# 02291: System Integration

Hubert Baumeister
hub@imm.dtu.dk

Spring 2012

## Contents



## 1 General Information

**System Integration**

- Type
  - 5 ECTS Points
  - Audience: Masters / advanced Bachelor students
  - Lectures with exercises
  - Exam: modelling project with project presentation
- Time and Location
  - Lecture Wednesdays 8:30-10:15 aud 13 building 308
  - Exercise session after the lecture (rooms 97, 96, 98 building 306)
- How to reach me
  - hub@imm.dtu.dk; office 322/010
- Teaching assistant: Piotr Jacek Puczynski (s091137@student.dtu.dk)
- Web Page
  - `http://www.imm.dtu.dk/courses/02291`

**Motivation**

**Outsourcing Software Development**

- To be able to communicate the *requirements*, *architecture*, and *design* of a software system using models so that programmers can implement software according to that model
- *Validate* the design and *test* the delivered program
- *Understand* the connection between the different types of diagrams: e.g. use cases, class diagrams, components and state machines describing the life cycle of objects.

**Goal:**

- Know about
  1. (Object-oriented) modelling languages: mainly UML
  2. Why model?
  3. How to model
- To be able to
  - Model larger systems
  - Use models for *understanding*, *designing*, *communication*, *analysing*, *verification*, and *creating* Software
  - Document models
  - Analyse models
  - Validate models

**Course structure**

- Reading assingment to be done before the lecture
- By Monday: Pre-flight tests: tests that you have done your reading assingment
- Tuesday: Lecture with exercises
- Exercises:
  - 3 exercises following the structure of the exam project

**Exam Project**

- Project
  - Task is to model a larger system (*requirements*, *design*, *validation/verification*) so that programmers can implement the design
  - Duration 4 weeks at the end of the lecture
  - Written report
  - Groups: 4—6 people
- Project presentation
  - About the project
  - About the lecture
  - Date sometime in the examination period

# 2 Overview

**Model (definition)**
`http://en.wiktionary.org/wiki/model`

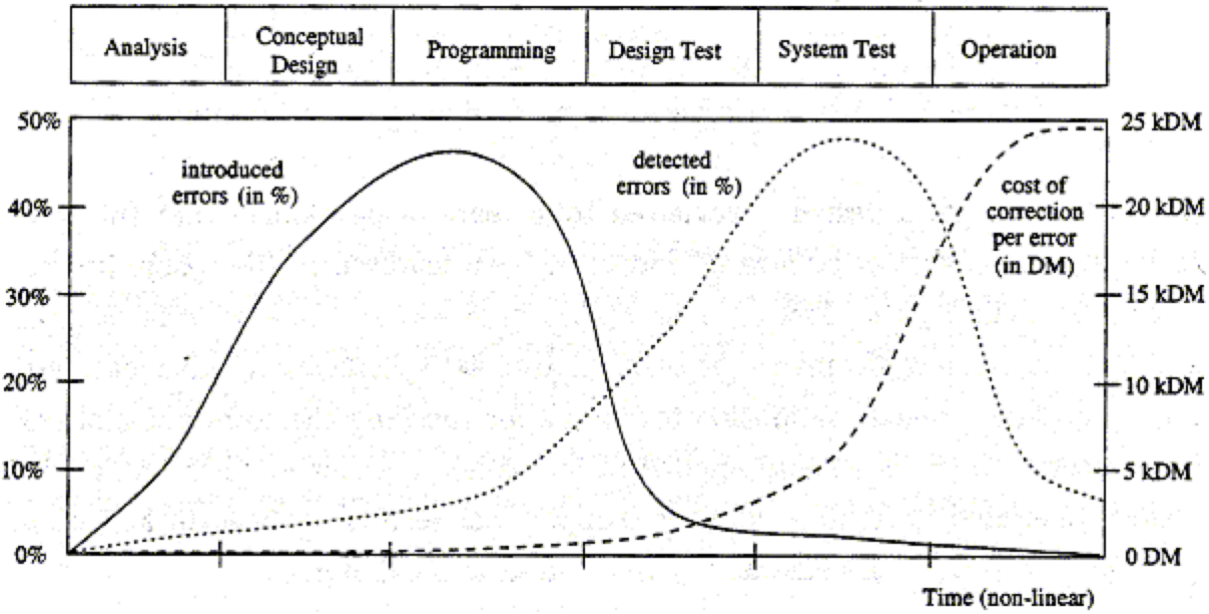
...

