cui FSM è la seguente.



Proviamo a farlo con modeljunit

1. predisporre il progetto eclipse

Per prima cosa apriamo un nuovo progetto eclipse che dobbiamo predisporre per modeljunit.

Scarica modeljunit.jar da:

<http://sourceforge.net/projects/modeljunit/>

Metti il jar nel build path del progetto eclipse.

[Il progetto d'esempio lo trovi sotto mbtxfsm]

2. scrivere il modello della FSM

Il modello FSM si scrive direttamente in Java.

Normalmente questa classe si collega alla SUT (e non è la SUT stessa) (cioè ha in se un riferimento al sistema under test).

La classe del modello FSM deve estendere l'interfaccia `FsmModel`

Con i seguenti metodi:

`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.

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

`@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.

`boolean nameiGuard()`: 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.

NOTA: il controllo di correttezza del comportamento del SUT viene fatto nei metodi `@Action` e/o `getState`. In `Action` dovrei controllare l'output, in `getState` che il sistema vada nello stato opportuno.

Nel nostro esempio la classe è:

```
public class OnOff implements FsmModel{
    enum State {ON, OFF};
    State state;
    public OnOff(){
        state = State.OFF;
    }
    @Action
    public void change(){
        System.out.println("cambio stato");
        if (state == State.OFF) state = State.ON;
        else state = State.OFF;
    }
    @Override
    public Object getState() {
        return state;
    }
    @Override
    public void reset(boolean arg0) {
        state=State.OFF;
    }
}
```

La classe andrebbe collegata alla SUT.

3. Ci sono due modi per testare la FSM.

A. ONLINE testing

Scrivi un caso di test Junit che testa la tua classe. Del tipo:

```
// create our model and a test generation algorithm
//Tester tester = new RandomTester(new OnOff());
Tester tester = new GreedyTester(new OnOff());

// build the complete FSM graph for our model, just to ensure
// that we get accurate model coverage metrics.
tester.buildGraph();

// set up our favourite coverage metric
CoverageMetric trCoverage = new TransitionCoverage();
tester.addCoverageMetric(trCoverage);

// ask to print the generated tests
tester.addListener(new VerboseListener());

// generate a small test suite of 20 steps (covers 4/5 transitions)
tester.generate(20);

tester.getModel().printMessage(trCoverage.getName() + " was " +
trCoverage.toString());
```

Puoi cambiare:

1. il tester:
 1. RandomTester: Test a system by making random walks through an EFSM model of the system.
 2. GreedyTester: Test a system by making greedy walks through an EFSM model of the system. A greedy random walk gives preference to transitions that have never been taken before. Once all transitions out of a state have been taken, it behaves the same as a random walk.
 3. LookaheadTester: A test generator that looks N-levels ahead in the graph.
2. il misuratore di coverage

Eseguendo questo codice, viene esplorata la FSM e vengono chiamati i metodi (action e getState). All'interno di questi metodi dovrei vedere che il mio sistema si comporti correttamente.

B. OFFLINE testing (con l'interfaccia)

Usando il tool come interfaccia grafica possiamo esplorare il modello e generare le tracce offline. Queste tracce andranno poi copiate in un test Junit e convertite in test concreti a mano.

Come aprire il progetto con il tool modeljunit

1. Scrivi la classe che rappresenta la tua FSM e la tua SUT (come detto prima)
2. produci un ajr in cui metti questa class (usa export di eclipse: per comodità salva lo script così ti basterà rilanciare lo script)
3. esegui il tool modeljunit:
 1. o con doppio click sul jar
 2. oppure in riga di comando `java -jar modeljunit-2.5-jar-with-dependencies.jar`
4. crea un nuovo progetto con la tua FSM. A modeljunit servono due cose:
 1. il jar che contiene il .class della tua FSM
 2. il nome (completo della tua FSM).

Salva quindi il tuo progetto in un file .mju.

Se hai problemi puoi sempre partire da un mju fatto (ad esempio prendilo da un esempio) e modificarlo. Ad esempio, per far funzionare il tutto sotto linux io sono riuscito solo creando un file di progetto `exampleOnOff.mju`

Nel quale ho modificato:

```
<className>it.unibg.tvsw.OnOff3</className>  
<packageLocation>file:esercizio.jar</packageLocation>
```

Esercizi FSM testing

1. Macchinetta per il caffè

Un macchina per il caffè riceve in ingresso dei gettoni e produce quando richiesto il caffè. L'utente può inserire fino a 5 gettoni. Esistono due tipi di caffè: quello normale (1 gettone) e quello super (2 gettoni). Il credito attuale viene visualizzato sul display.

A.

Formalizza (su carta):

- gli stati della macchina
- l'input che accetta la macchina (token e produci caffè e reset?)
- le transizioni di stato

B.

prova a implementare la macchinetta in Java . Con opportuni output e metodi per sapere quanti crediti ci sono e così via.

Implementa la sua specifica FSM in Java

Prova ad eseguire l'online testing con il tool

Prova anche a generare tracce con il tool.