02291: System Integration

Week 1

Hubert Baumeister
huba@dtu.dk

DTU Compute
Technical University of Denmark

Spring 2018

Contents

Overview

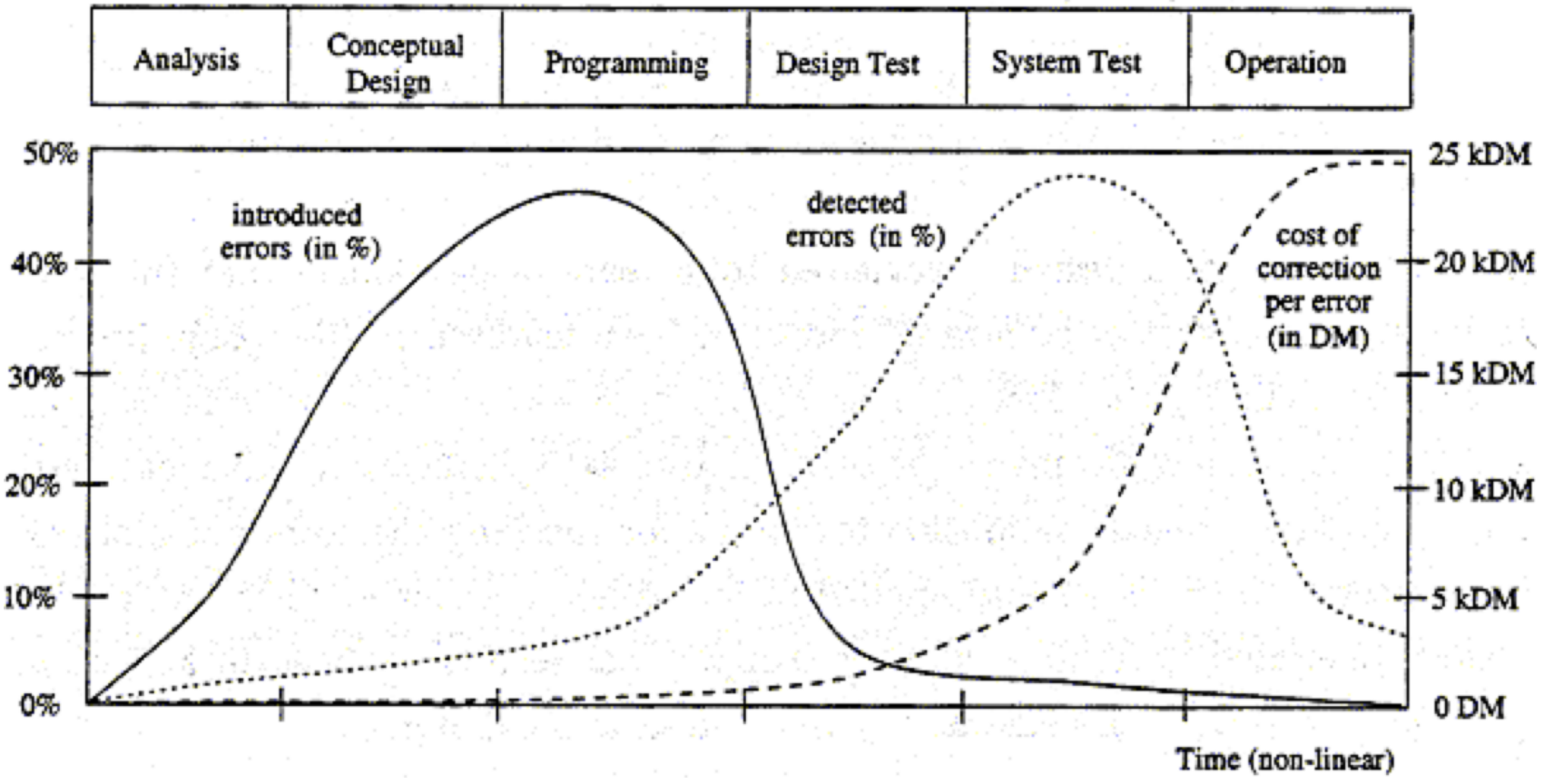
Introduction to UML

Practical Informations

System Integration

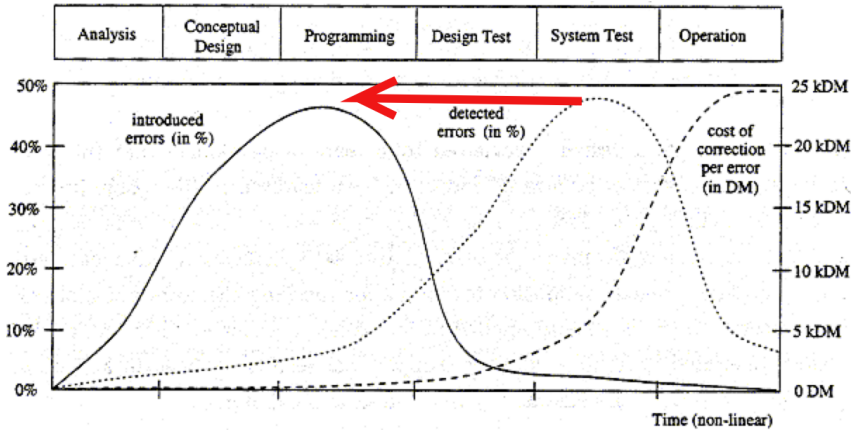
- Type
  - 5 ECTS Points
  - Audience: Masters / advanced Bachelor students
  - Lectures with exercises
  - Exam: modelling project with project presentation
- Time and Location
  - Lecture Wednesdays 10:15–12:00
  - Exercise session before the lecture 8:15–10:00
- How to reach me
  - huba@dtu.dk; office 303A/058
  - Teaching assistants: Constantina Ioannou and Wenhao Li
- Web Page
  - `http://www2.compute.dtu.dk/courses/02291`
  - Piazza

Problem in Software Engineering



- Liggesmeyer 1998

Why model?



- Abstraction is the key
  - → *Models* for the problem and the solution
  - → *Feedback* early
    - Model inspection
    - Automatic code generation
    - Analysis of models

Model (definition)

`http://en.wiktionary.org/wiki/model`

...