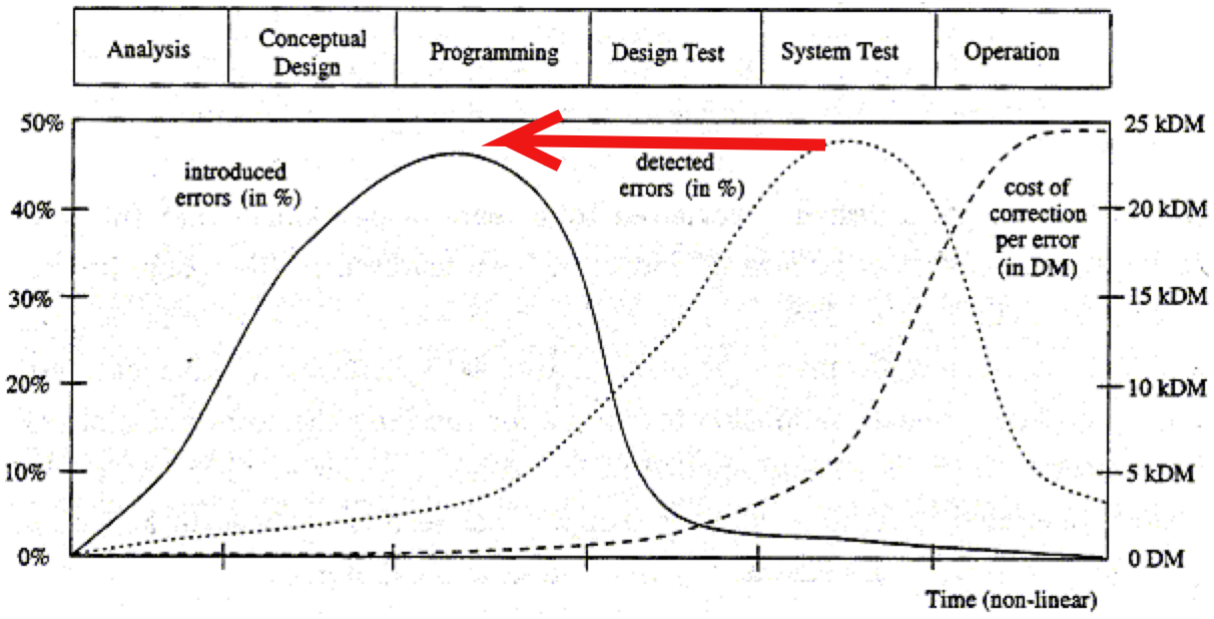
3. A *simplified* representation (usually mathematical) used to explain the workings of a real world system or event.
   - "The computer weather model did not correctly predict the path of the hurricane."

- A plane in a windtunnel

5. The *structural design* of a complex system.
   - ”The team developed a sound business model”

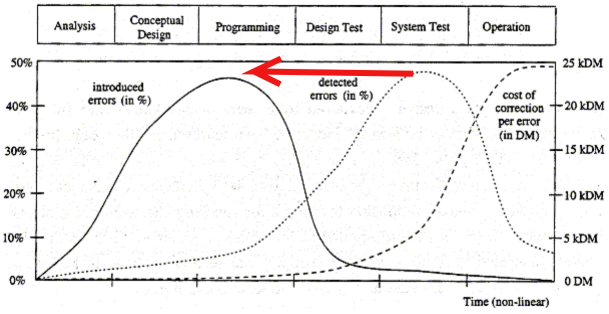
**Why model?**






- Liggesmeyer 1998

**Why model?**




- Raise the abstraction level of problem and solution description

  → *model* the problem and the solution

  → Feedback

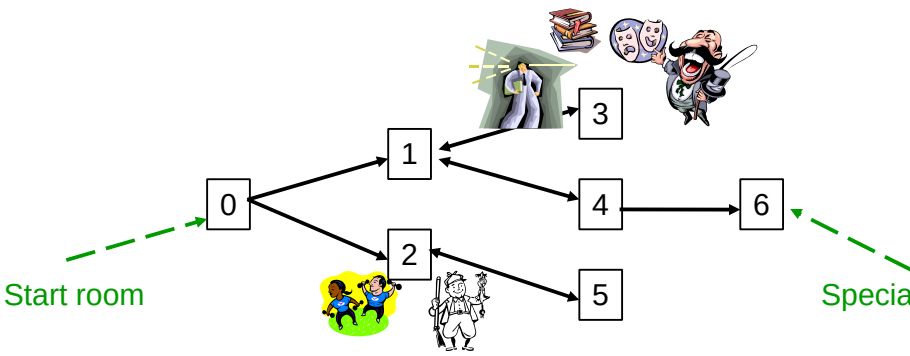
    * Model inspection
    * Automatic code generation
    * Analysis of models

