Statechart simulation with Yakindu Statechart Tools (SCT)

Fall 2013
Abel Armas-Cervantes (abel.armas@ut.ee)

In this practice session, you will learn how to model and simulate behaviour specified as Statecharts using the toolkit Yakindu Statechart Tools (SCT). As the running example, we will use a controller for a lamp bulb. The practice session is organised in two parts. In the fist part, you will model the basic behaviour, that is, a lamp bulb controlled with a simple switch. In the second part, you will extend the basic Statechart (and some other simulation elements) to specify a flashing feature.

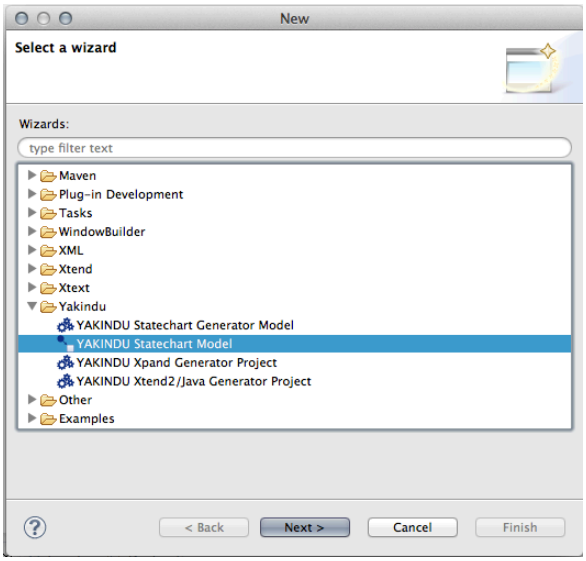
PART 1: Basic behaviour

1. **Download the Yakindu SCT**
   http://statecharts.org/download.html

2. **Unzip the package and open the Eclipse icon**

3. **Create a new java project**
   Inside your project, add a new folder for storing all the files related to your statechart.

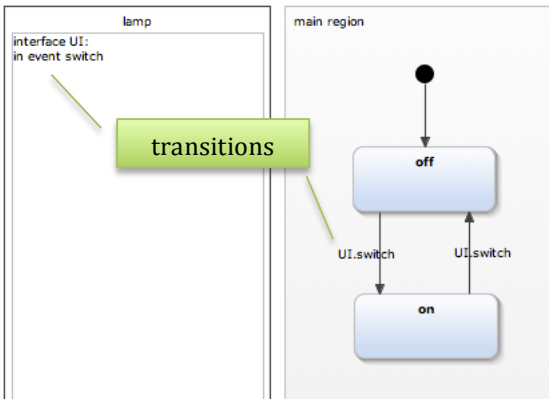
4. **Create a new Yakindu Statechart Model**
   a. Right click on your project > New > Other > Yakindu > Yakindu Statechart Model.
   b. Choose a name for your model (e.g., Lamp.sct) and put it into the folder created in the previous step.



5. Defining the behaviour (Statechart)

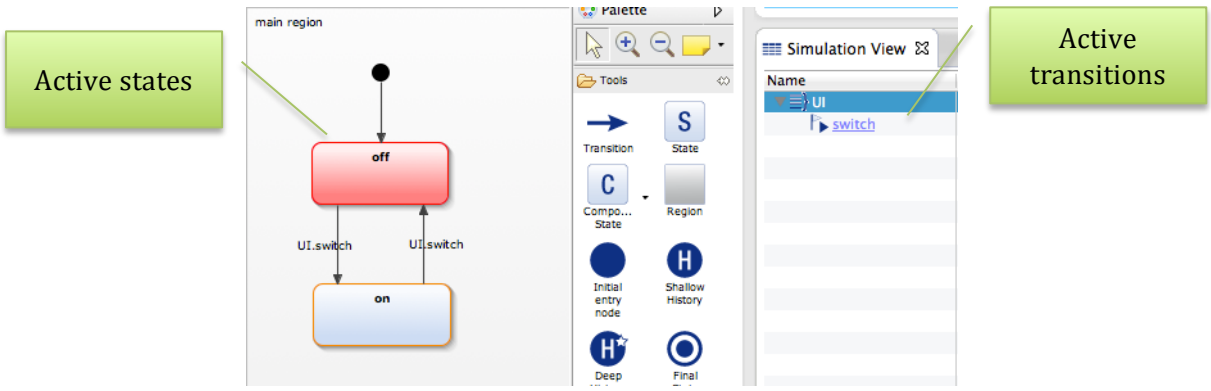
In this practice, we will design a java-based user interface for our statechart. Therefore, events and operations need to be externally visible. Hence, all the declarations of events and operations will be within the scope ***interface***. First of all, we need to specify the behaviour of our basic system using a statechart.

