Design of Software Systems (Ontwerp van SoftwareSystemen)

1 Introduction

Roel Wuyts

OSS 2014-2015

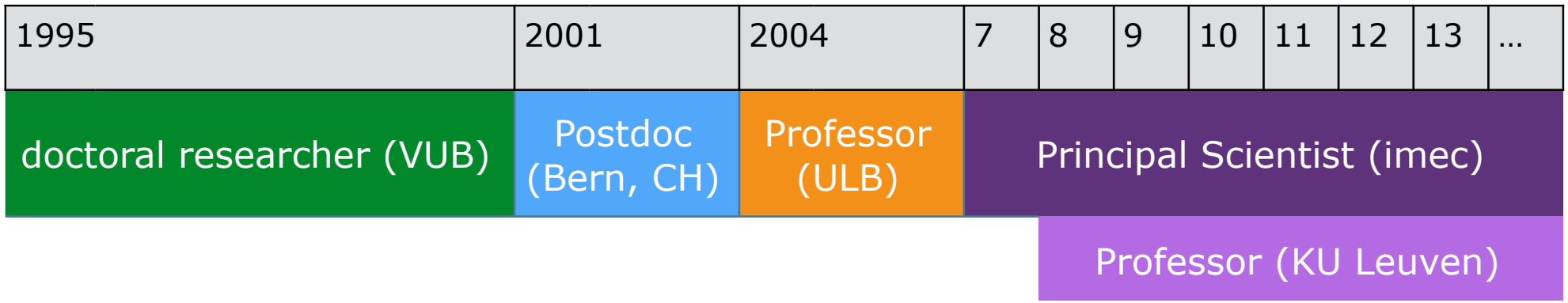
People

- Roel Wuyts
  - URL: http://roelwuyts.be/
- Philippe De Ryck
  - philippe.deryck@cs.kuleuven.be
- Mario Henrique Cruz Torres
  - mariohenrique.cruztorres@cs.kuleuven.be
- Pieter Agten
  - pieter.agten@cs.kuleuven.be

About me...

Logic Meta Programming Language Soul
Reflection, language symbiosis
Co-evolving Design & Implementation

In-memory object versioning
Aspect-oriented Programming



Traits (OO method composition model)
ClassBoxes (OO module composition model)
Data-centric component model for
hard-realtime embedded systems

Reengineering & Program Visualization

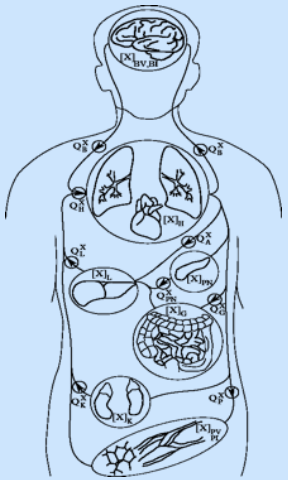
CleanC Eclipse Plugin
Dynamic scheduling of CPU/GPU tasks
+ simulator
High Performance Computing

High Performance Computing for Life Sciences

How to parallelize and distribute ?

How to deal with Big Data and Big Compute ?

How to let different stakeholder cooperate ?



SAEM
Bayesian PK/PD

BWA-Cilk
elPrep
BWA-TBB-aln
BWA-TBB-mem
Scientific Workflow Languages

(initial results)

Course Goals

"This course is concerned with the design of software systems. The focus lies on object-oriented methods. The primary objective is learning how to take design decisions by comparing positive and negative aspects of possible design solutions with respect to analysis and requirements, design, implementation and organizational impact. The theoretical aspects of the course are applied in a group project where a non-trivial, existing (but new to the students) application is extended with new functionality. "

Prerequisite knowledge

"Solid knowledge of object-oriented concepts and practical experience with at least one object-oriented programming language. Practical skills needed to develop software, such as the usage of an Integrated Development Environment like Eclipse or Netbeans and version control software (such as subversion)."

- Overview of software development processes.
- Object-oriented analysis and design using the UML modeling language.
- Study, evaluation and usage of GRASP and design patterns.
- Implementation techniques for realizing high quality object-oriented implementations.
- Techniques for assessing the quality of the design and implementation of existing software systems.

Course Material

- Slides and links on the website of the course

  http://roelwuyts.be/OSS-1415/

- Material
  - Applying UML and Patterns (3rd ed.), *Craig Larman*.
  - Design Patterns: Elements of Reusable Object-Oriented Software, *E. Gamma, R. Helm, R. Johnson, J. Vlissides*.
  - Refactoring: Improving the Design of Existing Code, *M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts*.
  - "No Silver Bullet: Essence and Accident in Software Engineering ", *F.P. Brooks*.

Project: applying the theory in practice

- Group Project
  - Number of persons in a group not yet known!
- Three iterations:
  1. Investigate and evaluate an existing implementation

     Analysis of an existing system
  2. Extend it (Trade-offs!)