**Characteristics of Models for Software Developement**

- Are written in an *artifical language* and express
  - information
  - knowledge
  - systems
- Application areas
  - computer science

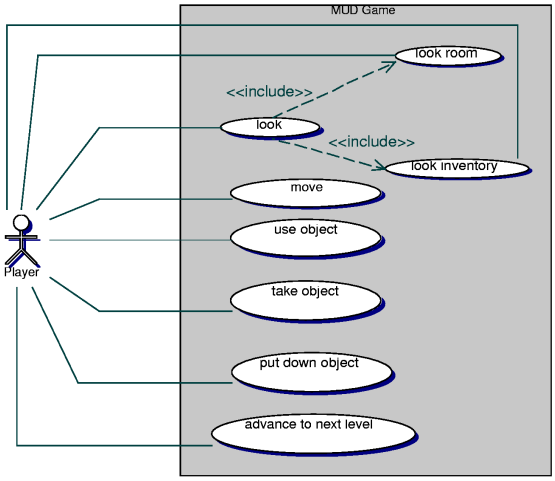
  - information management
  - business process modelling
  - software engineering
  - systems engineering
- Type of models
  - graphical
  - textual
  - both

**Example MUD**




A Multi User Dungeon (MUD) game

**Example of a graphical model**

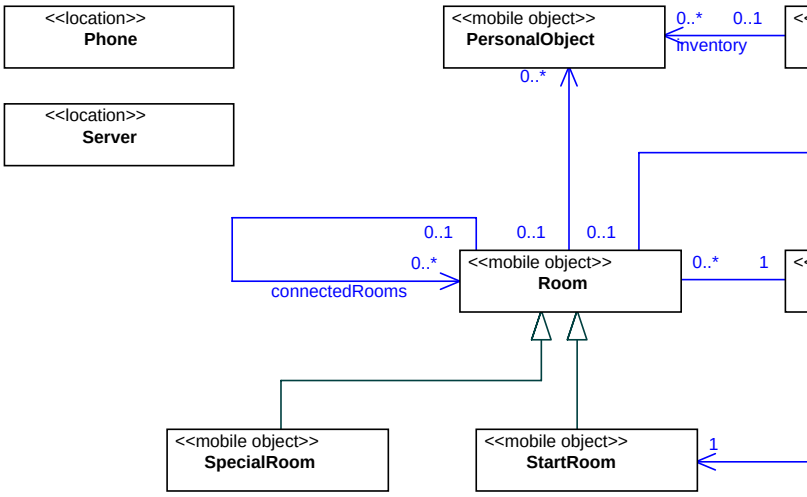



Use Case Diagram for the MUD game

**Models II**

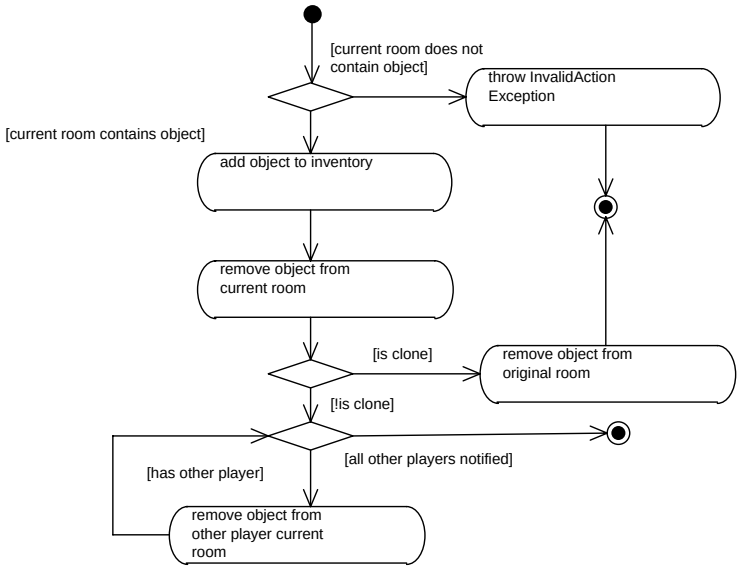
- Models specify for stakeholders
  - system requirements
  - structures
  - behaviours
- Stakeholders are (among others)
  - customers
  - operators
  - analysts
  - designers

