Development of Rigorous Adaptive Information Systems

Dr. Nasreddine Aoumeur
FIN, ITI, DB group
aoumeur@iti.cs.uni-magdeburg.de

Course Site:
wwwiti.cs.uni-magdeburg.de/~aoumeur
wwwiti.cs.uni-magdeburg.de/iti_db/lehre/oois/inde
Information Systems: Working definition

- \textit{reactive} systems (i.e. in continuous interaction with their environment), with
- large \textit{amount} of immutable and non-immutable \textit{data} (i.e. fixed and changing) and, with
- \textit{processes and activities} for exhibiting behaviors on these (state-less and -full) data.
Conceptual Modelling of IS in general

- **Structural aspects**
  - State-less and -ful DATA
  - E/R or Object paradigm

- **Behavioural aspects**
  - Processes and Rules
  - Petri Nets
Conceptual Modelling of IS in UML

State-less and stateful DATA
- Use Cases
- Class Diagrams
- Object Diagrams
- Object Constraint Language

Processes and Rules
- Sequence Diagrams
- Collaboration Diagrams
- State Diagrams
- Activity Diagrams
- Component / deployment diagrams (implementation)

For all C in ....

Structural aspects

Behavioural aspects
Most required UML-diagrams in the conceptual level of information systems consist in:

1. **Conceive a use-case diagram**
   - identify actors
   - identify major functional requirements

2. **Conceive an initial Class diagram**
   - discover principle classes
   - represent important relationships
   - Add pre- post- OCL constraints.

3. **Conceive sequence diagrams**
   - Examine possible (inter-class) object interactions

4. **Conceive statechart diagrams**
   - Capture object-states change in response to methods and messages
To model the requirements of a system ... 

- Identify all actors (users of the system).
- Identify the needs, from the system, of each individual actor.
- Make each need a use case.
- Identify redundant behavior within your set of use cases, and factor it into common base-class use cases (generalization).
- Do the same for actors.
- Show the relationships between actors and use cases.
UML: Use Cases --- Library illustration

- Student borrower
- Staff borrower
- Browser

- Borrow book
- Reserve book
- Return book
- Browse catalogue
- Check member status
- Register member
- Usage report
- Update catalogue
- Return late book

Counter staff
Manager
UML: Class Diagrams concepts Compacted
UML: Class Diagrams---Illustration
UML: Class Diagrams---Illustration

Element

Carbon

Hydrogen

0..4

0..1

0..4

0..4
An object is an instance of class. There are at least three kinds of objects:

- **Entity objects**
  - Represent the entities of the domain model
  - Often represent persistent data

- **Boundary objects**
  - Implement the interaction with users
  - Contribute to UI components
  - Subject to most frequent modifications

- **Control objects**
  - Implement the interactions with the user
**UML: Object Diagrams---Concept**

- An **object diagram** is a kind of class diagram
- It represents a **system snapshot**
- It specifies the **values of object attributes and associations** in a particular moment of time
- **Different points in time** may correspond to **different object diagrams** (snapshots)
UML: Object Diagrams---Illustration

Objects with their attribute values and links
UML: Object Diagrams---Illustration

- Link (pointer, reference)
- Anonymous instance of class Hydrogen
- Object named c of class Carbon

Methan
Objects with their attribute values and links form a system snapshot.
UML: Object Diagrams---Illustration

- Heap (Machine Level Representation)
  
  **Heap model**
  - Stack for the local variables
  - Heap for the objects

- Link (reference, pointer)

- Stack for the local variables
  - Stack for the objects

- Object of class Point
  - $x = 1$
  - $y = 2$

- Object of class Point
  - $x = 2$
  - $y = 1$
UML Behavior : Sequence Diagrams--Concepts

• A sequence diagram presents an interaction, i.e. is a set of messages between objects

• Sequence diagrams can be used to
  – Understand the interaction and communication
  – define test cases
  – generate code (method body)
**UML Behavior: Sequence Diagrams -- Concepts**

- **message()**
  - A **synchronous message** to another object or design class.

- **message()**
  - An **asynchronous message** to another object or design class.

- **message()**
  - A **call to a method** at the same object or class.
UML Behavior: Sequence Diagrams--Concepts

Allow **modelling interactions** between **objects** with a same class or between different classes, using **message exchanges**.