A simplified representation 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

Clay model of a car

→ Abstraction

- Focus on some aspects only, e.g. air resistance
- Disregard the other aspects
- Aggregate details by introducing new concepts

Learning objectives:

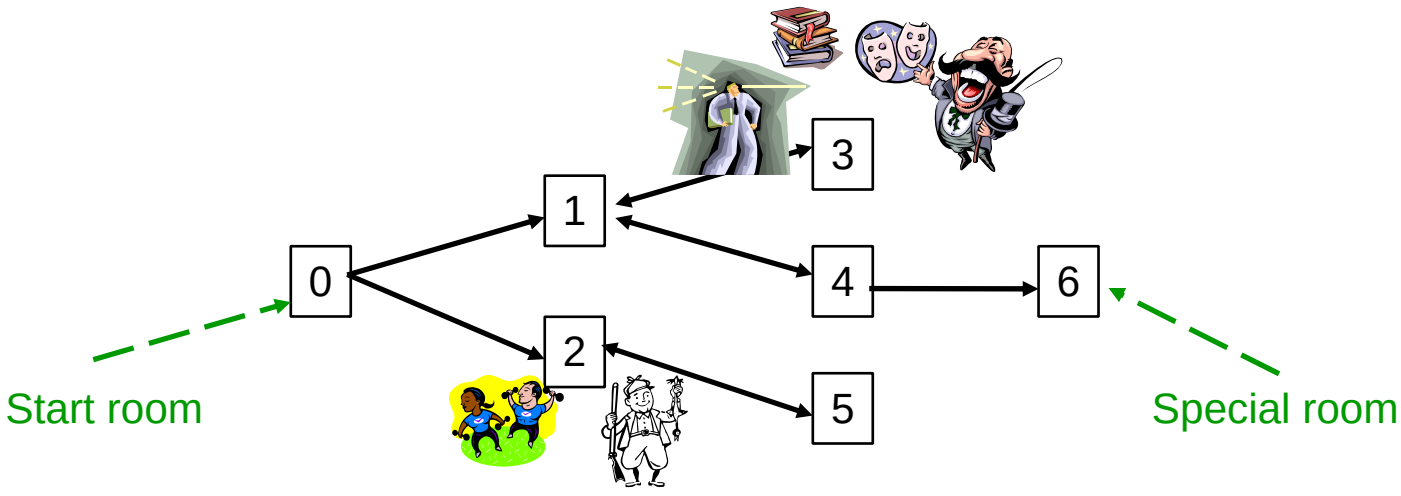
- Know about
  1. (Object-oriented) modelling languages: mainly UML
  2. Why model?
  3. How to model
  4. How to validate a model
- To be able to
  - *Model* larger systems
  - Use models for *understanding*, *designing*, *communication*, *analysing*, *verification*, and *creating* software
  - *Document*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

Why model?

- Conjecture:
  - it suffices to communicate only key abstractions rather than the whole programm
  - it is easier to create a model of the system than to program it
  - a model of the system is easier to understand than the program itself