**Example of a structural diagram**




Excerpt of the class diagram for the MUD game

### Example of a behavioural diagram




Take object activity in the MUD game

### Models III

- The use of models can be
  - informal

    - communication
    - → problem of ambiguity
    - drawings on a black board
  - formal
    - verification / validation
    - simulation
    - code generation

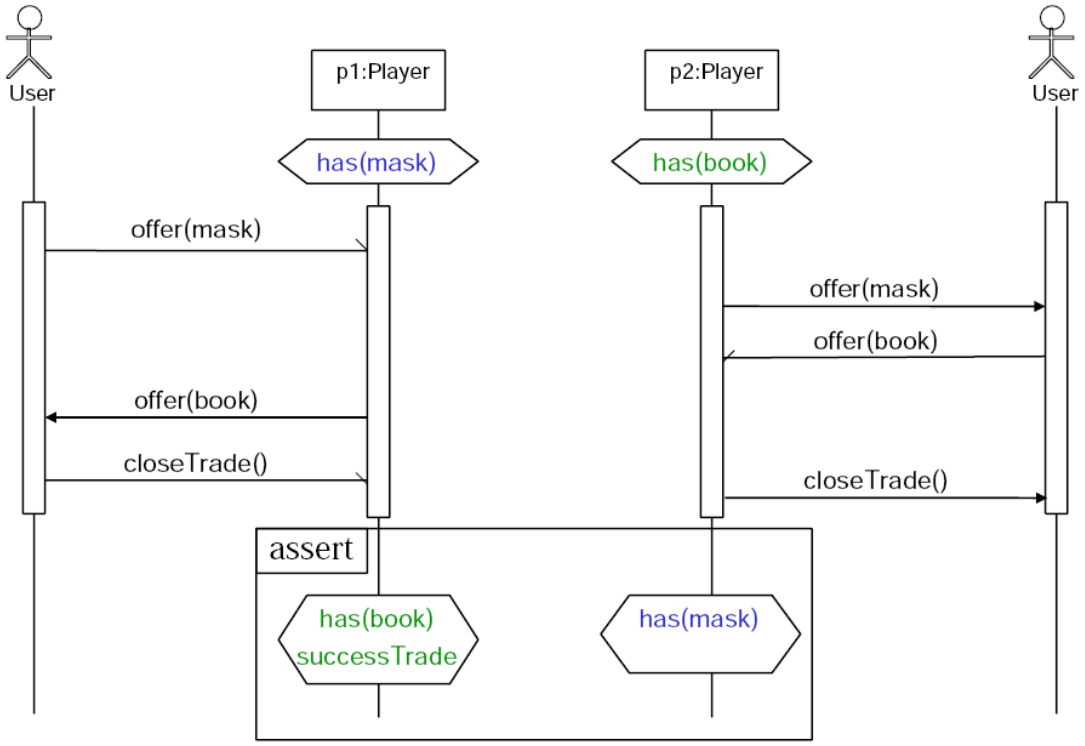
**Example for an informal model**

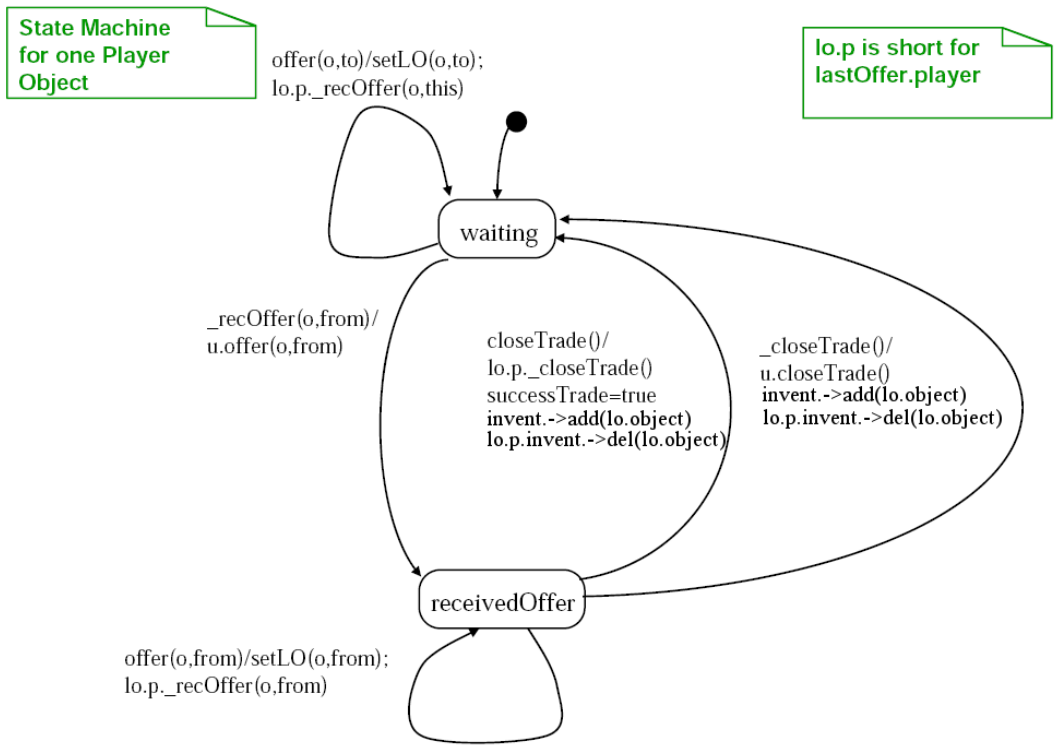
Class Responsibility Collaboration (CRC) cards for the HotDraw editor `http://www2.imm.dtu.dk/courses/02291/examples/draw.html`

**Example verification**

Checking that trading in the MUD game works

- No deadlock (the trading protocol always finishes)
- Player agree on the outcome of a trade




Homomorphic abstraction of the state space

Model-checking:

- No Deadlock, but ...

**Property does not h**
**Players don't agree**
**trade**

**Example validation**

Fit tests of a Multi User Dungeon (MUD) game

At the beginning, the player is in the start room of the first level and he can move to rooms 1 and 2.

| BuildGameFixture | | |
|---|---|---|
| name | health | createPlayer() |
| p0 | 5 | |

| fit.ActionFixture | | |
|---|---|---|
| start | GameFixture | |
| check | look | 1,2 |

Now we leave the start room. The player cannot see the start room.

| fit.ActionFixture | | |
|---|---|---|
| action | move | 1 |
| check | current room | 1 |
| check | look | 3,4 |

Let's move to another room. Now the player can sse the room he came from because it is not the start room.

| fit.ActionFixture | | |
|---|---|---|
| action | move | 4 |
| check | look | 1,3,5,6 |

Search

Edit

Test

At the beginning, the player is in the start room of the first level and he can move to rooms 1 and 2.

