(236368) Formal Specifications for Complex Systems

Tirgul on Statecharts in Rhapsody 7.5.3

In this tirgul we'll learn how to use Rhapsody in a minimal way so as to be able to play with statecharts and solve Homework no.3.

This document is meant to take you through a defined path, so please try not to deviate significantly until you finish it. Later, you are encouraged to learn more about the tool by trying its tutorials and playing with it as much as you wish. We hope you will enjoy this tutorial. Have fun!

1 Rhapsody

Rhapsody is installed in the SSDL lab (check soon that you can access and use it).

The license is only available for working from the facutly, the license server is: 19353@ssdl-100 .

Additional information about working with the virtual machines is available at:

`http://ssdl-wiki.cs.technion.ac.il/wiki/index.php/Working_in_SSDL`

The regulations for working in the lab are:

`http://www.cs.technion.ac.il/~ssdl/ssdl/regulations.html`

2 How to build a minimal environment to play with statecharts

- Run the program:
  Start→All Programs→IBM Rational →IBM Rational Rhapsody In Java.
  1) Make sure you never run Rhapsody twice simultaneously - it causes problems. 2) If you get an announcement that the license server is not defined, inform us.f 3) Make sure you always use the Java version (Rhapsody in J) and not the C++ version that is also installed (Rhapsody in C++).

- File→New - enter a name for the project (e.g. XX) and the directory under which it will be saved. If the directory doesn't exist, approve its creation.

- A Browser window appears. Expand "Packages". Click on Default with the right mouse button (RMB from now on). In the menu that pops up, choose Add new→Class.
- Click on class_0 with the RMB. Choose Add New→Diagram→Statechart. A statechart editor opens, and the statechart drawing buttons appear on the left side. Draw the following statechart: (note: all the lines in your drawing will be solid)

```
    ---------      a      ---------
.->| state_0 |---------->| state_1 |
   |         |<----------|         |
    ---------    b        ---------
```

- Go back to the project's browser. Expand Components →DefaultComponent →Configurations. Double-click on DefaultConfig. Under the tab "initialization", expand "default" and check the "class_0" box. This requests an initial instance for this class. Click on the "Derived" option above (so that any other classes required by "class_0" will also be initialized).
- Under the tab "Settings"-Instrumentation, choose "Animation". Press OK.
- Save your project.
- Look at the package Default in the browser: It now has Events (a() and b()), and the class_0 now has Operations a() and b(), meaning it can receive those events. Also notice that the icon of class_0 has a yellow thing that indicates that this class has a statechart.
- We now want to animate the statechart. Only statecharts belonging to *instances* of classes can be animated. Our statechart belongs to the class (as can be seen in its title). Therefore, it can't be animated. Look at the main menu option
  Tools→AnimatedStatechart: it's not selectable. This is because there are no instances of class_0 yet.
  From the main menu, choose Code→"Generate/Make/Run". This can be done faster by pressing the "GMR" icon, the second one to the left

of the selection list where it says "Default Component". If everything works, the code is generated and compiled, and the executable is run. Two things appear:

  - The Animation toolbar, with buttons for controlling the animation.
  - A DOS (black) window where the executable is now running. In this example there is no console I/O, therefore the best thing is to minimize it.

  The "Call Stack" and "Event Queue" should appear as two windows (together with another one) at the bottom. If you don't see them, they may have appeared "collapsed" to the right or the left. Drag their vertical edges to expand them.

- Press now the second button from the left in the Animation toolbar (Go). We would expect to see state_0 with a different color (pink), meaning that it is now the current state, but this doesn't happen. The reason is that our old statechart belongs to class_0, not to one of its instances. It is not an "animated statechart".

- Now the animation is running (we already pressed "Go"). Notice in the browser that class_0 already has an instance (class_0[0]). We now have two ways to open animated statecharts:

  - Clicking the RMB on the instance in the browser and selecting "Open Instance Statechart"
  - With Tools→AnimatedStatechart from the main menu. (it's selectable now)

  Use one of these methods to get the animated statechart. You should now see the initial state marked in pink.

