Overview of Yakindu

Vesal Vojdani
University of Tartu
Yakindu Docs

- The user guide is available at the following location: https://www.itemis.com/en/yakindu/state-machine/documentation/user-guide/sclang_statechart_language_reference

- We link to it in the course material!

- I will summarize the key syntax!
Definitions

• We can define communication with external components!

• We can also have an internal section for our own definitions.

• Keep in mind that for each (external) interface code is generated
External Interface

• Input events generate code that allows the interface to raise the event: `sciui.raiseSwitch()`

• Output events generate getter methods that allows you to query whether that event was raised. (Observable interface also possible).

• Variables in external interface generate getters and setters to query that variable!

• Operations declared in the interface results in a Callback interface being generated and a method to set an implementation of that callback interface.
External Interface

- Input events generate code that allows the interface to raise the event: `sciui.raiseSwitch()`

- Output events generate getter methods that allow you to query whether that event was raised. (Observable interface also possible).

- Variables in external interface generate getters and setters to query that variable.

- Operations declared in the interface result in a Callback interface being generated and a method to set an implementation of that callback interface.

For the homework: DO NOT MODIFY!
Internal definitions

- You may add a variable! (initialization is optional)
  ```
  var x: integer = 0
  ```

- You can define your own internal events!
  ```
  event e
  ```

- You can define global actions:
  ```
  every 1s / x++
  e / x--
  ```
trigger [ guard ] / effect

- Triggers: events, timer and entry/exit.
- Guards are boolean expressions.
- The effect is either assignment, operation call, or raising an event.
- Combining events:
  ev1, ev2 [c1 || c2] / eff1; eff2
Statechart Features

- You should understand Composition and Orthogonality and use them appropriately in your designs!

- In the docs, you may find a few more state chart features that we did not need: final states, named entry states (with transitions using those names), choice nodes and synchronization.

- You may still use them obviously!

- Let’s just quickly look at synchronization…
Synchronization

Only exit if both are in the waiting state!
How can we achieve this?
Like this, obviously!
Okay, but somewhat more elegantly?
Using synchronization

Yakindu doesn’t care about elegance, but if you use models to communicate ideas, you should care...
Execution Models:
1. Cycle-Based
2. Event-Driven
During the Labs…
We can raise events…
They are triggered left-to-right!
Triggering Events?

• What exactly does it mean to trigger an event?
• Let’s say we are responding to a user event and we raise an event in region 1.
• Does region 2 react to both events?
• Can it take multiple steps?
• What would be the best experiment to get a better understanding of this?
Experiment!

What will be the result of pressing “go”?
We only react to “e”

But what can we conclude?
Internal events take precedence!?
Or maybe…

There’s another reason “e” is preferred?
Different result!

If we reorder transition priorities, we get “only go”
What can we conclude?
private boolean switchEvent;

public void raiseSwitch() {
    switchEvent = true;
}

What can we conclude?
Think of events as just boolean flags (on/off).
Execution Models:
1. Cycle-Based
2. Event-Driven
Event-Driven Execution

• For the homework, we will use event-driven execution.

• This makes unit-testing much easier.

• Event-Driven is more intuitive for normal use-cases!

• It is only complicated if you do crazy things…
  (such as deliberately writing cycle of events)
This example again...

Examples like this just work as expected!
Here’s the key experiment!

What will be the result of pressing “go”?
Let’s hear some guesses and explanations!
This is what happens!

The event for “e” is queued and handled after we have ran to completion with “go”.
IMPORTANT!

• This difference in behavior only applies to internal events. If you try this at home, make sure “e” is internal!

• Even with event-based execution, raising input events is like setting boolean flags. They are not added to the event queue!

• If you repeat these examples, but move “e” to be an input event in the interface, you will get the same behavior as for cycle-based execution.
Smart light switch

- The light has 6 levels of brightness (from 0=switched off up to 5=max brightness). There is only one button to control it!

- If the light was on, then a single push and release on the button, will switch off the light.

- If the light was off, then a single push and release on the button, will switch on the light at the previous brightness level.

- One push and hold on the button makes the level of brightness increase (resp. decrease) if it was decreasing (resp. increasing) previously.

- Once the maximum (resp. minimum) level of brightness is reached the brightness level decrease (resp. increase).
Very smart light switch

- The light has 6 levels of brightness (from 0=switched off up to 5=max brightness). There is only one button to control it!

- If the light was on, then a single push and release on the button, will switch off the light.

- If the light was off, then a single push and release on the button, will switch on the light at the previous brightness level.

- One push and hold on the button makes the level of brightness increase (resp. decrease) if it was decreasing (resp. increasing) previously.

- Once the maximum (resp. minimum) level of brightness is reached the brightness level decrease (resp. increase).

- In case of double push and release, if the light is on, the brightness goes up to the maximum.