| BuildGameFixture | | |
|---|---|---|
| name | health | createPlayer() |
| p0 | 5 | p0 |

| fit.ActionFixture | | |
|---|---|---|
| start | GameFixture | |
| check | look | 1,2 |

Now we leave the start room. The player cannot see the start room.

| fit.ActionFixture | | |
|---|---|---|
| action | move | 1 |
| check | current room | 1 |
| check | look | 3,4 |

Let's move to another room. Now the player can sse the room he came from because it is not the start room.

| fit.ActionFixture | | |
|---|---|---|
| action | move | 4 |
| check | look | 1,3,5,6 |

Search

Edit

---

**How to model? I**

- Domain Driven Design
  - Use domain models to drive the program design
  - Use abstractions from the domain in your program
- CRC cards
  - Informal and simple (paper based)
  - Domain model or design model
  - Connects object structure with object behaviour

**How to model? II**

- Model Driven Architecture
  - OMG (`http://www.omg.org/mda`)
  - Alternative names: Model Driven Development, Model Driven Design ...
  - Abstract but (UML) models are transformed to more concrete (executable UML) models and finally to code
- Explorative Modelling
  - Observation: A lot of UML models use only that part of UML that is available in an object-oriented programming language
  - Use an (object-oriented) programming language for modelling
  - The model is executable (but still a model and not the final program)
  - This model can then be further refined and refactor to yield the resulting program

**How to model? III**

- Agile Modelling
  - Values
    - Communication
    - Simplicity
    - Feedback
    - ...
  - Principles
    - Software is Your Primary Goal
    - Assume Simplicity
    - Incremental Change
    - ...
  - Practices
    - Create Several Models in Parallel
    - Iterate to Another Artifact
    - ...