a. Complete your statechart as shown in the following figure:



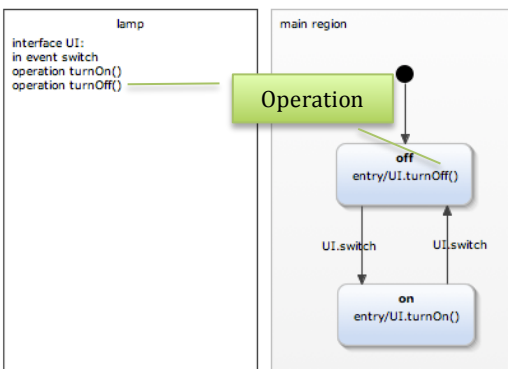
b. You can simulate your state machine by right-clicking on your model > Run as > Yakindu Statechart.

c. Initially, the active state is "`off`", and you can change to another state by selecting an active transition on the right hand side of the model.



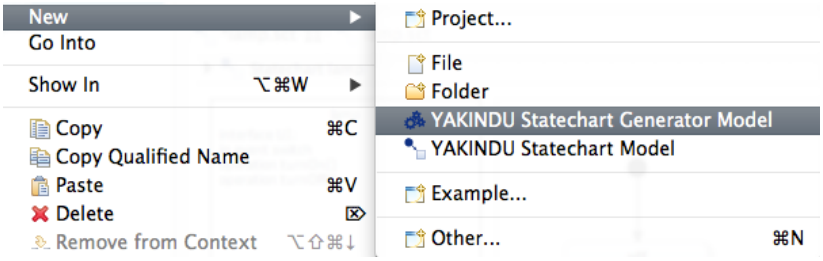
6. Defining operations

Modify your model as follows:

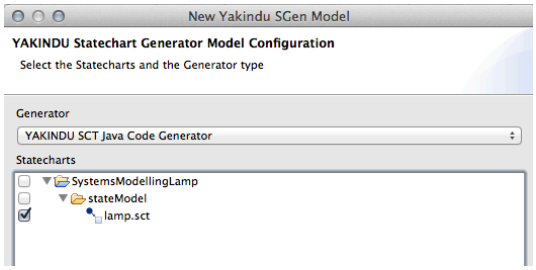


The word "`entry`" specifies the actions that are carried out when entering to a given state.
In what follows, we will start creating the user interface for our statechart. The interface will consist of a button and a JPanel for displaying an image. In short, the button will represent the transition `switch` and it will allow us to traverse over the two existing states. Thus, every time the button is pushed, a different image shall be displayed in the JPanel (such behavior will be specified in the operations turnoff() and turnOn()).

7. Create a new Yakindu Statechart Generator Model



By uniformity, keep the generator model inside the same folder of your statechart. Pick up a name for your Generator Model and select your statechart when required:



8. Modify your .sgen file as follows:

The values in italics vary depending on the name of your project and the name of your statechart, therefore, do not modify the values in `targetProject` and **`statechart.`**

```
GeneratorModel for yakindu::java {
    statechart lamp {
        feature Outlet {
            targetProject = "Lamp"
            targetFolder = "src"
        }
        feature GeneralFeatures {
            RuntimeService = true
            TimerService = true
        }
    }
}
```

9. Generate the Java code for integrating your statechart to the GUI swing-based interface we will construct.

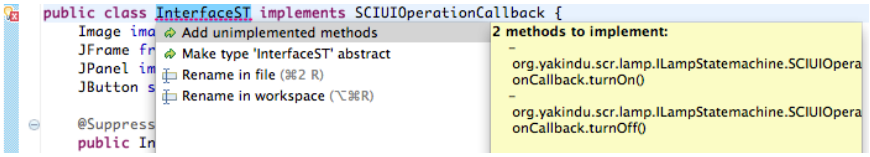
Right click on your .sgen file and select `Generate Statechart Artifacts`. A set of Java classes will be generated and put into the folder specified in the .sgen file (parameter `targetFolder`). Have a look to the following link to read a more detailed explanation about each of the generated Java classes: http://statechart.org/documentation.html#SpecJava

10. Create a new Java class for defining the swing-based GUI interface

The code below defines a simple interface containing a JPanel that will be the container for an image and a button.