- We can now create events from the Animation toolbar (third button from the right). Try entering an event "a": in the dialog that opens, select the instance (we only have one yet), then select an event (from "a" and "b"). Later, you'll find your previous selections under Events History. Press OK. The statechart now shows that class_0[0] is in state_1.

- Pause the simulation, pressing the "animation break" button "||" in the animation toolbar. Enter a "b" event, as you did before with "a".

Check the event queue window (bottom). You should see the event waiting to be handled. Add another "a" event. Both events should appear in the queue.

- Start stepping the simulation with the "▷|" button (the leftmost). Each step, check out the statechart colors and the three windows at the bottom of the screen. The current state is pink, the previous state and transition are dark green.
- Of course, everything in this example can have a non-default name: from DefaultComponent, through the class_0 to the statechart states. It is easy to rename these by clicking on the names or by opening their respective dialogs. (e.g., the Features dialog for states).
- Stop the simulation (with the "eject"-like button).

3 Experimenting with Another Statechart

We will now get to know the pre-made project "tutorialJava". The project contains a single statechart which presents some of the statechart syntax that you will also use in HW3.

- Download from the course's web site into your home directory the zip file "tutorial-Java.zip". Extract its contents.
- Open the project (from within Rhapsody; look for the icon tutorial.rpy - an orange R icon). If the project browser doesn't appear on screen, use "View→Browser", or the "show/hide browser" icon at the far right of the lower toolbar.
- Open the project's single statechart.
- NOTE: there are some differences between Rhapsody's statecharts and the original language of statecharts defined by David Harel. Syntactic differences:
  - to generate an event internally, it is not enough to write its name after the "/" on a transition. If the event "go" is to be raised within the same object, then the appropriate syntax is "gen(new go())"
  - The syntax of boolean conditions (written between "[ ]") is that of boolean expressions in Java. Therefore, comparisons are done with "==", etc.

  - The syntax of all the actions done is that of Java code. For instance you may print a message to the console using *System.out.println("message");*

  Some of the semantic differences:

  - It is not possible to have several events occurring at the same time in the same statechart: every active object takes a single event at a time from its event queue.
  - The semantics of the "steps" is different, in Rhapsody everything is more sequential. The differences are described in the UML documentation, but are best understood by playing with the tool itself.

- Start the animation (press the GMR button). Minimize the DOS window. Press the "Go" button in the animation toolbar. Use Tools→AnimatedStatechart to open the animated statecharts for the single instance.
- Make the bottom windows somewhat smaller by dragging down their top edge. Leave enough space in them for 3 or 4 lines. Use the Window→TileVertically option to make the statecharts fill the screen. Focusing on each statechart, you can use "Zoom to fit" to see the whole statechart.
- You can now play with the System's statechart in the way we saw before: entering events manually. This is the best method to see what happens before the model "stabilizes". You manually feed events, and single-step watching the windows at the bottom..
- While single-stepping, pay attention to the way the system reacts to the "go" event - the two parallel components take their transitions simultaneously, since they are both triggered by the same event in the same object.
- Note the *isIn*(..) syntax used within the guard. This is equivalent to the in(state) syntax used in class.
- You can also see the effect of history by triggering the "stop" event while the three parts are not in their default states, and then issuing "start" again. Note that as opposed to what explained in class, in Rhapsody each parallel component should be provided with its own history (if we want it to save history of course).

4 Difference between Theory and Rhapsody

- In Rhapsody, each parallel component can have its own history
- In Rhapsody, the events are enqueued. In the statecharts we have seen the external event and any internal event generated are handled at the same time.
- In Rhapsody, $isIn(state)$ is used to check if we are in a certain state
- In Rhapsody, **H** represents history for all the levels (equivalent to **H*** seen in class)
- Java syntax is used in Rhapsody to define actions.

5 Good luck

That is all for today. You can now make more complicated systems, add commands to the environment, etc etc. If you have some trouble with some feature which is needed for the homework, don't hesitate to ask.