# 3 Introduction to UML

**What is the UML?**

- Family of *graphical notations* (e.g. class diagram, interaction diagram, ...) for describing (object-oriented) software
- Backed by a single *metamodel* (provides a sort of *semantics* for the diagrams)
- Idea
  - "A picture is more than a thousand words"
  - Describe software on a higher abstraction level than programming languages
- UML is an open standard managed by the Object Management Group (`www.omg.org`)
- Web resources
  - `http://www.uml.org`
  - UML Standard Document (Superstructure)
    - `http://www2.imm.dtu.dk/courses/02291/files/UML2.1.1_superstructure.pdf`

**Ways of using the UML (I)**

- Sketch
  - Informal use
  - Used to discuss ideas
  - Used to understand the domain
  - Used to sketch software designs

  → In most cases incomplete

  → Use of whiteboards, simple graphic editors

- Blueprint
  - Between formal and informal use
  - Allows to construct software from that model (forward engineering)
  - Describes existing software (reverse engineering)
  - Completeness
  - Should convey all design decisions
  - Interfaces to subsystems
  - Use of CASE tools

**Ways of using the UML (II)**

- Programming language
  - Model-Driven Architecture
  - Executable UML
  - UML becomes the source code
  - Describing structure is well understood (in principle). But
    - How to describe the behaviour?
    - How well are activity diagrams, state machines and interactions diagrams suited for (object-oriented) programming?
  - Productiveness

**UML's history (I)**

- 1980s
  - Objects became main stream (e.g. C++ / Simula / Smalltalk)
  - Several different OO methods and graphical notations
    - Grady Booch, Peter Coad, Ivar Jacobson, Jim Odell, Jim Rumbaugh, Sally Shlaer and Steve Mellor, Rebecca Wirfs-Brock, ...
  - Problem
    - Different notations for the same concepts
    - Sometimes only slightly different!
- 1990s
  - Grady Booch, Ivar Jacobson and Jim Rumbaugh are all with Rational
  - First version of the UML (UML 0.8)
  - UML 0.8 unifies the notations of Booch, Jacobson and Rumbaugh
    - But UML does *not* unify their methodologies

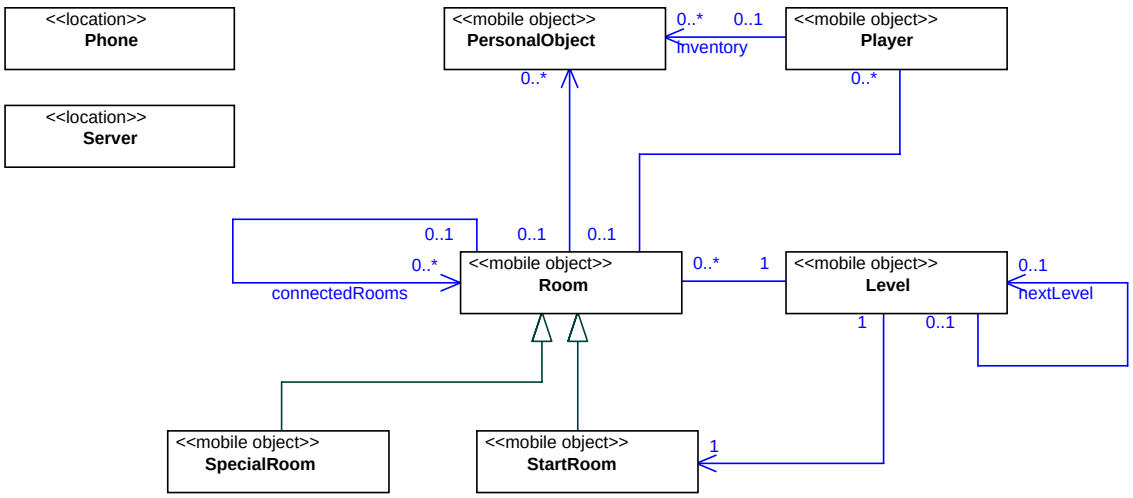
**UML's history (II)**

- 1996
  - UML is handed over to a standard organisation (The Object Management Group (OMG))
  - Driven by tool vendors!
  - Tool interoperability!
  - UML should not be owned by Rational alone
- UML revisions
  - 1.x with 1.4 and 1.5 being widely used
  - 2.x (UML 2.0 is ISO standard now)