     Decide what to modify to realize the extension
  3. Refactor it

     Clean up and maybe realize a smaller extension

Project Effort

- Effort of 120 hours / student.
- It is possible that you spend more or less !
  - notify me in time of possible discrepancies

Escalation Policy

- Groups do not always function smoothly
  - But dealing with this is part of your education
- In case of problems:
  - discuss within group.
  - if it cannot be resolved: mail to your assistant (with me in cc) to describe the problem.
  - assistant may decide to involve me if necessary.
- In case of problems with assistant: contact me.
- In case of problems with me: contact MA1 responsible.

Course grading: First session

- Score for Group Project defense (grade P)
- Individual oral exam (grade I)
- Grading Algorithm:

```
If  P <= 5 : final grade = P

elif I <= 8: final grade = I

else  final grade = (P + I) / 2
```

- If we find large work discrepancies within a group, specific grades for that group/person can be given

even 0 is possible when not collaborating !

Course grading: Second session

- Second session
  - Continuing project individually if P <= 5 (P')
  - New individual oral exam (I')
    - Same grading algorithm as first session but:
      - P replaced with P' (if applicable)
      - I replaced with I'

Project Defense

- Each of you gets questions and answers. Then other group members can provide more information.
- Questions originate from your report, design and implementation.
- You get two kinds of feedback:
  - during the defense:our questions and comments
  - right after the defense:

Grades

- 1st session:
  - Final grade ≥ 10 : done!
  - Final grade < 10 : redo in second session
- 2nd session:
  - Final grade ≥ 10 : done!
  - Final grade < 10 : credit not obtained

Questions ?

About You

- Let’s do an interactive “quiz”
  - there is no right or wrong for most of the questions here; goal is for me to learn your reflexes when faced with questions related to programming language, design, or implementation.

Is the following correct ?

"A message sent to super is sent to the parent of the object"

What is the result of the following expression?

```
class A {
   public void m(A a) { System.out.println("1"); }
}

class B extends A {
   public void m(B b) { System.out.println("2"); }
   public void m(A a) { System.out.println("3"); }
}


B b = new B();
A a = b;
a.m(b);
```

What do you think of the following implementation?

```
// Return null to signify end of file
protected IToken fetchToken() throws EndOfFileException {
    ++count;
    while (bufferStackPos >= 0) {
        // Tokens don't span buffers, stick to our current one
        char[] buffer = bufferStack[bufferStackPos];
        int limit = bufferLimit[bufferStackPos];
        int pos = bufferPos[bufferStackPos];

        switch (buffer[pos]) {

        case '_':
            t = scanIdentifier();
            if (t instanceof MacroExpansionToken)
                continue;
            return t;

        case '#':
            if (pos + 1 < limit && buffer[pos + 1] == '#') {
                ++bufferPos[bufferStackPos];
                return newToken(IToken.tPOUNDPOUND);
            }

            // Should really check to make sure this is the first
            // non whitespace character on the line
            handlePPDirective(pos);
            continue;
```

…

(390 lines of code in total)

Reuse versus hack

- Suppose you are responsible to add a new feature to an existing piece of software. The design of the existing software makes this hard. How do you decide whether to rewrite the existing software or whether to “hack in” the new feature ?

Understanding Existing Systems

- Your boss wants you to quickly develop a new tool. You decide to start from a large existing open-source application you found on SourceForge. How do you start ?

Object-Oriented Software Design Question

- A restaurant menu consists of dishes, e.g. “Flemish stew”, “Blood sausage with apples” and “Chicken Royale with Champaign”. Each dish consists of a number of ingredients and is either a starter, a main course or a dessert. The menu shows for each dish an authenticity score (1, 2 or 3), a calory score, as well as the price. Menus need to be printed in a variety of languages (dutch, french, english, japanese, arabic; some left-to-right and some right-to-left) and needs to be available on an interactive website (where a picture is shown of the dish). The menus change frequently with the seasons.

Why Software Engineering?

- Problem Specification → Final Program
- But ...
  - Where did the specification come from?
  - How do you know the specification corresponds to the user's needs?
  - How did you decide how to structure your program?
  - How do you know the program actually meets the specification?
  - How do you know your program will always work correctly?
  - What do you do if the users' needs change?
  - How do you divide tasks up if you have more than a one-person team?

What is Software Engineering? (I)

- Some Definitions and Issues
  - "state of the art of developing quality software on time and within budget"
- Trade-off between perfection and physical constraints
  - Software engineering deals with real-world issues
- State of the art!
  - Community decides on "best practice" + life-long education

What is Software Engineering? (II)

- "multi-person construction of multi-version software"

  – Parnas

- Team-work
  - Scale issue ("program well" is not enough) + Communication Issue
- Successful software systems must evolve or perish
  - Change is the norm, not the exception

Communication and Modeling

- Team-effort requires communication
- Results have to be communicated externally

UML

- Unified Modeling Language
- De-facto standard that I expect everybody to know and follow
  - working knowledge of at least the *use case*, *class*, *sequence* and *communication* diagrams
  - use throughout course (theory, practice, project)
- Self-study
  - I give a short overview
  - You do the study

General Goals of UML

- Model systems using OO concepts
- Establish an explicit coupling to conceptual as well as executable artifacts
- To create a modeling language usable by both humans and machines
- Models different types of systems (information systems, technical systems, embedded systems, real-time systems, distributed systems, system software, business systems, UML itself, ...)