- **Message types**
  - synchronous
  - return
  - asynchronous

- **Instance creation & destruction**

- **Conditional choice**
  - [ condition 1]
  - [ condition 2]

- **Iteration**
  - n times
**UML Behavior : Sequence Diagrams--Concepts**

- **Object**: Represents the object name.
- **Method execution**: Shows the execution of methods.
- **Object life line**: Indicates the life cycle of an object.
Anonymous object

Object name of class C

:Use

new C()

name : C

:Class

Anonymous object

Method call m with parameter a

m (a)

otherMethod ()

Object name of class C

lifeline
UML Behavior: Sequence Diagrams--Concepts

Object name with a unique identity and to which messages can be sent. The \underline{has} a state defined by the attributes.

A message sent from one object to another

Link connecting objects (instantiates association)
UML Behavior: Sequence Diagrams--Illustration

c:Customer

<<signal>>
Withdraw(c,m)
a.debit(m)

<<signal>>
Deposit(c,m)
a.credit(m)

debit(m)

credit(m)

b: bank

a: Account
Run:

```plaintext
insertCard(); requestPin(); typePin(); activateSession(1,22);
requestValidation(1, 22); validateUser(1); confirmValidation()
```
Run 1:
send(informA); receive(informA); send(informB); receive(informB);
send(confirmB); receive(confirmB); send(confirmA);
receive(confirmA);

Run 2:
send(informA); send(informB); receive(informA);
receive(informB); send(confirmA); send(confirmB); receive(confirmB);
receive(confirmA)

...
UML Behavior: Sequence Diagrams--Illustration

User

- Order-Item(i,s)
- Put_basket(i)
- Check-Out

Shop

- Available(i)

Item

Invoice

Credit Card

Account

- PrepareInvoice
- DebitCard(sum)
- Ok(sum)
- Debit(sum)
A state machine ...
- Models the behavior of an individual object (statechart diagram) or a society of objects (activity diagram).
- Is a dynamic model of the system.
- Is event driven.
  » Events trigger activities which, in turn, trigger actions.
  » Actions are atomic.
  » Actions may cause the return of a value or the change of state of an object.
- State machines come in two varieties:
  » Activity diagrams.
  » Statechart diagrams.
UML Behavior: State Machine--Concepts

- A state machine describes all possible behaviours of dynamic active object
- An active object is an object possessing its own thread
- Behaviour is modelled by a graph of state nodes interconnected by one or more joined transition arcs that are triggered by the dispatching of series of event instances
UML Behavior: State Machine--Concepts

• A state is a condition during the life of an model element or an interaction during which it satisfies some condition, performs some action, or waits for some event.

• A composite state is a state that has a graphical decomposition.
**UML Behavior: State Machine--Concept**

- **States**
  - Special categories of states
    - **Initial state** - indicates the initial starting state for the state machine or a substate.
    - **Final state** - indicates the state machine’s execution has completed.
  - Real-time state machines frequently do not include a final state.
    - Neither initial or final states contain any of the parts found in traditional states.
UML Behavior: State Machine--Concepts

Traffic Lights

Red

Yellow

Green

**Run**: Read; Yellow, Green; Red; ...
**UML Behavior: State Machine--Concepts**

- **Initial pseudostate**
- **Transition**
- **Entry action** performed when the state is entered
- **Do action** performed in the state
- **Top state**
- **Substate**
- **Trigger** an event triggering the transition
- **Entry action** or doThis
- **Do/doMore** action
- **Condition**
- **Final state**
- **Entry action** performed when the state is entered
- **Move**
- **Stop**
- **Start[condition]/inform**
**UML Behavior: State Machine--Concept**

- **States**
  - A state is a condition or situation during the life of an object in which it satisfies some condition, performs some activity, or waits for some event.
  - A state may include …
    - Name
    - Entry/exit actions
    - Internal transitions
    - Activities
    - Substates - may sequential or concurrent
    - Deferred events (infrequently used)
UML Behavior : State Machine--Concept

- **Advanced States and Transitions**
  - **Entry action** - Upon each entry to a state, a specified action is automatically executed.
    - Syntax (to be placed inside the state symbol): `entry / action`
  - **Exit action** - Just prior to leaving a state, a specified action is automatically executed.
    - Syntax (to be placed inside the state symbol): `exit / action`
UML Behavior: State Machine--Illustration

- **state name**: EnteringPassword
- **entry and exit actions**:
  - entry / display password
  - exit / validate
  - alphaKeypress / echo *
- **internal transitions**:
  - help / display
  - do / get
- **internal activity**: 


Event:

- **Event** is a noteworthy occurrence like method call, signal reception, timeout, change of a value, a condition becoming true,...