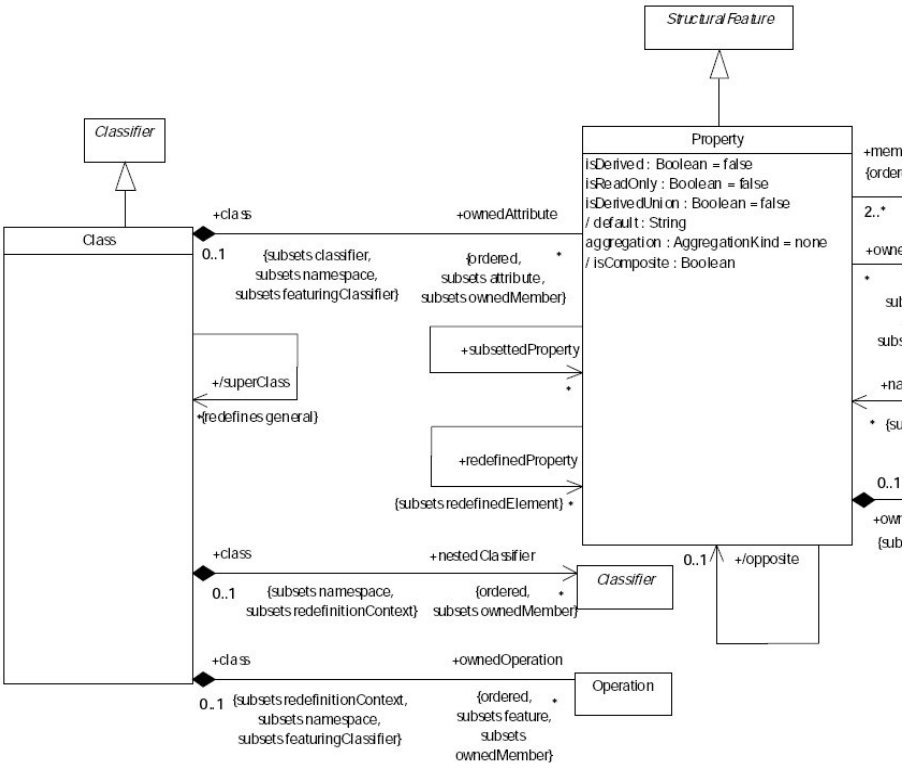
### Notations and Metamodels

- UML consists of
  - a set of graphical notations
  - and a single metamodel

### Example of a class diagram




### Metamodel example




**Metamodel**

- Describes the *abstract syntax* of UML diagrams
- Describes the *concepts* of UML elements
  - e.g. the concept of a class or association or feature ...
- concepts are mapped to notations
  - e.g. the concept of a class is mapped to rectangle with subcompartments
- The metamodel itself is described using a *graphical language* called MOF (Meta-Object Facility)
  - Which is based on *UML class diagrams*
  - Which in turn is *defined by its own metamodel*!!

  → MOF can be used to defined *ones own* modelling language

  → Courses Advanced topics in SE (02265) and SE II (02162)
- The metamodel is important for model exchange
  - Using XMI, an exchanged format based on the abstract syntax defined in the metamodel

**The meaning of UML**

- The abstract syntax / concepts are well defined
  - By using a metamodel
- The meaning of the concepts, however, is not well defined
  - What does a class mean?

  - How are activity diagrams executed?
  - How are state machines executed?
  - What is the meaning of interaction diagrams?
- Semantic variation points
  - Sometimes different interpretations are intended
  - Tool vendors can choose how they interpret the concepts

**UML is not enough**

- UML is a general purpose modelling language
  - contains a lot of diagram types
  - but specific domains have more specific or other diagram types
- UML can be extended
  - lightweight
    * without changing the metamodel
    * so called UML profile
  - heavy weight
    * by changing the metamodel

→ *Use the modelling language best suited for your purpose*

**Development Process**

- UML is a notation for models
  - Mainly software
  - But also
    * Business processes
    * Systems (→ System Modelling Language (SysML))
    * Knowledge
    * ...
- UML is *not* a software development process!!
- UML can be used in the context of almost any process
  - Waterfall / Iterative
  - (Rational) Unified Process
  - Agile Software Development (e.g Extreme Programming, Scrum, FDD ...)
  - ...