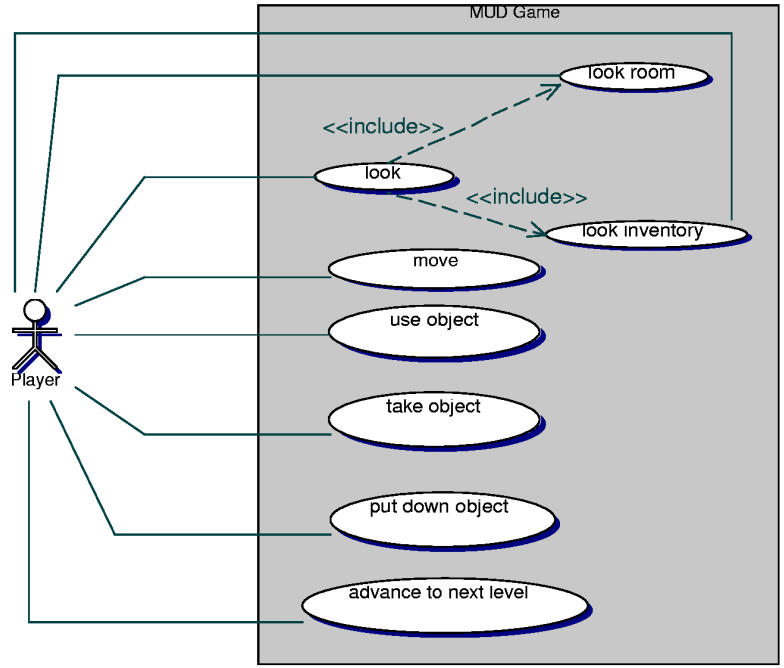
Example MUD

A Multi User Dungeon (MUD) game

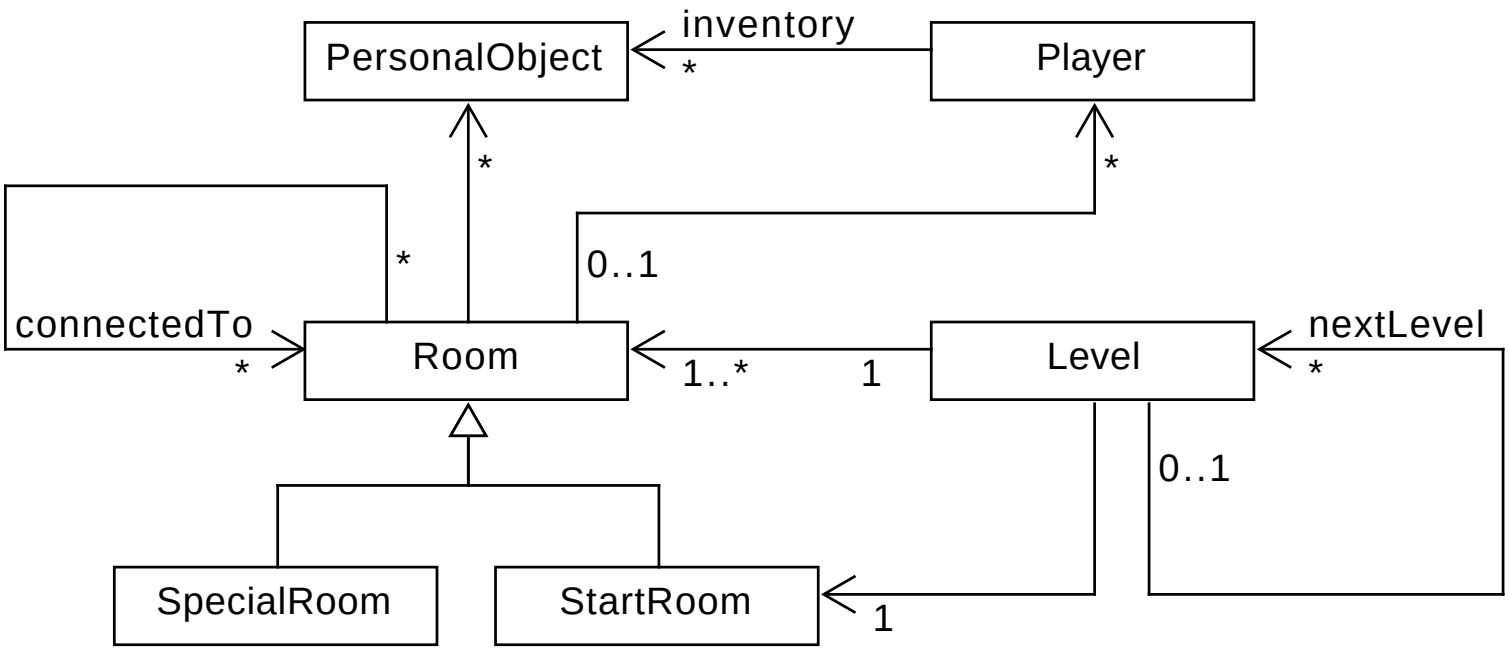


Abstraction: Focus on Functionality

Use Case Diagram for the MUD game

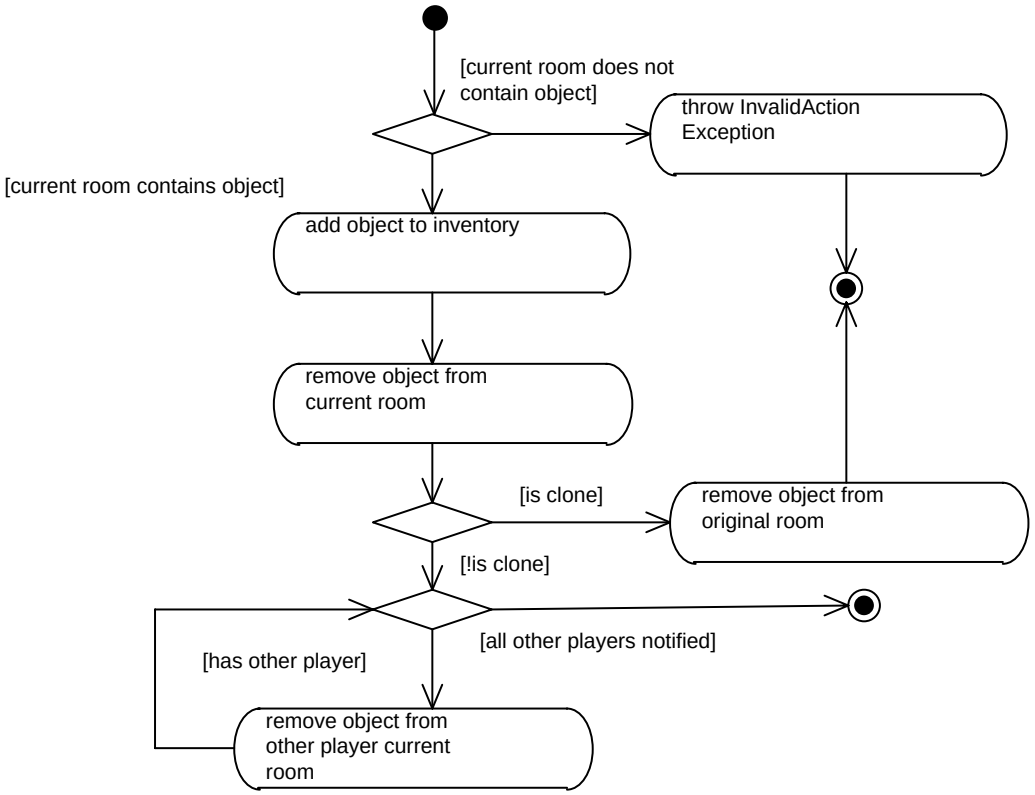


Abstraction: Focus on Structure



Abstraction: Focus on the Behaviour

Take object activity in the MUD game



Use of models

- informal
  - communication
  - $\rightarrow$ 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.compute.dtu.dk/courses/02291/examples/draw.html`

How to model?

- Domain Driven Design
- CRC cards
- Model Driven Architecture
- Explorative Modelling
- Agile Modelling

Contents

Overview

Introduction to UML

Practical Informations

What is the UML?

- Unified Modelling Language (UML)
- Family of *graphical notations* for describing aspects of (object-oriented) software
  - ”A picture is more than a thousand words”
- Based on a *metamodel*
- Not a development process
- UML is an open standard and an ISO standard
  - managed by the Object Management Group (`www.omg.org`)
- Web resources
  - `http://www.uml.org`