- An event has no duration

- An event may have parameters
• **Transition**
  - A directed relationship between two states.
  - A flow of control through a statechart diagram.
  - Contains five parts
    » **Source state** - current state before transition fires.
    » **Event trigger** - external stimulus that has the potential to cause a transition to fire.
    » **Guard condition** - a condition that must be satisfied before a transition can fire.
    » **Action** - an executable atomic computation.
    » **Target state** - new state after transition fires.
UML Behavior: State Machine--Concept

Figure 21-3: Transitions
UML Behavior : State Machine--Concept

- **Advanced States and Transitions**
  - **Internal Transitions** - The handling of an event without leaving the current state.
    - Used to avoid a states entry and exit actions.
    - Syntax (to be placed inside the state symbol):
      `event / action`
  - **Activities** - Ongoing work that an object performs while in a particular state. The work automatically terminates when the state is exited.
    - Syntax (to be placed inside the state symbol):
      `do / activity`
    - Activities are ongoing operations; actions are instantaneous operations.
UML Behavior: State Machine--Concept

- Advanced States and Transitions
  
  **Deferred Event** - An event whose occurrence is responded to at a later time.

  » Syntax (to be placed inside the state symbol): `event / defer`

  » Once an event has been deferred it remains deferred until a state is entered where that particular type of event is not listed as deferred.

  » The state diagram then responds to the event as if it had just occurred.
UML Behavior: State Machine--Concept

- **Advanced States and Transitions**
  - **Simple state** - A state that contains no *substates*.
  - **Composite state** - A state that contains *substates*.
  - **Substate** - A state that is nested inside another state.
    - Substates allow state diagrams to show different levels of abstraction.
    - Substates may be sequential or concurrent.
    - Substates may be nested to any level.
UML Behavior : State Machine--Concept

• Advanced States and Transitions
  - **Sequential Substates** - The most common type of substate. Essentially, a state diagram within a single state.
    » The “containing” state becomes an abstract state.
    » The use of substates simplifies state diagrams by reducing the number of transition lines.
    » A nested sequential state diagram may have at most one initial state and one final state.
UML Behavior: State Machine--Illustration

Doors

opened ↔ closed

open() → close()

lock() → unlock()

locked

Run: opened close closed; ...
UML Behavior: State Machine—Illustration

Initialization

- `addStudent/set #enrolled=0`
- `^CourseOOIS.Create`

Open

- `addStudent [#enrolled<5]`
- `semesterStarted`
- `cancel`
- `[#enrolled>5]`

Canceled

Closed

- `^CourseOOIS.Delete`
UML Behavior: State Machine---Illustration

Change event

InCredit

- deposit(m) / balance := balance + m
- withdraw(n) / balance := balance - n; return balance

[balance >= 0]

Overdrawn

- deposit(m) / balance := balance + m
- withdraw(n) / balance := balance - n; return balance
- when(balance < overdraftLimit) / notifyManager

[balance < 0]

Time event

Frozen

balance < Limit during(3 months)
Parallel States and Joins
UML Behavior: State Machine--Illustration

Result of Taking A Class

- Incomplete
  - Lab1: lab 1 done
  - Lab2: lab 2 done
  - Term project: project done
  - Take exam: pass
  - fail

- Passed

- Failed

- Lab1: lab 1 done
- Lab2: lab 2 done
- Term project: project done
Concurrent State Diagrams
UML Behavior: State Machine--Illustration

**VCR**
- Off
- On

**Remote Control**
- Controlling TV
  - "VCR"
- Controlling VCR
  - "power" button
  - "power" button

**Television**
- Off
- On

- "power" button
- "TV"
- ^VCR.toggle power
- ^Television.toggle power

- toggle power
- toggle power
Parallel States and Fork
UML Behavior: State Machine—Illustration

Example state
Concurrent state 1

A1 \rightarrow A2

Concurrent state 2

B1 \rightarrow B2

setup \rightarrow cleanup

multiple target states

multiple source states

synchronization bar (fork)

synchronization bar (join)
Structured State Charts

UML Behavior: State Machine--Illustration

Order

Fill Order

[librarian not available]

[librarian available]/give order

Check Order

stock available[ship date in future]

[Order not OK]

Reject

[Order OK]

Bring Book

Filled
Terms and Concepts

Figure 21-5: Sequential Substates
State Diagram for a Phone Line

Figure 5.3 Event trace for phone call

Figure 5.5 State diagram for phone line
State Diagrams ATM, Bank, and Consortium

Figure 8.20 State diagram for class ATM

Figure 8.21 State diagram for class Consortium

Figure 8.22 State diagram for class Bank