**UML Diagrams (I)**

- Structure Diagram
  - *Class Diagram*
    - Class, features, and relationships
  - *Object Diagram*
    - Example configuration of instances
  - *Package Diagram*
    - Hierarchical structure for models
  - *Component Diagram*
    - Structure and connections of components
  - Deployment Diagram
    - Deployment of artifacts to nodes
  - Composite Structure Diagram
    - Runtime decomposition of a class

**UML Diagrams (II)**

- Behaviour Diagram
  - *Use-Case Diagram*
    - How users interact with a system
  - *Activity Diagram*
    - Procedural and parallel behaviour
  - *State Machine Diagram*
    - How events change an object over its life
  - Interaction Diagram
    - *Sequence Diagram*: Interaction between objects; emphasis on sequence
    - *Communication Diagram* (formerly called *collaboration diagram*): Interaction between objects; emphasis on links
    - Interaction Overview Diagram: Mix of sequence and activity diagram
    - Timing Diagram: Interaction between objects; emphasis on timing

# 4 Summary

**Course Contents**

- UML
  - Class-, Component-, Object- . . . diagrams
  - Activity-, Interaction- . . . diagrams
  - Object Constraint Language
- How to model
  - Agile modelling
  - CRC cards
  - Explorative Modelling
  - Domain Driven Design
  - Model Driven Design
- Validation / Verification
  - Acceptance tests
  - (UML) modelchecking

**Final project**

- Goal
  - Model a larger system so that it could be given to a programmer who implements it
- Steps
  - Model the requirements
    - Domain Model
    - Use Cases
    - Create a test model
    - ...
  - Build a design model
    - Component model
    - Class diagrams
    - OCL constraints
    - ...
  - Verify and validate the design
    - Use interaction diagrams to show that the use cases are realizable
    - ...
  - Reflect on the methods you use

**Exercises and Final Project**

- Exercises
  - Week 2 — Week 4: Requirements model and Acceptance Tests
  - Week 4 — Week 7: Design model
  - Week 7 — Week 8: Use case validation
  - Exercises are *not* mandatory but are *useful* to get feedback
  - *Form groups (4–6) already now: this increases the chances to get feedback on your exercise report*
- Final Project
  - Week 9 — Week 13 (4 weeks)
  - Teams: 4 — 6 people

**Literature: Some general remakrs**

- There is no book covering *all* the topics introduced in this course.
- This is a masters course
  - You need to be able to get an overview over the literature and then decide what parts to skip, read only lightly, and which parts to read in depth
  - You need to be able to know where you can find further information when you need them.
  - E.g. looking into standards is a good way to resolve technology questions (sometimes :-)

**Literature: UML**

- Most books on UML will do. Make sure it deals with UML 2.x. Some options
- Grady Booch, James Rumbaugh, and Ivar Jacobson *The Unified Modeling Language User Guide (2nd Edition)* (Users manual from the creators of UML)
- Martin Fowler *UML Distilled. Third Edition* (Contains a *distilled* version of the most important UML diagrams)
- Perdita Stevens *Using UML* (A text book for teaching OO and UML)

**Literature: OCL, Modeling**

- OCL
  - Jos Warmer and Anneke Kleppe *The Object Copnstraint Language. Second Edition*
- Modeling
  - Scott Ambler *Agile Modeling*
  - Eric Evans *Domain-Driven Design*
  - Nancy Wiklinson *Using CRC Cards*
  - Anneke Kleppe, Jos Warmer, Wim Bast *MDA Explained*
- Further reading and Web resources will be given when coming to the relevant topics

**Tools**

- Two classes: simple *drawing tools* and meta model based *modelling tools*
- Drawing tools
  - UMLet (Eclipse plugin `www.umlet.com`)
  - Violet UML (Eclipse plugin `http://alexdp.free.fr/violetumleditor/page.php`)
  - Visio (Windows only; available though DTU agreement with Microsoft)
- Modeling tools
  - Topcased
  - MagicDraw
  - Rational Modeller (IBM)
  - Eclipse UML (Omondo `www.omondo.com`)
  - eUML (Soyatec `www.soyatec.com`) (Only class and sequence diagrams)
- Tool to be used in the course: *Topcased*