11 diagrams in UML 2

- Class diagram
- Internal Structure Diagram
- Collaboration diagram
- Component diagram
- Use case diagram

Structural

- State machine diagram
- Activity Diagram
- Sequence diagram
- Communication Diagram

Dynamic

- Deployment diagram

Physical

- Package diagram

Model Management

Requirements Engineering and Use Cases

- Requirements: documented need for what a system or project should do
  - 37% of problems with software projects have to do with requirements
  - 25% of the requirements change during the project (and 35-50% in large projects)
- Therefore: embrace change!

Types of Requirements: FURPS+ categorization

Functional

features, capabilities

Use Cases

Usability

human factors, help, documentation

Reliability

frequency of failure, recoverability

Performance

Response times, throughput, accuracy, resource usage

Supportability

Adaptability, maintainability, configurability

+

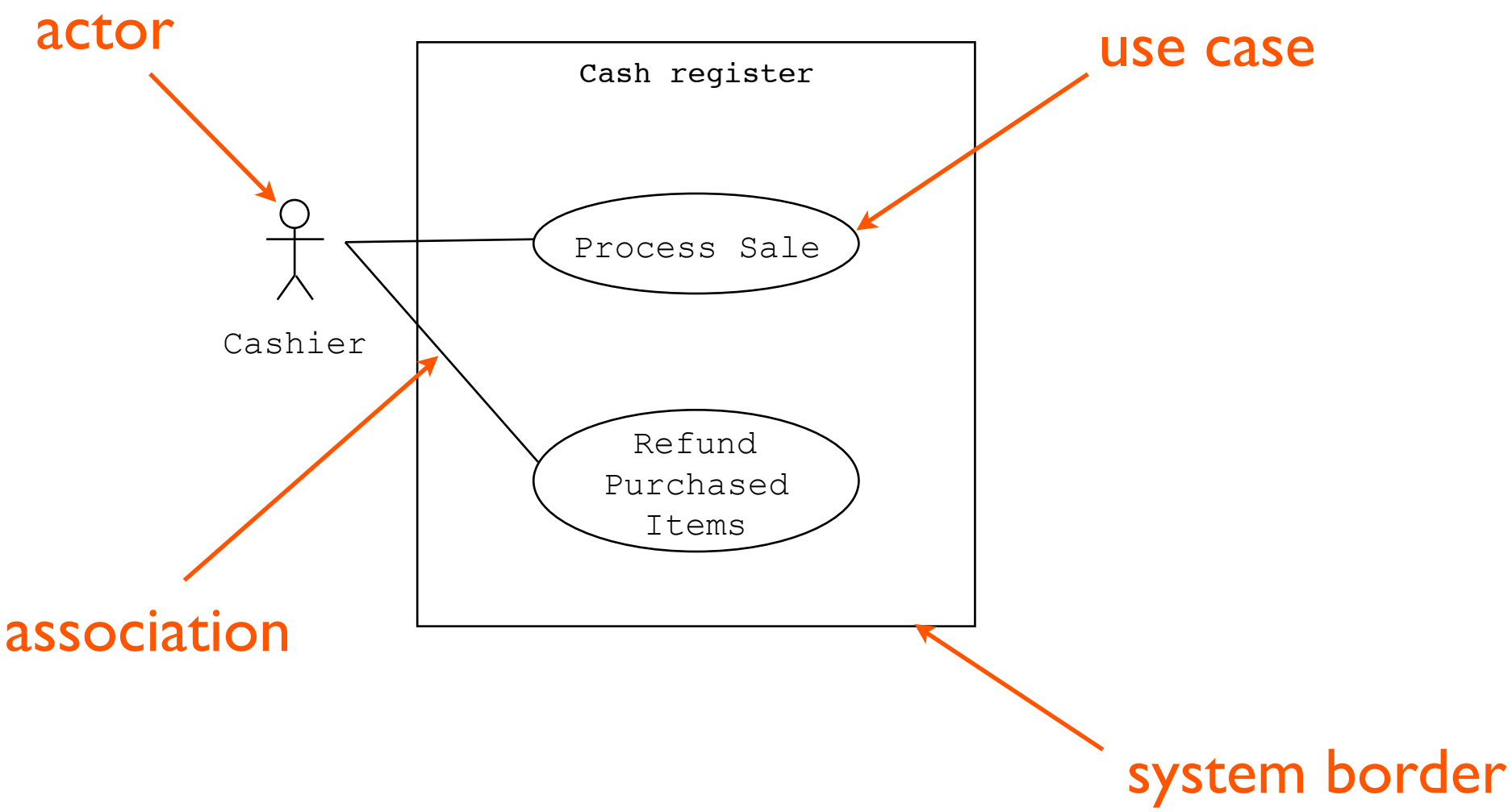
implementation, interface, operations, packaging, legal

