State modeling

Luciano García Bañuelos
Institute of Computer Science
About me

Luciano García-Bañuelos

E-mail: luciano.garcia@ut.ee
Office: J.Liivi 2-311

Senior Researcher
Software Engineering Group
University of Tartu

PhD, Grenoble Institute of Technology, France
Grading

- One single project: 10 points
  - To be completed in teams of 4 persons
  - It is not a programming assignment!
    - Although the result is a full-fledged application
    - You must provide a model (Statechart)
    - Such model will be connected to a GUI (provided)
How can we model the state of the shopping cart?

It depends on:

- Object state
- Variable assignment
- Relation status (i.e. number of items)
- Operating methods & processes
- History
State Machine

- A machine whose output behavior is not only a direct consequence of the current input, but of some past history of its inputs
- Characterized by an internal state which represents this past experience

If the phone is ON, then pushing this button will turn it off
If the phone is OFF, then pushing this button will turn it on
State Machine (Automaton) Diagram

Initial state (entry)

Phone is off

State

Transition

Phone is on

on

off
Another example
Actions on the State Machine

- State changes can induce side-effect actions

**Mealy automaton**

**Moore automaton**
Automata can be extended with variables

\[ c := c + 1 \]
Exercise 1

- Consider the state machine below
  - Add the following: Call detected, Answer call, Play announcement, Record message, Caller hangs up, Announcement complete

- Revise the state machine so that the machine answers after five rings.
Since version 2.0, UML adopted the Statechart formalism for state modeling.

A Statechart extends state machines, making them suitable for specifying behavior of complex systems, using:

- Visual formalism
- Rich alphabet model
- State Modularity
- Parallel behavior
A State model consists of multiple state diagrams.

Values of attributes and links in an object instance are grouped together to describe a particular state.

The state is modified in response to the occurrence of events.
An order cannot be placed unless the cart is not empty.
States

- A state is an abstraction of attribute values and links of a particular object
  - An object has a finite number of possible states
  - It can only be in one state at a time
  - At a given moment, various objects can exist each one with its own state

- Solvent
- Minimized
- Maximized
- Insolvent
- Waiting for authorization
- Decision notified
Events

An event is the specification of some occurrence that may potentially trigger effects by an object.

Kinds of events
- Call event
- Signal event
- Change event
- Time event
Call events

- A call event represents the reception of a request to invoke a specific operation.
- A call event is distinct from the call action that caused it.
A signal is an explicit one-way transmission of information from one object to another.
- A signal event is asynchronous.
- A call event is a two-way synchronous communication.
- Signal events can be specified as UML classes.

### Signal Events

<table>
<thead>
<tr>
<th>«signal»</th>
<th>FlightDeparture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>airline</td>
</tr>
<tr>
<td></td>
<td>flightNumber</td>
</tr>
<tr>
<td></td>
<td>city</td>
</tr>
<tr>
<td></td>
<td>date</td>
</tr>
<tr>
<td></td>
<td>time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>«signal»</th>
<th>MouseButtonPushed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>button</td>
</tr>
<tr>
<td></td>
<td>location</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>«signal»</th>
<th>SelectionChanged</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>targetControl</td>
</tr>
<tr>
<td></td>
<td>selectionIndex</td>
</tr>
</tbody>
</table>
Change events

- A change event is an event that is caused by the satisfaction of a boolean expression
  - UML specifies that the expression is continually tested
  - An implementation would not continuously check the expression, but at least often

Examples:
- when (room temperature < heating set point)
- when (room temperature > cooling set point)
- when (battery power < lower limit)
- when (tire pressure < minimum pressure)
A time event is an event that is caused by the occurrence of an absolute time or the elapse of a time interval.

Examples absolute time:
- at (January 1, 2010)
- at (20:00)

Examples time interval:
- after (10 seconds)
- after (10 days)
Transitions and guard conditions

- A transition is the change from one state to another
  - E.g. A phone line transitions from “Ringing” state to “Connected” when somebody picks the phone up
- A boolean expression can be used to add constraints in the firing of a transition
  - Interesting when more than one transition can be selected at a given time
Transition effects and do-activities

- Mealy automata actions correspond to UML statecharts’ transition effects
  - A transition effect can be an assignment or the call to an operation

```
<table>
<thead>
<tr>
<th>State</th>
<th>Action/Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>right button down / display pop-up menu</td>
</tr>
<tr>
<td></td>
<td>right button up / hide pop-up menu</td>
</tr>
<tr>
<td>Menu visible</td>
<td></td>
</tr>
</tbody>
</table>
```

- UML statecharts can also specify actions attached to state nodes (as for Moore automata)
  - A “do-activity” is an activity that should execute continuously for an extended time

```
<table>
<thead>
<tr>
<th>State</th>
<th>Action/Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper jam</td>
<td>do / flash warning light</td>
</tr>
</tbody>
</table>
```
Entry/Exit Activities

An entry activity is performed whenever the state is entered.

Whenever the state is exited, by any outgoing transition, the exit activity is performed first.
Order of activities

After first “off” event

- print(“exiting”)  
- print(“to off”)  
- turnLightOff()  

After second “off” event

- print(“exiting”)  
- print(“needless”)  
- turnLightOff()
Event handling within a state

In this case “off” event is handled bypassing both the entry and exit activities

- print(“skipped”)

LightIsOff

entry/turnLightOff()
exit/print(“exiting”)
off/print(“needless”)

LightIsOff

entry/turnLightOff()
off/print(“skipped”)
exit/print(“exiting”)
More about transitions guards

All transitions are triggered by the same event. Note that guards can be arbitrarily complex.
Static control branching

- Merely a graphical shortcut for convenient rendering of decision trees

It is easier to see that there is a single event

The behavior is the same as in the previous diagram

![Diagram]

- Selling
  - bid
  - [value < 100] / reject
  - [value >= 200] / sell

- Unhappy

- Happy

- [((value >= 100) AND (value < 200)) / sell]
Dynamic control branching

- **Choice pseudo-state**: guards are evaluated at the instant when the decision point is reached.

This allows some extra actions to be performed:

- **Selling**
  - [gain < 100] / reject
  - [gain >= 200] / sell
  - [(gain >= 100) AND (gain < 200)] / sell

- **Happy**

- **Unhappy**
  - bid / gain = computeGain(value)
Exercise 3

- Prepare the state diagram for a washing machine
  - On / Off button
  - Start button (but no stop button)
  - Feedback is given on the current stage (soaking, rinsing, draining, drying)
  - Three washing programs
    - Regular
    - Delicate (no soaking)
    - Super delicate (no soaking, no drying)
  - Off can be clicked only before starting, or after finishing
Exercise 4

A simple digital watch has a display and two buttons to set it, the A button and the B button. The watch has two modes of operation, display time and set time. In the display time mode, the watch displays hours and minutes, separated by a flashing colon.

The set time mode has two submodes, set hours and set minutes. The A button selects modes. Each time it is pressed, the mode advances in the sequences: display, set hours, set minutes, display, etc. Within the submodes, the B button advances the hours or minutes once each time it is pressed. Buttons must be released before they can generate another event. Prepare a state diagram of the watch.