Behavioral Modeling in UML (part I)

Vadim Malvone
vadim.malvone@telecom-paris.fr

(theses slides are inspired by tutorials given at the OMG and the material of prof. Élie Najm)
The main UML diagrams

- Description of requirements
  - Use case diagram
- Structural view (static aspect of a model)
  - Class diagram
  - Object diagram
- Functional view (interaction between objects)
  - Sequence diagram
  - Communication diagram
- Behavioral view (dynamics of objects)
  - Activity diagram
  - State machine
Structural view

Client
- id: int
- name: string
- address: string
- prepay()
  - 1
  - 1..*

Reservation
- id: int
- arrival_date: date
- departure_date: date
- prepaid: bool
- prepay()
  - 1..*
  - 1..*

Room
- number: int
- number_beds: int
- price: int

Real system

:Client
  - id: 33
  - name: Norbert
  - address: Paris

:Reservation
  - id: 1222
  - arrival_date: 3/8/15
  - departure_date: 6/8/15
  - prepaid: False

:Room
  - number: 3
  - number_beds: 2
  - price: 75

:Reservation
  - id: 1245
  - arrival_date: 3/9/15
  - departure_date: 4/9/15
  - prepaid: False

:Client
  - id: 42
  - name: Dupont
  - address: Lyon

:Reservation
  - id: 1246
  - arrival_date: 7/9/15
  - departure_date: 9/9/15
  - prepaid: True

:Room
  - number: 10
  - number_beds: 2
  - price: 75

:Room
  - number: 12
  - number_beds: 1
  - price: 60

:Room
  - number: 14
  - number_beds: 1
  - price: 60
Behavior of “classic” Classes

- Receive and process operation invocations
- Uniform behavior (does not change over time)
Need for state-based behavior

- A "machine" whose output behavior depends not only on the current input, but also on the history of inputs

- The History of inputs can be characterized by a state that is an abstraction of the past experience.

=> State Machine Diagram (UML 2.0)
State Machine Diagram (example)

A lamp controller

Lamp On

Lamp Off

on

off

on

off

A lamp controller
Actions and Outputs (I)

When the machine changes state it can do actions:

Mealy machine

Moore machine

Question: are they equivalent?  NO!
Actions and Outputs (I)

- Equivalences between diagrams?

Mealy machine

Moore machine
Actions and Outputs (II)

- When the automaton changes state, it can perform an output action: synchronous call or asynchronous signal.

- Synchronous call:
  - Lamp On
  - Lamp Off

- Asynchronous signal:
  - Lamp On
  - Lamp Off

Example:
- Lamp on():
  - On
  - Off

- Send_signal(lamp_on):
  - On
  - Off
State machine attached to a class

- Operations and Attributes of the class used in the actions ("extended state")

```
CTR 

ctr : Integer

operations
```

```
Lamp On

on

on/ctr := ctr + 1

off

Lamp Off

off

CONTROLLER

ctr : Integer

operations
```
State machine reacts to events

Types of events

- Interaction:
  - Synchronous call of an operation on an object (call event)
  - Reception of an asynchronous signal (signal event)

- Temporal occurrence (time event)
  - Expiration of a deadline
  - Arrival at a predefined time

- Changing the value of a variable (change event)
Signals and Operations: declarations

Operations: declared in the classes

| attributes
| operations

Signals: declared independently (stereotype classes)

<< signal >>
Shutdow

Pure signal contains no data

<< signal >>
EngineTemp

temp : Integer

Signal that contains data
Simple State Machine Diagram

- **Initial pseudostate**: 
- **State**: 
- **Trigger**: 
- **Final state**: 

**Transition**:

- **Ready**

**Action**:

- Stop/ $ctr := 0$

- Stop
Actions Entry and Exit

Lamp On

entry/lamp.on()
exit/lamp.off()
Order of Actions: Simple Case

- The exit action precedes the transition action
- The entry action is executed after the transition action

Resulting action sequence:

```python
print(exiting)
lamp.off()
print(to off)
lamp.off()
```

```
Lamp On
entry/lamp.on()
exit/print(exiting)
```

```
Lamp Off
entry/lamp.off()
exit/printf(exiting)
```

- off / print(to off)
- off / print(needless)
Internal transitions

- Avoid the execution of entry and exit actions

**Lamp Off**

- entry/lamp.off()
- exit/print(exiting)
- off/print(itsOff)
Guards

- Introduce conditional execution of transitions
- Boolean predicates without side effects