Non-functional

Use Cases

- Stories that describe usage of the system
  - describe sequence of actions with an observable result for a specific actor
  - used by all kinds of stakeholders
- It does not describe the internal working of the system
  - What, not How
  - Responsibilities of the system are described

Use Case Diagram



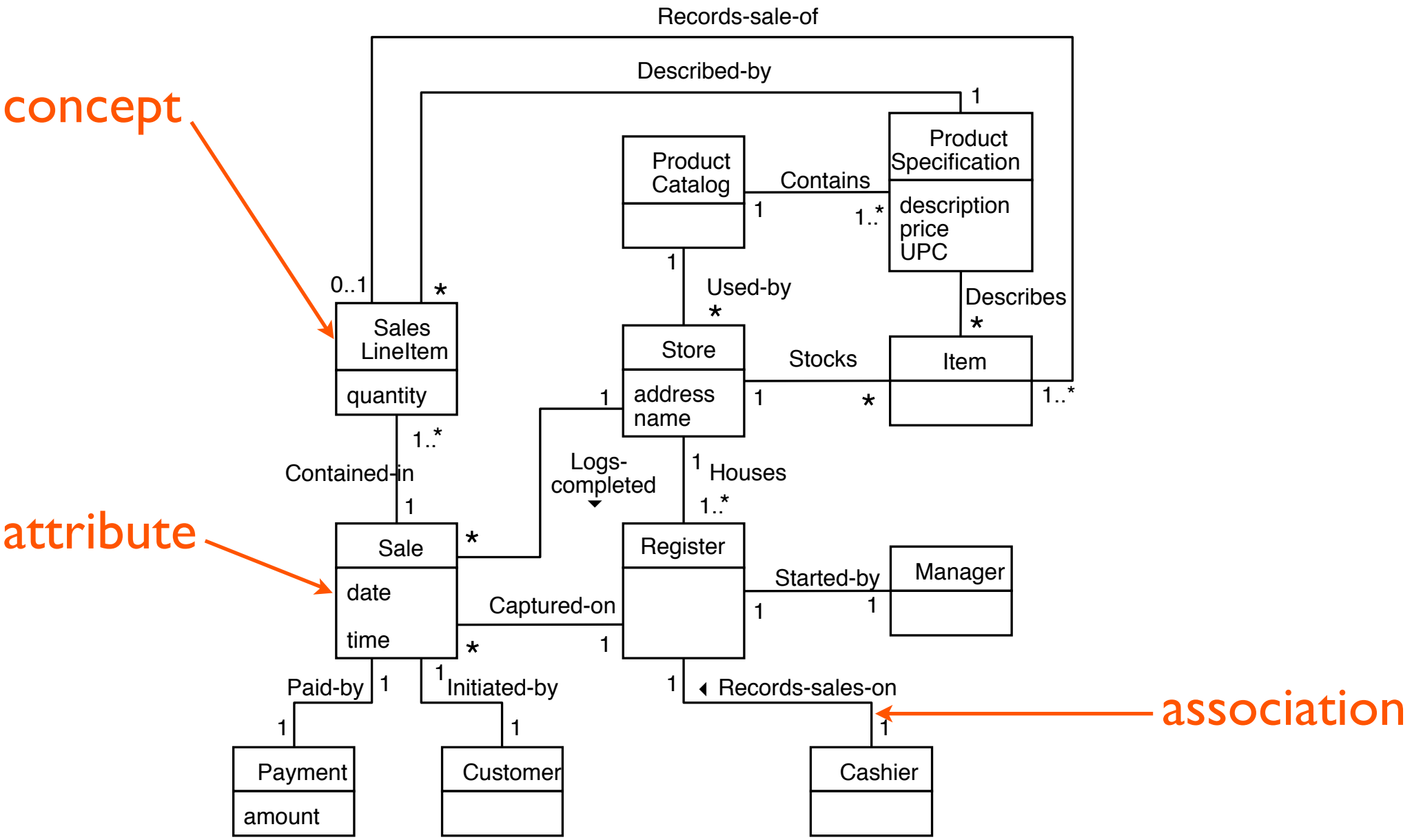
Fully Dressed Use Case Description

Use case:	**Process Sale**
Primary Actor:	Cashier
Stakeholders and interests:	• Cashier: wants accurate, fast entry, and no payment errors, as cash drawers shortages are deduced from his/her salary • Customer: wants purchase and fast service with minimal effort. Wants easily visible display of entered items and prices. Wants proof of purchase to support returns. • Manager, Government, Payment Company, ...
Precondition:	Cashier is identified and authenticated
Success Guarantee (postcondition)	Sale is saved. Tax is correctly calculated. Receipt is generated. Accounting and inventory are updated. Payment info is recorded.

