Unified Modeling Language 2

State machines
History and predecessors

• 1950’s: Finite State Machines
  • Huffmann, Mealy, Moore

• 1987: Harel Statecharts
  • conditions
  • hierarchical (and/or) states
  • history states

• 1990’s: Objectcharts
  • adaptation to object orientation

• 1994: ROOM Charts
  • run-to-completion (RTC) step
Usage scenarios

- **Object life cycle**
  - Behaviour of objects according to business rules
  - in particular for active classes

- **Use case life cycle**
  - Integration of use case scenarios
  - Alternative: activity diagrams

- **Control automata**
  - Embedded systems

- **Protocol specification**
  - Communication interfaces

Modelling with UML, with semantics
States and transitions

- State machines model behaviour
  - using **states** interconnected …
  - with **transitions** triggered …
  - by **event** occurrences.

![State machine diagram]

<table>
<thead>
<tr>
<th>Simple State</th>
<th>Transition</th>
<th>Effect (CallAction)</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>pay()</td>
<td>cancel()</td>
<td>passenger.creditMiles/self</td>
</tr>
</tbody>
</table>
Modelling with UML, with semantics

Relation to class diagrams

- State machines are defined in the context of a BehavioredClassifier.

- **Context** defines which
  - events can occur
  - features are available

- **Operation** corresponding **CallEvent**
- **CallAction** called **Operation**

```
<table>
<thead>
<tr>
<th>Booking</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
</tr>
<tr>
<td>pay()</td>
</tr>
<tr>
<td>cancel()</td>
</tr>
<tr>
<td>startOff()</td>
</tr>
<tr>
<td>change()</td>
</tr>
<tr>
<td>handle()</td>
</tr>
</tbody>
</table>

Reserved → Booked → StartedOff

- change()[kind <> #Economy]
- pay()
- cancel()
- startOff()
- handle() / passenger.creditMiles(self)

Passenger

name : Name
creditMiles(b : Booking)

Flight

* passenger
* flight
* miles : int
```

Modelling with UML, with semantics
Triggers and events (1)

- **SignalEvent**
  - completion event (no explicit trigger)

- **ChangeEvent**

- **TimeEvent** (relative)
  - deferred event

### Boarding

- **BoardingControl Ready**
- **ReadBoardingPass**
  - entry / check validity
  - exit / read passenger id
  - closeFlight / defer
- **CheckBoardingPass**
  - entry / start query
  - do / blink
  - closeFlight / defer
- **AcceptBoardingPass**
  - entry / eject
  - do / release turnstile
  - closeFlight / defer
  - after(10s) / block turnstile
- **RejectBoardingPass**
  - entry / eject
  - closeFlight / defer
  - when(turnstile sensor="turn") / block turnstile

Modelling with UML, with semantics
Triggers and events (2)

- **CallEvent**
  - receipt of a (a)synchronous Operation call
  - triggering after Behavior of Operation executed

- **SignalEvent**
  - receipt of an asynchronous Signal instance
  - reaction declared by a Reception for the Signal

- **TimeEvent**
  - absolute reference to a time point (at $t$)
  - relative reference to trigger becoming active (after $t$)
    - presumably meaning relative to state entry

- **ChangeEvent**
  - raised each time condition becomes true
    - may be raised at some point after condition changes to true
    - could be revoked if condition changes to false
Triggers and events (3)

- **Completion event**
  - raised when all internal activities of a state are finished
    - do activity, subregion
    - no metamodel element for completion events
  - dispatched before all other events in the event pool

- **Deferred events**
  - events that cannot be handled in a state but should be kept in the event pool
    - reconsidered when state is changed
    - no predefined deferring policy

- **Internal transitions**
  - … are executed without leaving and entering their containing state
    - normally, on transition execution states are left and entered

\[ S \]

*trigger [guard] / effect*
Behaviours

Modelling with UML, with semantics
How state machines communicate

signals: *asynchronous* (no waiting)
calls: *asynchronous* or *synchronous* (waiting for RTC of callee)

*No* assumptions are made on timing between
event occurrence, event dispatching, and event consumption.

Event occurrences for which no trigger exists may be discarded
(if they are not deferred).
Hierarchical states (1)

- Hierarchical states allow to **encapsulate** behaviour and facilitate reuse.
- However, they are rarely used this way.
- UML 2.0 provides concepts supporting this usage.
  - entry and exit points

Transition triggering is **prioritized** inside-out, i.e., transitions deeper in the hierarchy are considered first.
Hierarchical states (2)
Orthogonal regions

- **Simple State**: containing no Region
- **Composite State**: containing at least one Region
  - simple composite State: exactly one
  - orthogonal composite State: at least two

Orthogonal states are “concurrent” as a single event may trigger a transition in each orthogonal region “simultaneously”
Forks and joins

**fork** Pseudostate
(one incoming, at least two outgoing Transitions; outgoing Transitions must target States in different Regions of an orthogonal State)

all Regions must be entered simultaneously

**join** Pseudostate
(restrictions dual to forks)

all Regions are left simultaneously (if FinalStates are reached)
Entry and exit points (1)

- Entry and exit points (Pseudostates)
  - provide better encapsulation of composite states
  - help avoid “unstructured” transitions

![State machine diagram](image)

- Entry point (on border of state machine diagram or composite state)