<table>
<thead>
<tr>
<th>&lt;&lt; signal &gt;&gt;</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v: int</td>
</tr>
</tbody>
</table>

- bid \[v < 100\] / reject
- bid \[v \geq 200\] / sell
- bid \([(v \geq 100) \& (v < 200)]\) / sell
Conditional branching

Graphic shortcut for better rendering the decision trees

[v < 100] / reject

[v >= 200] / sell

[v >= 100) & (v < 200)] / sell

Unhappy

[Selling

Happy

bid

[v >= 200] / sell

[v >= 100 & (v < 200)] / sell

Unhappy

[v < 100] / reject

[v >= 200] / sell

[v >= 100 & (v < 200)] / sell

Unhappy
Conditional branching

Using a state instead of the pseudo-state

- 
  - \([v < 100]\) / reject
  - \([v >= 200]\) / sell
  - \([(v >= 100) & (v < 200)]\) / sell
  - \([v >= 200]\) / sell
Conditional branching and transitions

- **Selling**
  - bid \([v < 100]\) / reject

- **Unhappy**
  - bid \([v >= 160 \& (v < 200)]\) / sell

- **Happy**
  - bid \([v >= 200]\) / sell

For \(v = 120\):

- Final state = ?
Conditional branching and transitions

\[ v = 120 \]
\[ \text{final state} = ? \]

- \( [v < 100] / \text{reject} \)
- \( [v \geq 200] / \text{sell} \)
- \( [(v \geq 160) \& (v < 200)] / \text{sell} \)
- \( v = 120 \)
- \( \text{Unhappy} \)
- \( \text{Selling} \)
- \( \text{Happy} \)
Conditional branching and transitions

\[
\begin{align*}
[v < 100] & / \text{reject} \\
[v >= 160] & / \text{sell} \\
[v < 200] & / \text{sell} \\
[v >= 200] & / \text{sell} \\
[(v >= 160) & (v < 200)] & / \text{sell} \\
\end{align*}
\]

\[v = 120\]

final state = ?
Conditional branching and transitions

\[ v = 120 \]
\[ \text{final state} = ? \]

\[ v < 100 \] / reject

Selling → bid

Test value

\[ v \geq 200 \] / sell

[else] / sell

Unhappy

[v >= 200] / sell
Transitions and non-determinism

![Diagram showing transitions](image)

- **Selling**
  - $v < 100$ / reject
  - $v \geq 50$ / sell

- **Test value**
  - bid
  - $v \geq 50$ / sell
  - [else] / sell

- **Unhappy**
- **Happy**

$v = 75$
final state = ?
Cascading Guard Transitions and Tests

\[ v = 75 \]
\[ \text{final state} = ? \]

\[
\begin{align*}
\text{Selling} & \quad \text{v < 100} / \text{reject} \\
\text{Test value1} & \quad \text{bid} \\
\text{Test value2} & \quad [\text{else}] \\
\text{Unhappy} & \quad [\text{else}] / \text{sell} \\
\end{align*}
\]
Cascading Guard Transitions and Tests

- **Selling**
  - **Test value1**
    - ![v < 100] / reject
    - ![v >= 50] / sell
    - ![else]
  - **Test value2**
    - ![else] / sell
    - **Unhappy**

- **Happy**
  - v = 75
  - final state = ?

- **v = 75**
  - final state = ?

- **[v < 100] / reject**
- **[v >= 50] / sell**
- **[else]**
- **[else] / sell**
Syntax of transitions (I)

\[ \text{trigger [guard] / action} \]
Syntax of transitions (II)

S1 \[ a \geq 2 \] / c:=14 \rightarrow S2

a
b = 1
Syntax of transitions (III)

\[ a \ [b > 2] \ / c := 14 \]

S1 \rightarrow S2

\[
\begin{align*}
  a &= 3 \\
  b &= 3
\end{align*}
\]
Syntax of transitions (IV)

S1 → S2  [b>2]  S3  \(c:=14\)  S4


S5
Syntax of transitions (V)

S1 → S2

S2 → S3

S3 → S4

S4 → S5

a

[b>2]

c:=14

a

b = 1
Syntax of transitions (VI)

S1 \xrightarrow{a} S2 \xrightarrow{[b>2]} S3 \xrightarrow{c:=14} S4

a
b = 3
Time modeling

c := 0

ready

start / c := 0
tm(10) / print(c)
count

e / c := c + 1
Time modeling

c := 0

ready

start / c := 0

tm(10) / print(c)

count

e / c := c + 1