Domain Modeling

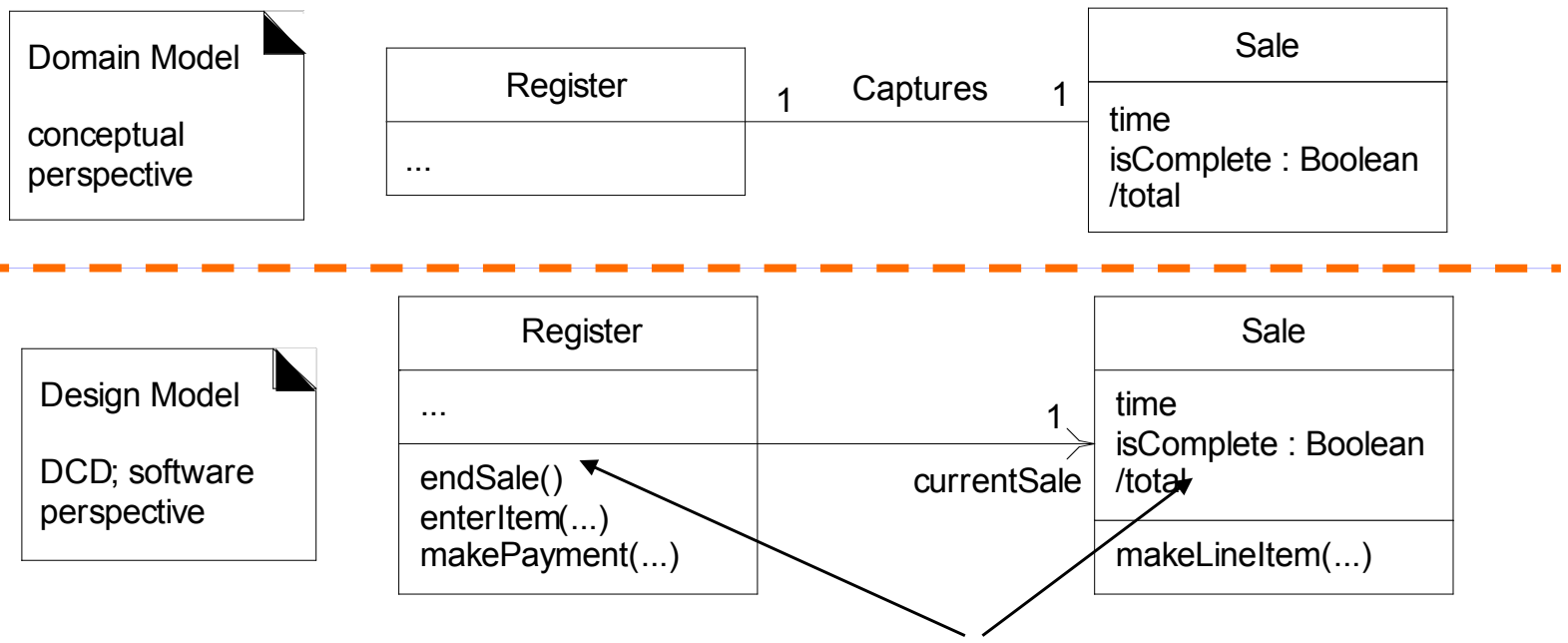
- A domain model describes meaningful concepts in the problem domain
  - again about the what, not the how
  - does not model design artifacts (how), but models conceptual artifacts, real-world things

