Exercise Unit 2: Modeling Paradigms - RT-UML

- UML: The Unified Modeling Language
- Statecharts
- RT-UML in AnyLogic
**RT-UML: UML**

- **Unified Modeling Language**
  - a mix of visual modeling languages for software engineering
  - pragmatically successful
    - well-developed tools (IBM Rational Software, etc.)
    - broad acceptance in industry
  - history:
    - 1996 defined by the "three amigos":
      Grady Booch, Jim Rumbaugh, Ivar Jacobson
    - 1997 standardized as UML 1.1 by Object Management Group (OMG)
    - since 2005: UML 2.0
    - since 2008: UML 2.2 (latest specification)
  - problems: semantics and interaction of the formalisms
  - no description of time and randomness

- **Diagram types**
  - structural diagrams
  - behavior diagrams
  - physical diagrams and diagrams for model management
**RT-UML: UML**

- **Structural Diagrams**
  - Class Diagram
  - Component Diagram
  - Object Diagram
  - Collaboration Diagram
  - Composite Structure Diagram (= active objects with ports)
  - Use Case Diagram

- **Behavior Diagrams**
  - Sequence Diagram
  - Timing Diagram
  - Activity Diagram (similar to Petri nets)
  - Interaction Overview Diagram
  - State Machine Diagram (= statecharts)
  - Communication Diagram

- **others**
  - Deployment Diagram, Package Diagram
Real-Time UML

RT-UML: communicating active objects

- in UML 2 conform variant:
  - message-sending active objects as Composite Structure Diagrams
  - (internal) statecharts as State Machine Diagrams (anyhow)

- applications:
  - initially for embedded, real-time and communication systems
  - also well suited for manufacturing systems, workflow systems, ...

RT-UML for simulation

- we need additional constructs to express time or randomness
  - defined as standard-conform extensions of UML ("profiles")
  - implemented in proprietary tool realizations

AnyLogic: flexible modeling paradigm for simulation

- uses ideas of RT-UML (but not conform to UML 2)

- extensions for
  - deterministic and random times
  - probabilities
  - discrete and continuous state changes
Modeling Paradigms: RT-UML

- UML: The Unified Modeling Language
- Statecharts
- RT-UML in AnyLogic
RT-UML: Statecharts

- Model of object's event-oriented behavior
  - describes all possible states of an object
  - describes how state changes as the result of events

Diagram:
- Initial state
- Checking state with do/check item
  - Transition to Waiting state on item received [some items not in stock]
  - Transition to Dispatching state on item received [all items available]
- Waiting state
  - Self-transition on item received [some items not in stock]
- Dispatching state with do/initiate delivery
  - Transition to Delivered state on item received [all items available]
- Delivered state

Simulation and Modeling I
RT-UML: Statecharts

Representation as a state machine

- **Activity**
  - associated with states: *Do / Activity*
  - ongoing non-atomic execution within state machine
  - may be interrupted

- **Action**
  - associated with transitions
  - results in a change in state
  - occurs quickly and is uninterruptible (atomic)

- **Transition**
  - relationship between two states
  - if object is in the first state, it will perform certain actions and enter the second state when a specified event occurs and specified conditions/guards are satisfied
  - transition label: *Event [Guard] / Action*
RT-UML: Statecharts

Transition label: *Event [Guard] / Action*
- event
  - can trigger a state transition
- guard
  - logical condition
  - returns "true" or "false"
  - guarded transition occurs only if guard evaluates to "true"
- state responds (conditionally) to event with action

Extensions
- superstates / composite states
  - combine several substates, which inherit transitions from superstate
- concurrent sections:
  - system may be in different states at same time ("AND-states")
Modeling Paradigms: RT-UML

- UML: The Unified Modeling Language
- Statecharts
- RT-UML in AnyLogic
RT-UML in AnyLogic

- AnyLogic’s modeling paradigm is a variant of RT-UML

- inclusion of time and randomness
  - events are triggered after certain times (deterministic + random)
  - competing events can be triggered with a probability

- basic model elements
  - active objects: model real-world objects
  - statecharts: internal behavior of active objects
  - ports: asynchronous message passing
  - variables: shared variables
RT-UML in AnyLogic

- Active objects
  - main building blocks of AnyLogic models
  - model real-world objects
  - mapped to Java classes
  - structure diagram:
RT-UML in AnyLogic

Elements of structure diagram:

- **this object**
  - pane of current structure diagram
  - public elements appear in dashed boundaries in structure diagram
  - encapsulated objects may be placed anywhere on pane
  - solid frames: only presentation shapes to depict class boundaries

- **encapsulated objects**
  - displayed by red circle with letter `A`
  - instances of other active objects
  - some interface elements, like ports, displayed