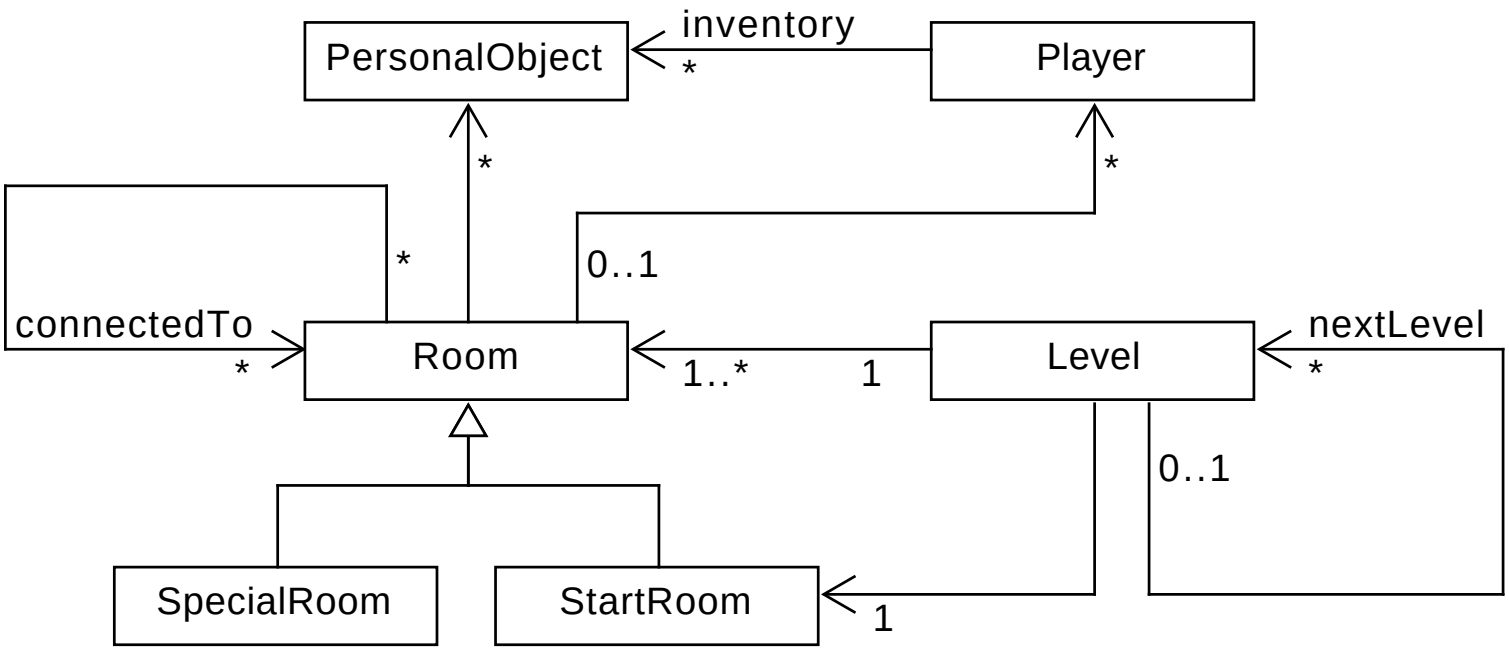
UML's history

- 1980s
  - Objects became main stream
  - OO methods and graphical notations
    - Grady Booch, Peter Coad, Ivar Jacobson, Jim Odell, Jim Rumbaugh, Sally Shlaer and Steve Mellor, Rebecca Wirfs-Brock, ...
- 1990s
  - Grady Booch, Ivar Jacobson and Jim Rumbaugh at Rational
  - First version of the UML (UML 0.8)
- 1996
  - The Object Management Group (OMG)
  - Driven by tool vendors!
    - Tool interoperability!
  - 1.x with 1.4 and 1.5 being widely used
- 2005
  - UML 2.0 ISO standard

Notations and Metamodels

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

Example of a class diagram



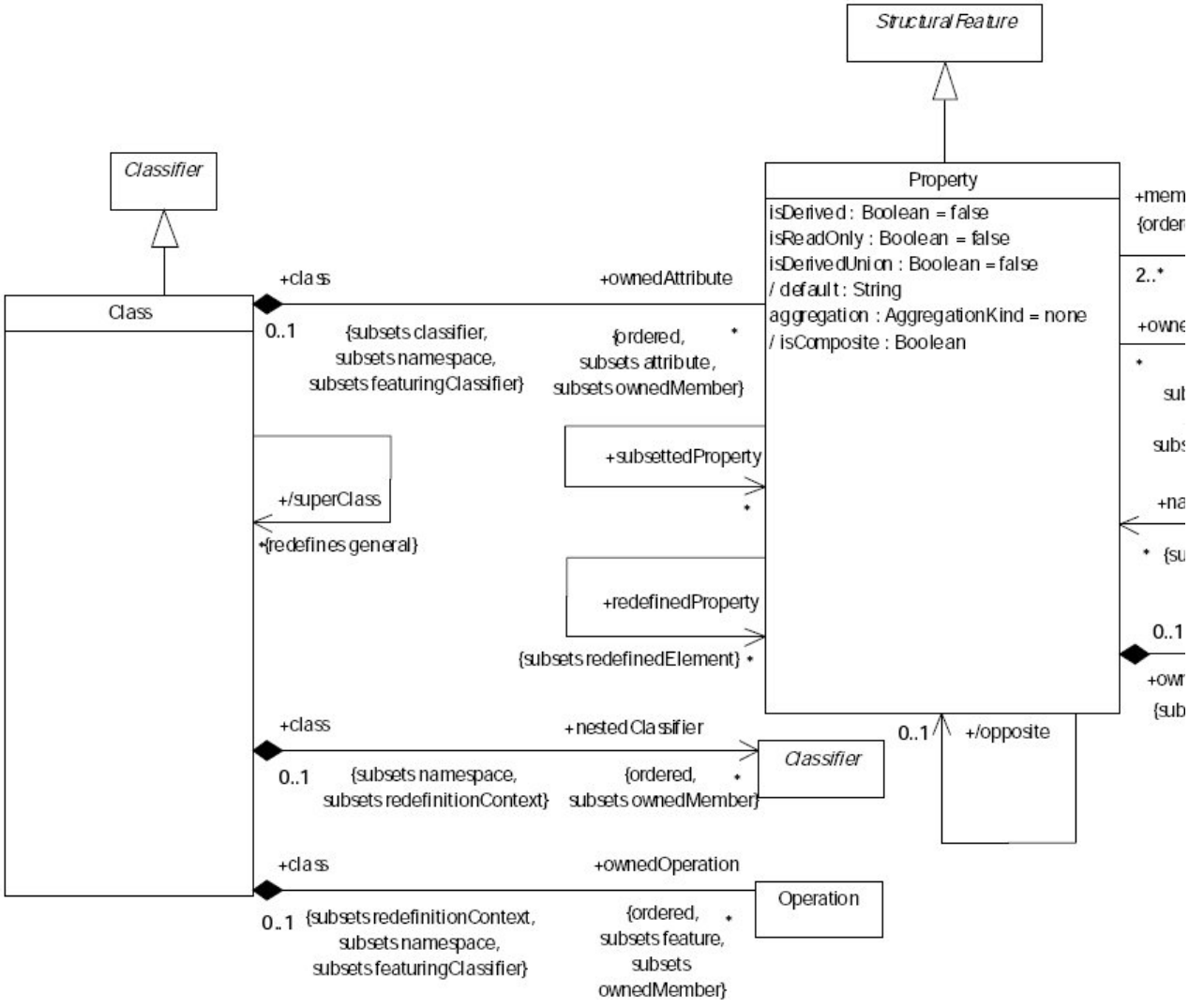
Metamodel

- Abstract syntax of UML diagrams
- Describes the *concepts* of UML elements
  - e.g. class, association, ...
- concepts are mapped to notations
  - e.g. the concept of a class is mapped to rectangle with subcompartments
- Metamodel written in MOF (Meta-Object Facility)
  - Basically *UML class diagrams*

  $\rightarrow$ MOF can be used to defined *ones own* modelling language

  $\rightarrow$ Course Software Engineering II (02162)
- The metamodel is important for model exchange
  - XMI (XML Metadata Interchange)