Domain model for the Cash Register example



Class Diagrams

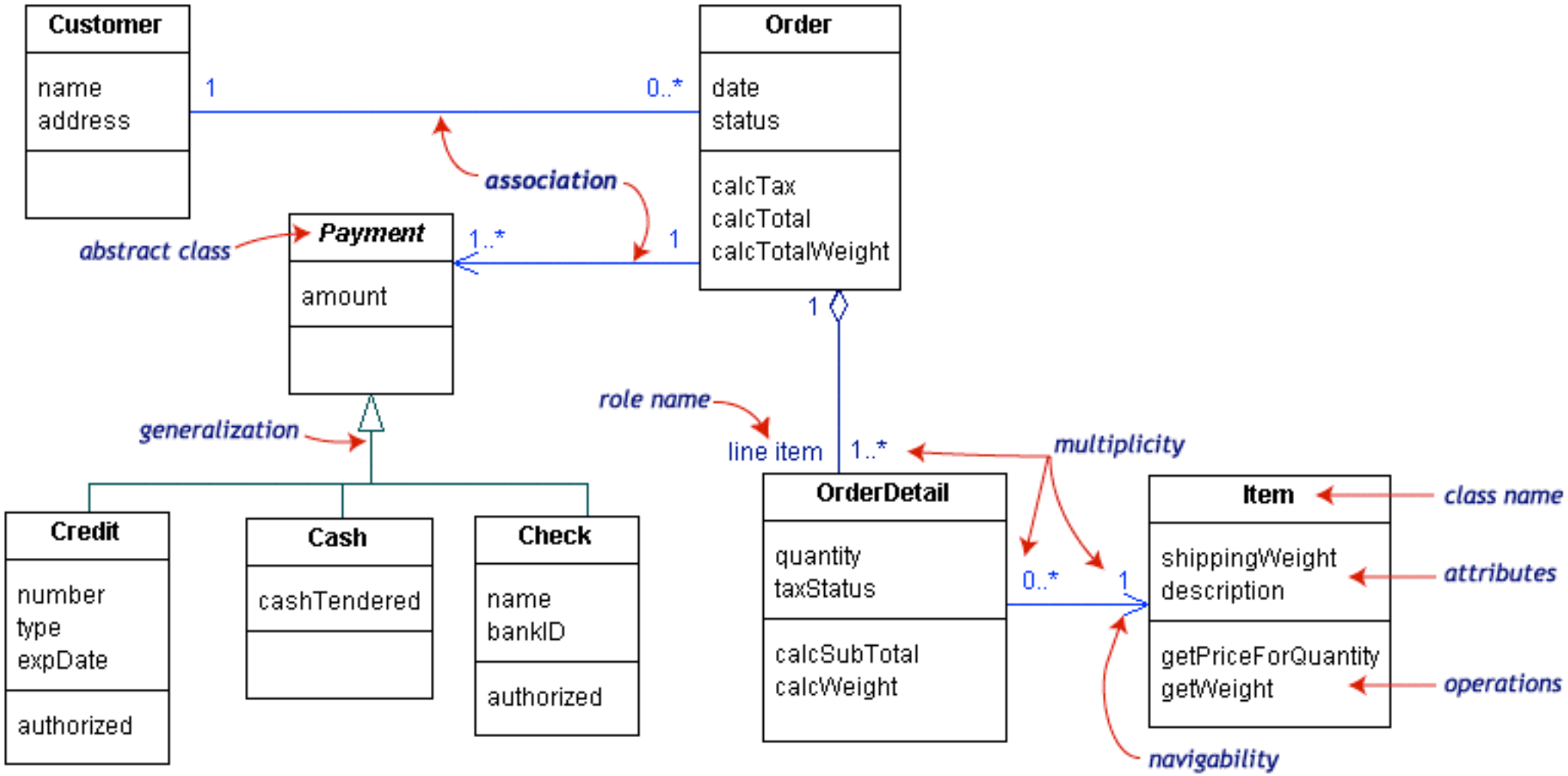
Domain model is the analysis *class diagram*
Don't show methods



Avoid showing no-argument constructors & getters/setters

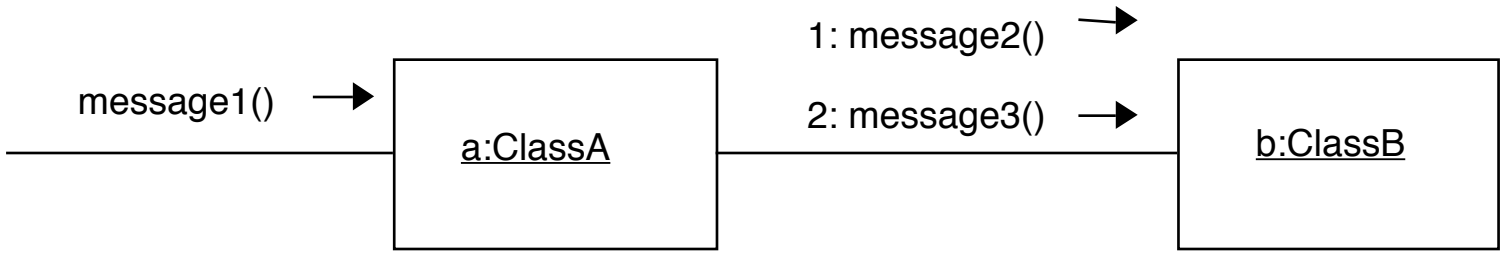
Design model *class diagrams* shows methods and visibility (arrowhead on association)
Register has reference to Sale; Sale does not have reference to Register