- **port**
  - displayed as a small square
  - port queues need to be implemented, e.g., via collection variables
  - may be defined as public or private (via check box)

- **port references**
  - public ports of encapsulated objects
**RT-UML in AnyLogic**

- **variables**
  - plain or collection variables
  - plain variables displayed as orange circles with ‘V’
  - collection variables (array or list) displayed as three orange dots
  - may be declared as public or private (via check box)

- **parameters**
  - similar to plain variables
  - should be used to initialize different model instances
  - may be propagated through model hierarchy

- **statechart**
  - activities executed inside active object

- **(dynamic) events**
  - used to schedule events at user-defined times with user-defined actions
  - a dynamic event deletes itself after its action
  - a simple event survives and can be restarted
**RT-UML in AnyLogic**

- **connectors**
  - a line between two ports
    - for message transport
  - a connection from port to statechart
    - via `port.map(statechart)`: incoming messages are signaled to statechart (message trigger)

```
this object
encapsulated object
port reference
port (without queue)
statechart
event
dynamic event
text box
```

```
connector
parameter
variable
collection variable (may be used to implement port queue)
```
RT-UML in AnyLogic

Message passing

- primary interaction mechanism via `port.send(message)`:
  - messages
    - can model real-world objects, e.g., parts, products, people, ...
    - can be of any Java class
    - can contain arbitrary data (e.g., attributes, packet lengths, ...)
  - transmission of messages
    - sent and received at ports
    - asynchronous
      - sending does not affect receiver
      - receiving does not affect sender
    - immediate:
      - messages go to the destination port without advancement of the simulation clock
    - broadcasting possible
      - messages with multiple recipients
RT-UML in AnyLogic