Metamodel Excerpt



The meaning of UML

- The abstract syntax / concepts: Ok
  - $\rightarrow$ metamodel
- The meaning of the concepts: not well-defined
  - Meaning of *class*?
  - Execution of *activity diagrams* and *state machines*
- Semantic variation points
  - Left open in the standard
  - Tool vendors choose meaning

Ways of using the UML

- Sketch
  - Informal use

  → Use of whiteboards, simple graphic editors

Ways of using the UML

- Sketch
  - Informal use
  - → Use of whiteboards, simple graphic editors
- Blueprint
  - Forward Engineering: e.g. create class stubs
  - → Use of CASE tools

Ways of using the UML

- Sketch
  - Informal use
  - → Use of whiteboards, simple graphic editors
- Blueprint
  - Forward Engineering: e.g. create class stubs
  - → Use of CASE tools
- Programming language
  - Executable UML
  - Problems
    - How to describe the behaviour?
  - → Use of CASE tools

UML is not enough

- UML is a general purpose modelling language
- UML can be extended
  - lightweight (UML profile)
  - heavy weight

$\rightarrow$ *Use the modelling language best suited for your purpose*

UML Diagrams (I)

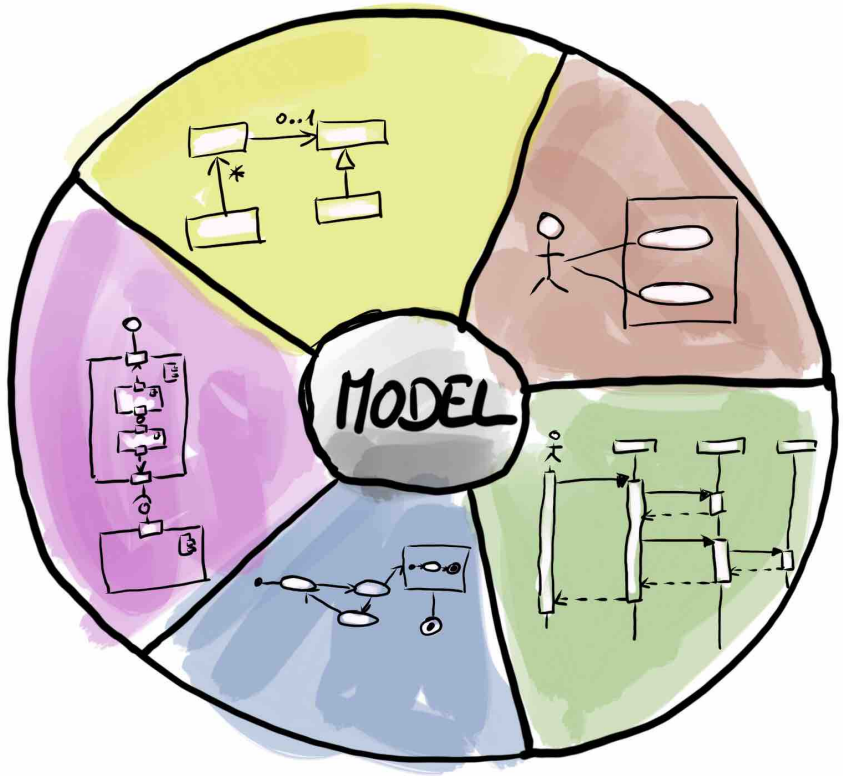
- Structure Diagrams
  - *Class Diagram*
    - Class, features, and relationships
  - *Object Diagram*
    - Example configuration of instances
  - *Package Diagram*
    - Hierarchical structure for models
  - *Component Diagram*
    - Structure and connections of components
  - Deployment Diagram
 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

UML Diagrams (II)

- Behaviour Diagrams
  - *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

Focus of the course

Many diagrams, but only *one model*

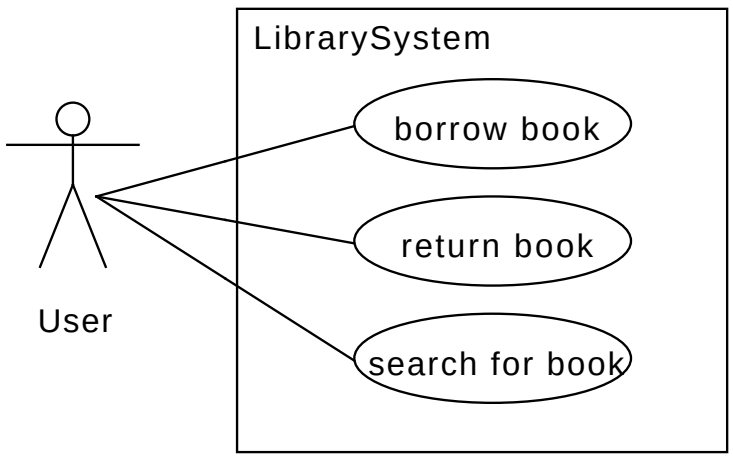


Different Views on the system

- Functionality: Use Case diagram, state machines, activity diagram, ...
- Structure: Component diagram, Class diagram
- Validation: Interaction diagram

Library Example: Detail of Use case *borrow book*

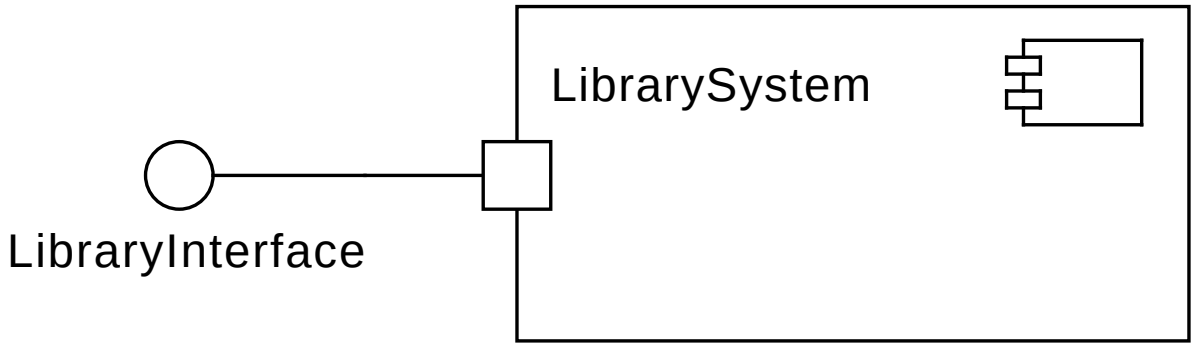
Use case diagram

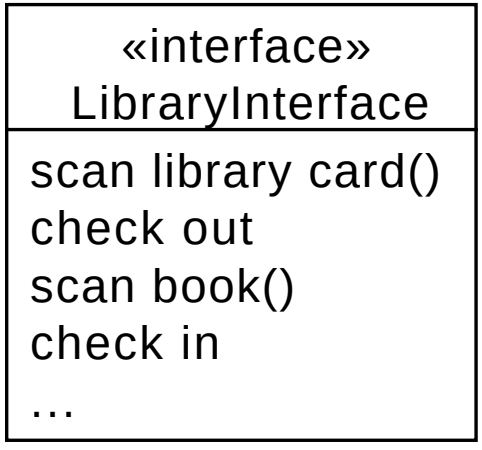


Use case *borrow book*
Basic course of events:

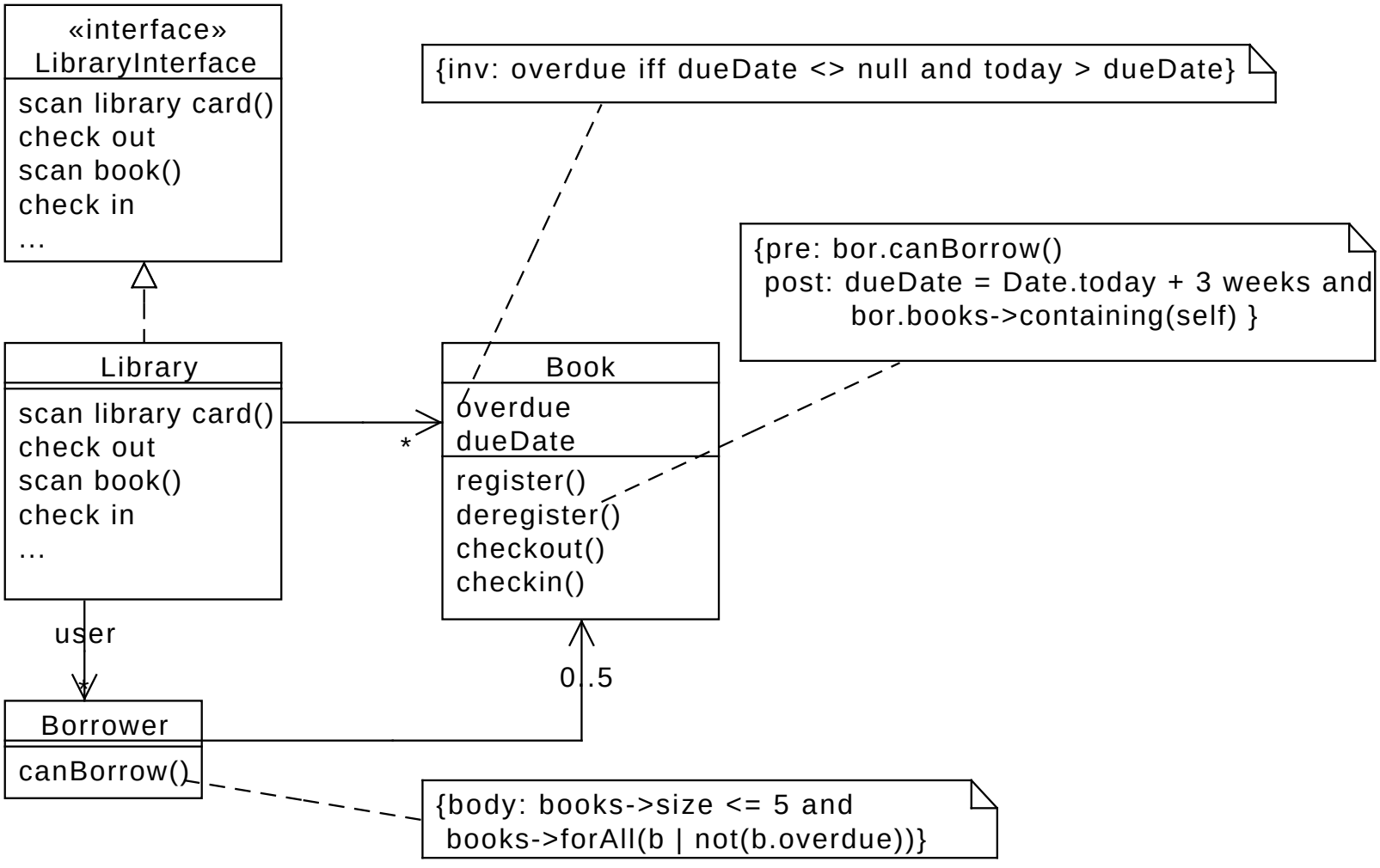
1. User scans his library card
2. User selects check out
3. User scans the book
4. System confirms loan