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.WindowConstants;

import org.yakindu.scr.lamp.ILampStatemachine.SCIUIOperationCallback;

public class InterfaceST implements SCIUIOperationCallback {
   Image image, on, off;
   JFrame frame;
   JPanel imagePanel;
   JButton switchButton;

   @SuppressWarnings("serial")
    public InterfaceST() {
         on = new ImageIcon("bulb-on.png").getImage();
         off  = new ImageIcon("bulb-off.png").getImage();
         image = off;

         imagePanel = new JPanel() {
            public void paintComponent(Graphics g) { g.drawImage(image, 0, 0, null); }
         };

         Dimension size = new Dimension(image.getWidth(null), image.getHeight(null) + 30);
         imagePanel.setPreferredSize(size);
         imagePanel.setMinimumSize(size);
         imagePanel.setMaximumSize(size);
         imagePanel.setSize(size);

         switchButton = new JButton("Switch");

         JPanel container = new JPanel(new BorderLayout());
         container.add(switchButton, BorderLayout.CENTER);
         container.add(imagePanel, BorderLayout.PAGE_START);

         size = new Dimension(image.getWidth(null), image.getHeight(null) + 90);
         frame = new JFrame();
         frame.add(container);
         frame.setPreferredSize(size);
         frame.setResizable(true);
         frame.pack();
         frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
         frame.setVisible(true);
   }
}
```

An error message must appear asking you to implement a set of methods. Select "`Add unimplemented methods`" as shown in the picture.



The implemented methods correspond to the operations defined in your statechart (i.e., turnOn(), turnoff()). Thus, within those methods you can specify the actions that must be carried out when entering to a given state.
Both images `bulb-on.png` and `bulb-off.png` are in: https://courses.cs.ut.ee/2013/system_modelling/fall/uploads/Main/Figures.zip. You just have to put both images into the root folder of your project.

The code for the methods "`turnoff()`" and "`turnOn()`" is as follows:

```
@Override
public void turnOn() {
        image = on;
        frame.repaint();
}

@Override
public void turnOff() {
        image = off;
        frame.repaint();
}
```

Basically, when entering to the state "`off`" in the statechart, then the image bulb-off.png will be set into the JPanel, and with `repaint` we will refresh the interface. The case for state "`on`" follows analogously.

11. Create a Java class containing a main method:

```
public static void main(String[] args) {
   ILampStatemachine sm = new LampStatemachine(); // Create a instance of your
                                                  // state machine

    InterfaceST ui = new InterfaceST(); // Create the instance of your interface handler
   ui.addEventListener(sm.getSCIUI()); // Add the event listener(s) to the button(s)
   sm.getSCIUI().setSCIUIOperationCallback(ui); // Link the callback methods to the
                                                // operations in the statechart
   sm.init(); // Initialize the internal objects of the statechart
   sm.enter(); // Enter the initial state

   RuntimeService.getInstance().registerStatemachine(sm, 1); // The statechart is passed to the
                                    // runtime service, where 1 corresponds to the delay for
                                    // the statechart internal clock.
}
```

Finally, the method `setEventListener` in the class of your interface looks as follows:

```
public void addEventListener(final SCIUI sciui) {
   switchButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
                sciui.raiseSwitch();
      }
   });
}
```

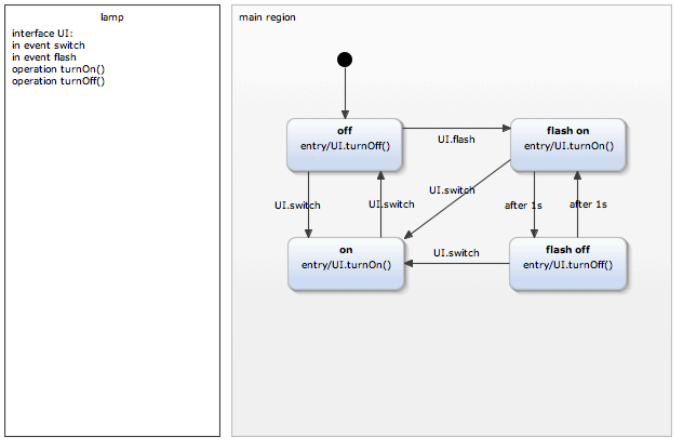
This method specifies that an event over the button "`switchButton`" corresponds to the event "`switch`" in the statechart.

12. Run the Java class containing your main method.

Right click on the class > Run as > Java application.

Part 2: Flashing lamp

Modify your project to specify a flashing behaviour. The intended behaviour is given by the following statechart.



Note that the transitions **flash on** and **flash off** respond to time events (a delay of 1 sec). With all the considerations above, you should be able to complete the statechart and simulate the flashing lamp. Modify the interface as you consider necessary. **Important:** In your main method, before initializing your statechart (i.e., `sm.init()`), it is necessary to add a new time service: `sm.setTimerService(new TimerService());`