- **Ports**
  - bidirectional
  - message processing depends on direction of message

```
receive()
```
```
port
```
```
onReceive()
```
```
send()
```
```
RT-UML in AnyLogic

When a message is sent at a port

1. **onSend()** method called: *On send action* of the port is executed. If it returns false, message not processed further.

   Otherwise:

2. If public port:
   - message forwarded along port connections outside active object
   - **send()** method is called at all public ports to which this port is mapped
   - **receive()** method is called at all private ports and connected ports of encapsulated objects

3. If private port:
   - message forwarded along port connections inside active objects
   - **send()** method is called at all public ports to which this port is mapped
   - **receive()** is called at all connected ports of encapsulated objects
When a message is received on a port

1. Message type check:
   check if message can be cast to message type of the port.
   If not, message is discarded.

2. onReceive() method is called:
   On receive action of the port is executed.
   If it returns false, message not processed further.
   Otherwise:

3. If port is public:
   message forwarded along port connections inside active object,
   receive() is called on all ports of encapsulated objects and private
   ports mapped to this port.
   If port is private:
   above does not happen.

4. Method receiveMessage() called at all statecharts that are mapped to the port: trigger of type message for transitions
RT-UML in AnyLogic

- Role of statecharts in AnyLogic
  - behavior of active objects
  - states
  - events causing state transitions
  - actions that result from a state transition

- An AnyLogic statechart:
RT-UML in AnyLogic

State
- represents a location of control with set of reactions to conditions and/or events
- composite states ("OR-states")
  - control always resides in one simple state
  - set of reactions: union of current simple state and all composite states containing it
RT-UML in AnyLogic

Transition

- controls flow between two states
- statechart may enter other state
  - if specified trigger event is signalled while statechart is in particular state
  - and specified guard condition is true
- actions may be associated
  - with exiting the state (exit action)
  - with the transition (transition action)
  - with entering next state (entry action)
- may freely cross simple and composite state borders
- internal transition
  - lies inside state
  - both start and end on the border of state
  - neither entry nor exit actions are executed when transition is taken
  - current simple state will not be exited
RT-UML in AnyLogic

- **Initial state pointer**
  - Points to initial state within level of hierarchy
  - When control is passed to composite state
    - Simple state is found inside by following initial state pointer
    - This state becomes current one
  - Exactly one initial state pointer on each level
RT-UML in AnyLogic

- **Final states**
  - termination point of a statechart
  - action is executed
  - statechart terminates activity
  - transitions may not exit final state

- **History States**
  - references to most recently visited state
  - shallow history: to composite or simple state on the same hierarchy level
  - deep history: to simple state within composite states
  - selected via check box
  - example: E was the most recently visited state upon entering A
RT-UML in AnyLogic

Branch
- Represents transition branching or connection point
- One default exit point:
  - Transition taken, if all other outgoing transitions are not enabled
  - Runtime error, if no transitions are enabled and no default branch exit
- Can model immediate choices

Simulation and Modeling I  Modeling with RT-UML 24
RT-UML in AnyLogic

行动计划执行顺序

1. 状态退出动作
   - 从旧的简单状态开始，向上遍历状态层次结构到最高退出的复合状态
2. 过渡动作
3. 状态进入动作
   - 从最高的进入的复合状态开始，向下遍历状态层次结构到达新简单或假状态（分支或历史状态）
4. 如果控制进入一个假状态：
   - 其动作代码被执行
   - 控制然后立即离开假状态
   - 控制只能在简单状态下停止
RT-UML in AnyLogic

- Current state N, transition T1:
  - N state exit action
  - M state exit action
  - T1 transition action
  - branch action
- T2 or T3, assume T2:
  - T2 transition action
  - I1 initial state pointer action
    (not entry/exit of state L)
  - M state entry action
  - I2 initial state pointer action
  - N state entry action
- If T3 is selected:
  - L state exit action
  - K state exit action
    (no action of state V)
  - T3 transition action
  - S state entry action
  - P state entry action
  - branch action
  - P state exit action
  - T4 transition action
    (guard open since branch exit)
  - Q state entry action
  - I3 initial state pointer action
  - R state entry action
Triggering a transition

- statechart enters a simple state:
  - triggers of outgoing transitions are collected

- statechart begins to wait for triggers to be signaled
  - when trigger is signaled, guard of the transition is evaluated
    - if guard is true, transition may be taken
    - if guard is false, wait for other signal
  - when several triggers are signaled at the same time & corresponding guards are true
    - transition to be taken is chosen randomly
      (equiprobabilistic serialization of concurrent events)
RT-UML in AnyLogic

- Types of triggers
  - **timeout**
    - triggered after specified amount of time after entering state
    - when syntax of trigger evaluates to `double` (either fixed value or sampled from random distribution, see methods in class `Utilities`)
    - example: `2.56`
    - value `0.0` triggers transition immediately
  - **rate**
    - triggered after exponentially distributed amount of time after entering state (special case of timeout)
    - when syntax evaluates to `double`
    - example: `2.56` (rate of exponential distribution)
    - identical to `timeout` with `exponential(2.56)`
  - **condition**
    - triggered when an expression becomes true (equation solving)
    - when the syntax evaluates to `boolean`
    - example: `x > 10`
RT-UML in AnyLogic

- message
  - triggered by messages arriving to the statechart:
    - via connected ports (e.g., `port.map(statechart)`): message arriving to port is forwarded to statechart via method `receiveMessage(message)` of the statechart
    - via method `receiveMessage(message)`: posts message of specified type to the statechart without queueing
    - via method `fireEvent(message)`: adds a message of specified type to the statechart queue (to be consumed in zero-time steps)
      - message may be of any basic type or Java class
      - type match initiates trigger, if in corresponding state
      - trigger may also depend on other properties of message
        - if message equals: e.g., a particular String
        - if expression is true: use `msg` to refer to current message
      - method `getEvent()` of statechart retrieves message from queue (e.g., in action code)
RT-UML in AnyLogic

- **Summary of basic model elements**
  - active objects: model real-world objects
  - ports: asynchronous message passing
  - (collection) variables
  - statecharts: internal behavior of active objects
  - all can be visualized

- **Summary of statechart behavior**
  - states represent system states (states of object)
  - transitions represent state changes
  - transition label: *event [guard] / action*
  - a transition may be taken
    - if the trigger *event* occurs
    - and the specified *guard* condition is true
  - an *action* can be performed when the transition is taken
  - all statecharts execute concurrently
RT-UML in AnyLogic

possible events

- **timeout**: after a fixed time has elapsed
  
  (timeout may be computed according to some distribution)

- **rate**: after an exponentially distributed time has elapsed (special case)

- **condition**: when an expression becomes true

- **message**: after arrival of a message at the statechart queue; occurs,
  
  - when Java object arrives at port, which is connected to statechart
    
    (via `port.map(statechart)`)
  
  - when explicitly called from Java code (via
    
    `statechart.receiveMessage()` or `statechart.fireEvent()`)

examples in chapter *Modeling Paradigms*

- queueing systems
- Stop and Wait protocol
- a Wireless LAN model
- ...

Simulation and Modeling I
Summary of AnyLogic

- High-level simulation environment
  - general purpose
  - discrete, continuous (and hybrid)
  - modeling paradigm is a variant of RT-UML (real-time Unified Modeling Language)

- Underlying is discrete-event simulation
  - simulation clock is always advanced to the time of the next event (anywhere in the model) and the event is then executed (transition is taken and the actions are performed)
  - time ties are broken randomly

- Java-based
  - textual model parts in Java
  - executable model + simulation engine mapped on Java
  - other Java code can be linked