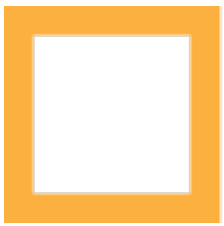
The Toolbox
工具箱
Vertical and horizontal contracts in large systems

Anders Kalhauge and Jacob Trier Frederiksen



cphbusiness

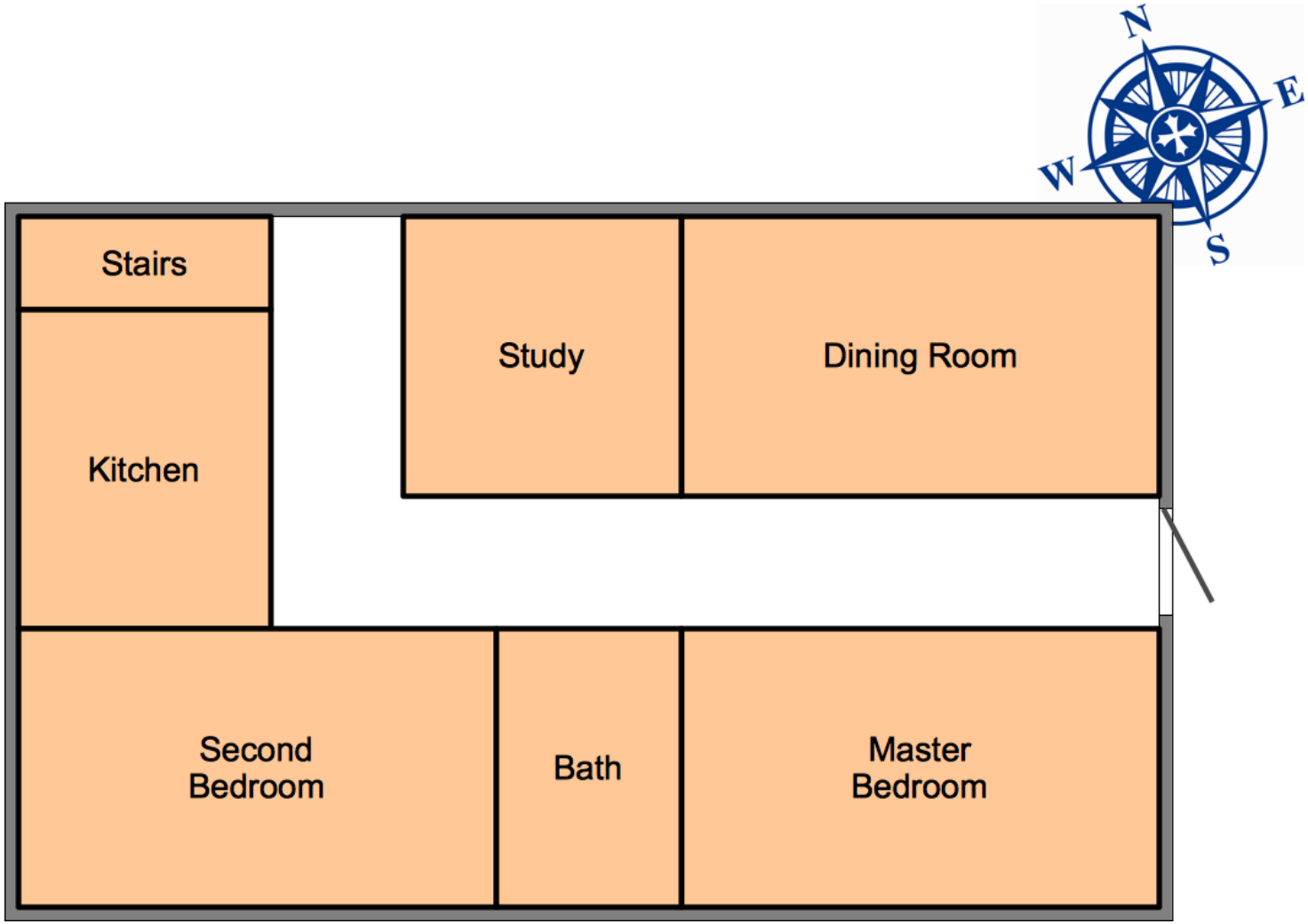
Fall 2019

The goal of this topic

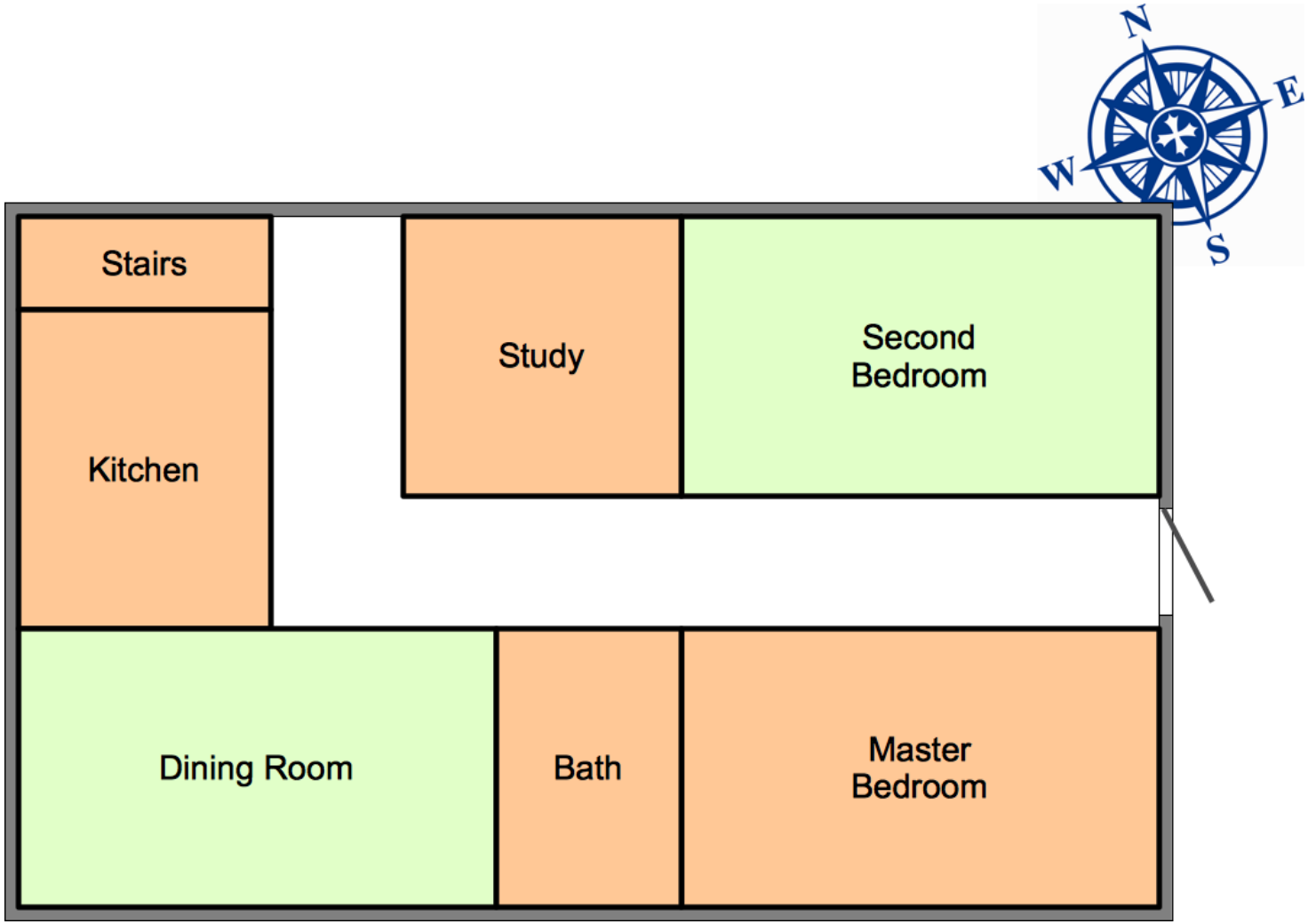
cphbusiness
COPENHAGEN BUSINESS ACADEMY

- You all understand the toolbox as a sound alternative, somewhere between the extreme formalization in Design by Contract, and no formalization in natural language contracts.
- You will master the central elements in the toolbox. Todays goal is **the Logical Data Model**.
- Understand and can use UML artifacts to define contracts between development groups working at the same level.
  - Vertical contracts
  - Horizontal contracts