Class Diagrams



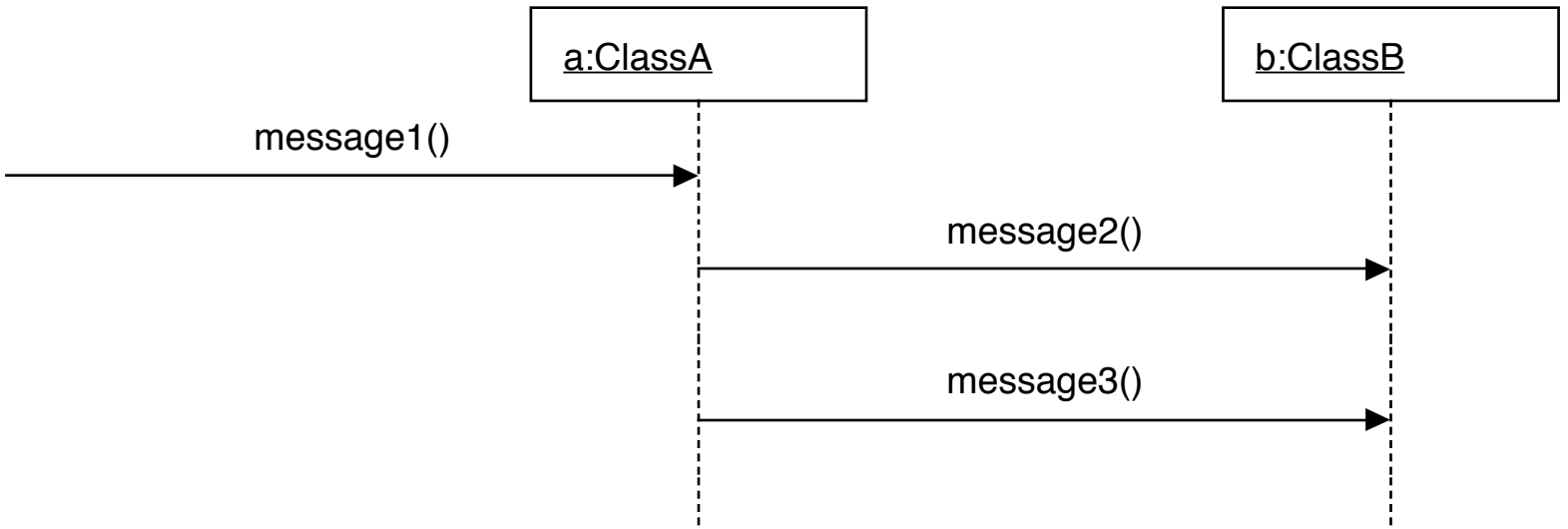
Interaction diagrams

- UML Interaction diagrams
  - model message-exchange between objects
- 2 kinds:
  - Communication Diagrams – focus on interactions
  - Sequence Diagrams – focus on time