Implementation: Component Diagram

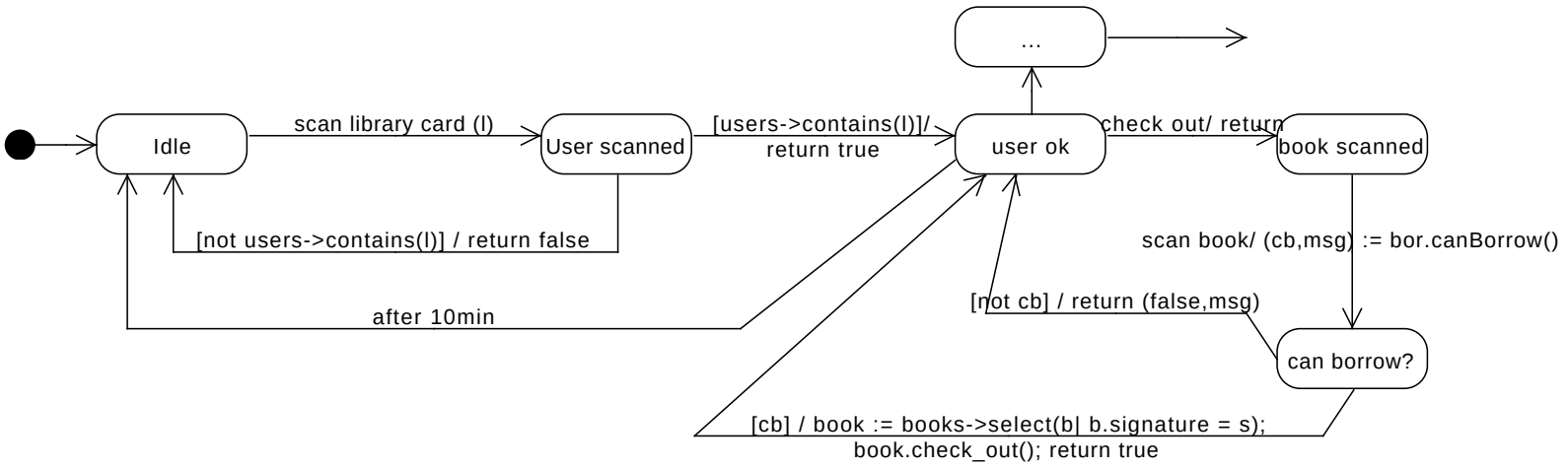




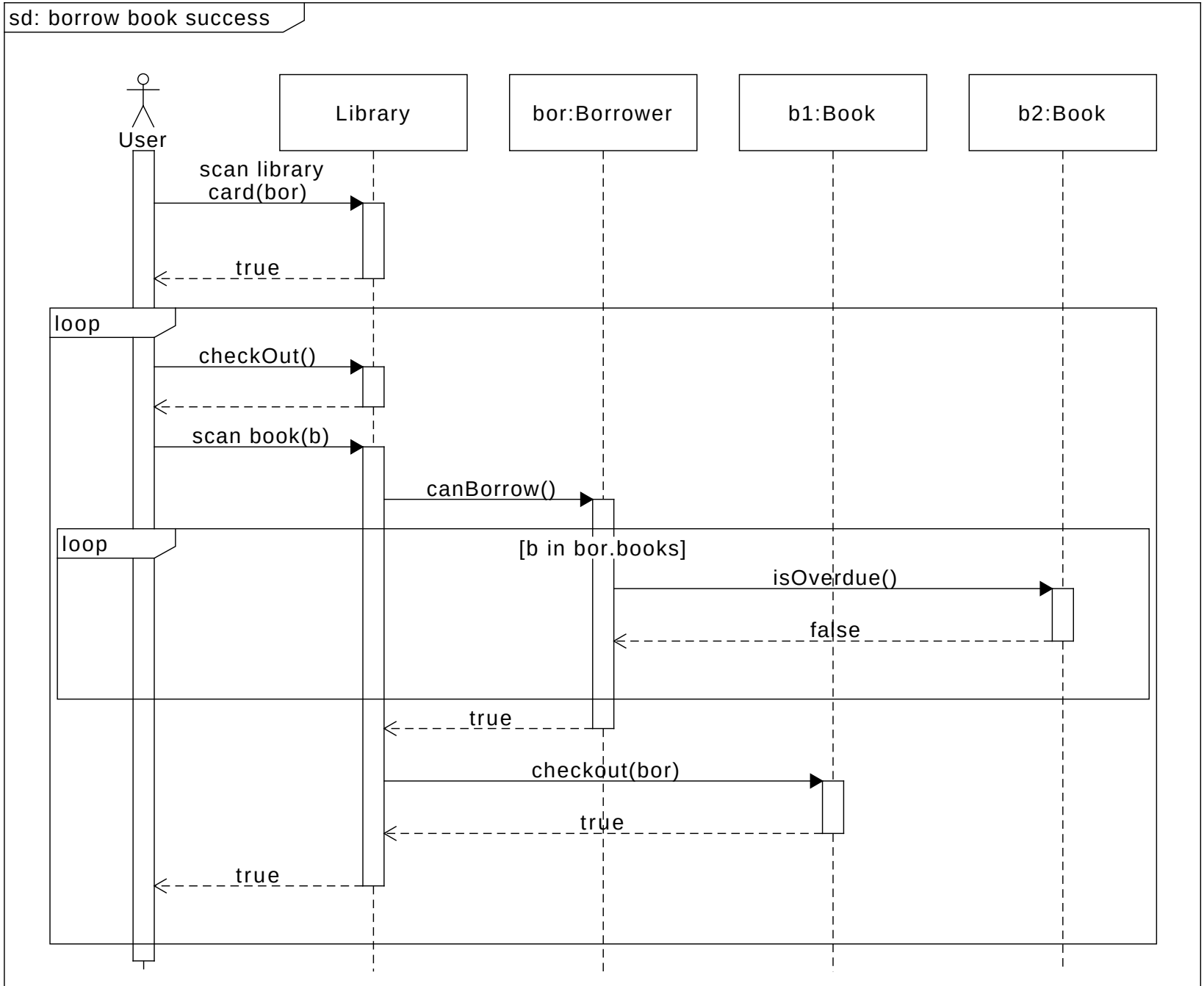
Implementation: Class Diagram



Library SM


...
Idle
scan library card (l)
User scanned
[users->contains(l)]/
return true
user ok
check out/ return
book scanned
[not users->contains(l)] / return false
scan book/ (cb,msg) := bor.canBorrow()
after 10min
[not cb] / return (false,msg)
can borrow?
[cb] / book := books->select(b| b.signature = s);
book.check_out(); return true