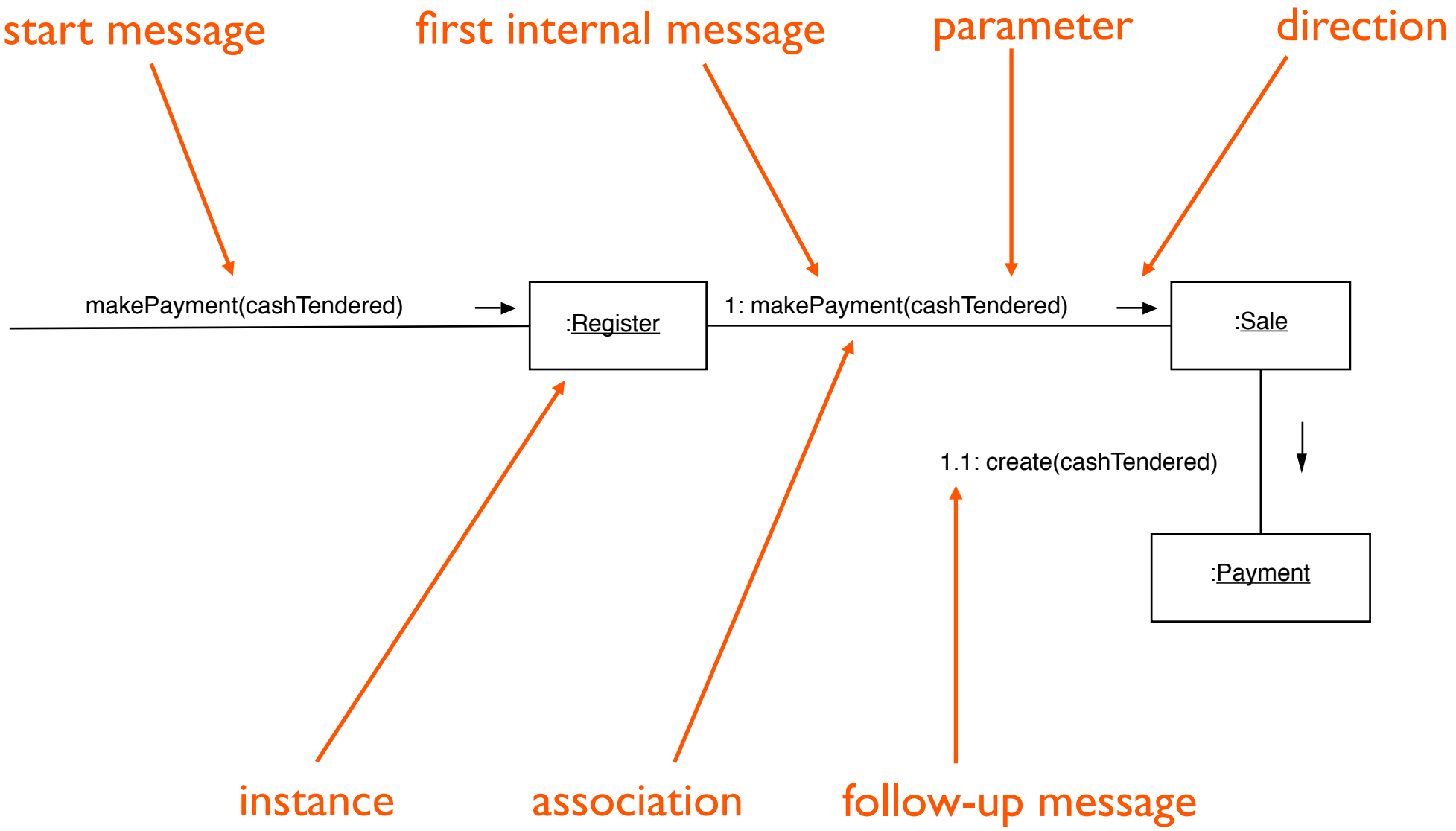


Interaction diagramma's

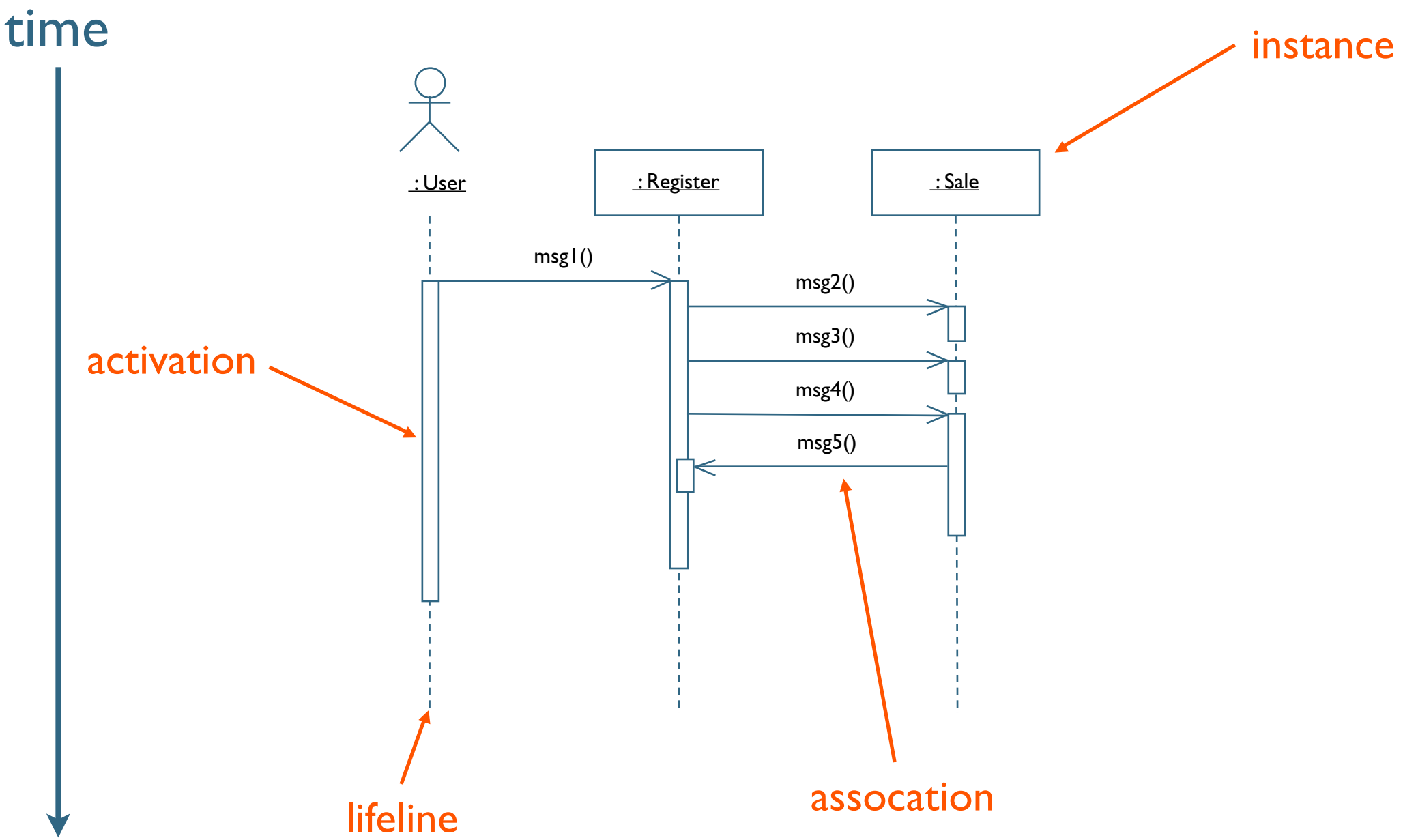
- UML Interaction diagrams
  - model message-exchange between objects
- 2 kinds:
  - Communication Diagrams – focus on interactions
  - Sequence Diagrams – focus on time



Communication Diagram Example



Sequence Diagram Example



Conclusion

- This course is about (OO) software design
  - Understand quality design and implementation
  - Make reasoned design decisions
  - Make trade-offs that balance quality, effort, design, and implementation
  - Be able to communicate your decision

http://roelwuyts.be/OSS-1415/

License: Creative Commons 4.0

http://creativecommons.org/licenses/by-sa/4.0/

You are free to:

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material

for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give **appropriate credit**, provide a link to the license, and **indicate if changes were made**. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the **same license** as the original.

No additional restrictions — You may not apply legal terms or **technological measures** that legally restrict others from doing anything the license permits.