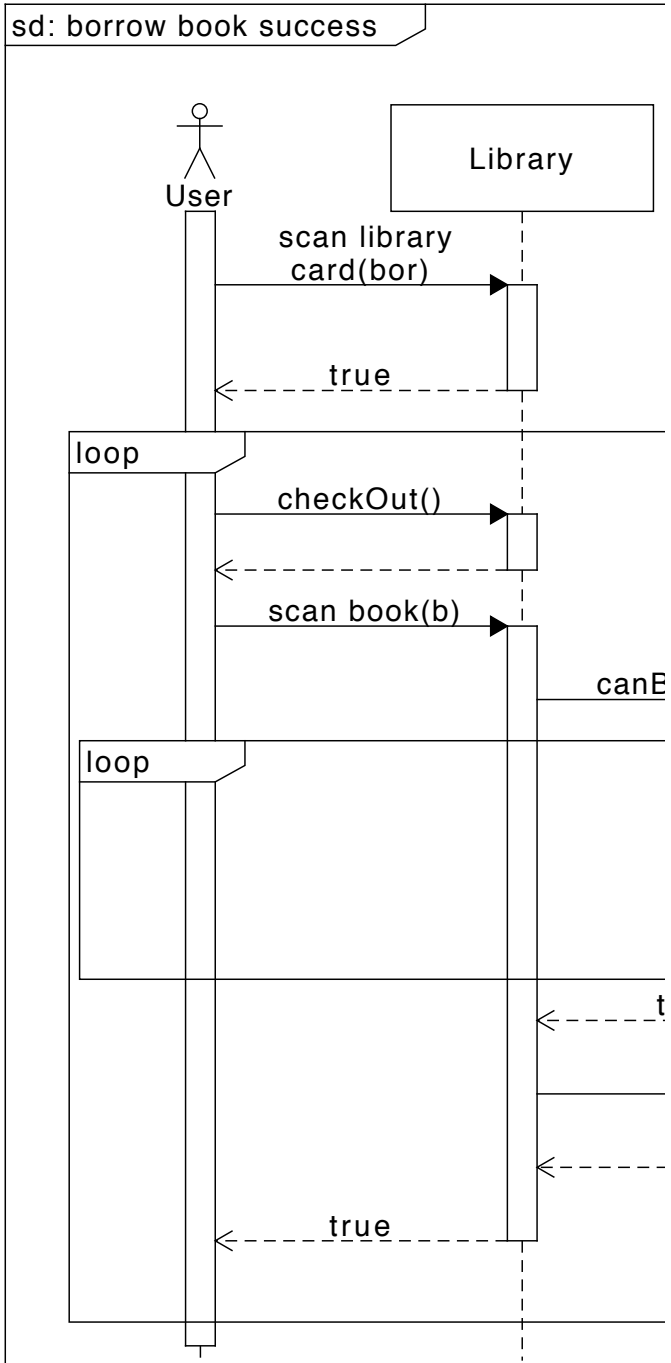
Use Case success scenario realisation



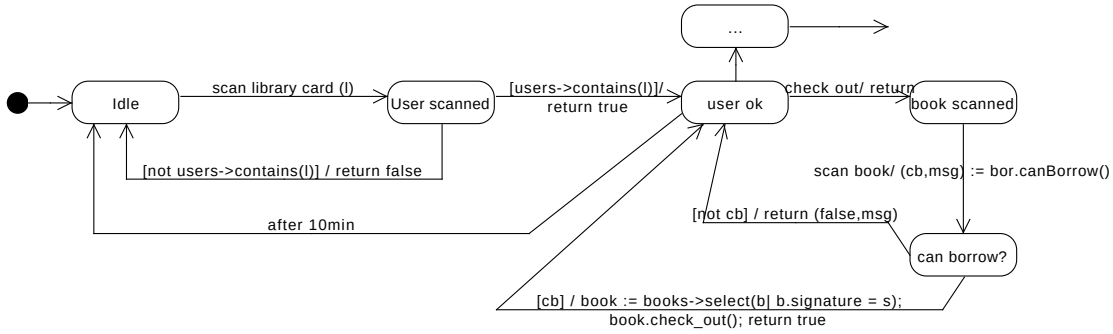
Use case *borrow book*: User interactions

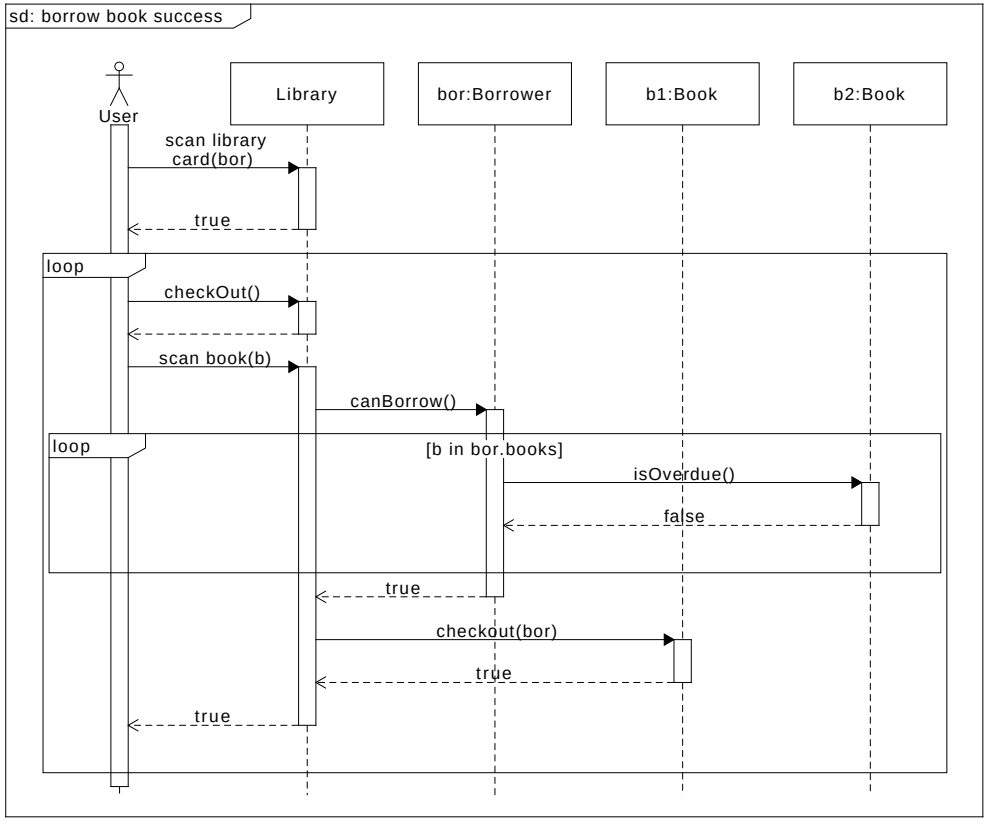
Use case *borrow book*:
Basic course of events

1. User scans his library card
2. User selects check out
3. User scans the book
4. System confirms loan



Library SM


...
Idle
scan library card (l)
User scanned
[users->contains(l)]/
return true
user ok
check out/ return
book scanned
[not users->contains(l)] / return false
scan book/ (cb,msg) := bor.canBorrow()
after 10min
[not cb] / return (false,msg)
can borrow?
[cb] / book := books->select(b| b.signature = s);
book.check_out(); return true


sd: borrow book success
User
Library
bor:Borrower
b1:Book
b2:Book
scan library
card(bor)
true
loop
checkOut()
scan book(b)
canBorrow()
loop
[b in bor.books]
isOverdue()
false
true
checkout(bor)
true
true

Contents

Overview

Introduction to UML

Practical Informations

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
  - Use Case Realizations

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
  - Dates: TBD (in the exam period)

Final project

- 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

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 Constraint 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