Subsystems


N
E
W
S
Stairs
Kitchen
Study
Dining Room
Second Bedroom
Bath
Master Bedroom

Subsystems


N
E
W
S
Stairs
Kitchen
Study
Second Bedroom
Dining Room
Bath
Master Bedroom

Subsystems

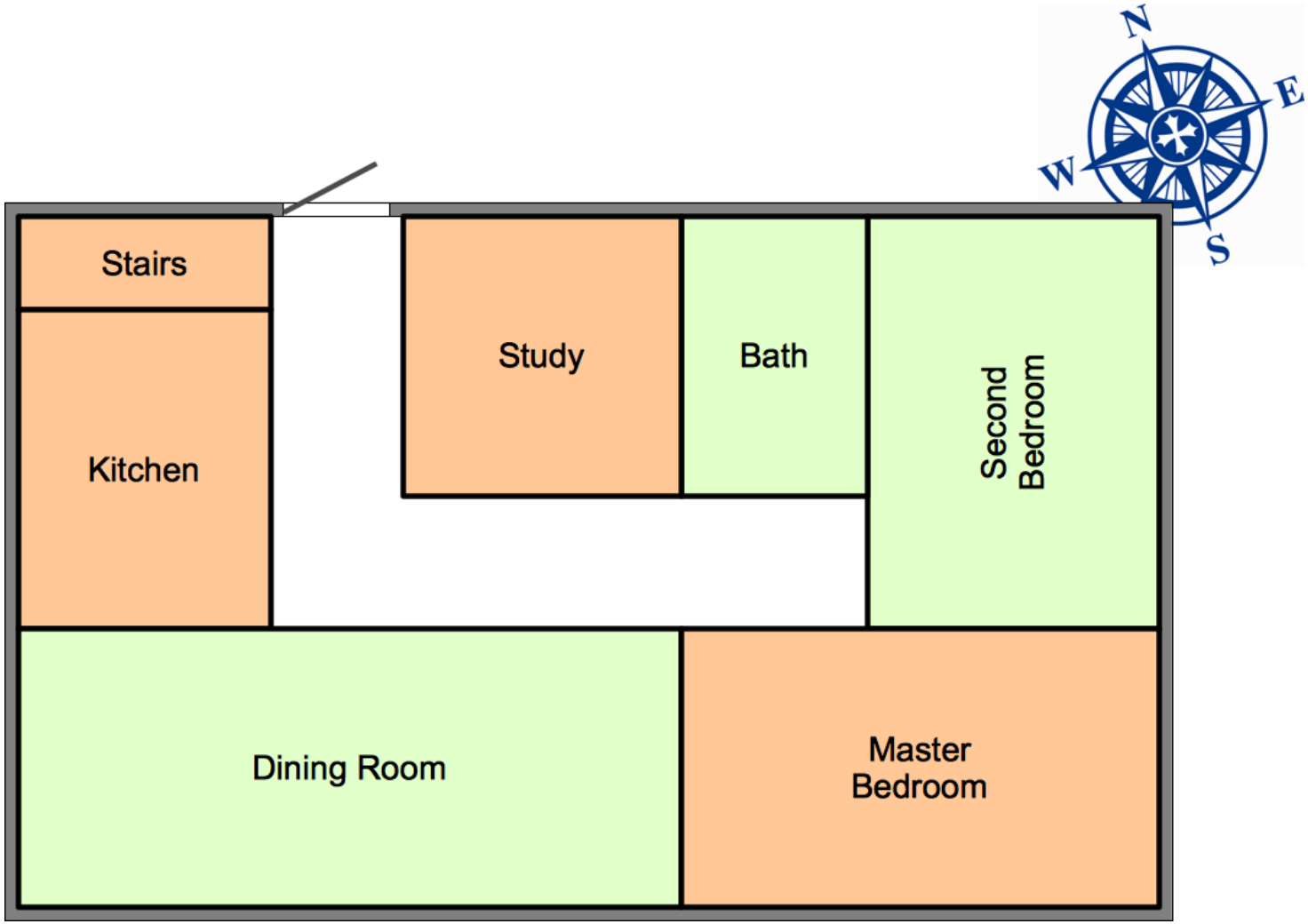

N
E
W
S
Stairs
Kitchen
Study
Bath
Second Bedroom
Dining Room
Master Bedroom

Table 6-1 Mapping of architectural and software engineering concepts.

	Architectural concept	Software engineering concept
Components	Rooms	Subsystems
Interfaces	Doors	Services
Nonfunctional requirements	Living area	Response time
Functional requirements	Residential house	Use cases
Costly rework	Moving walls	Change of subsystem interfaces

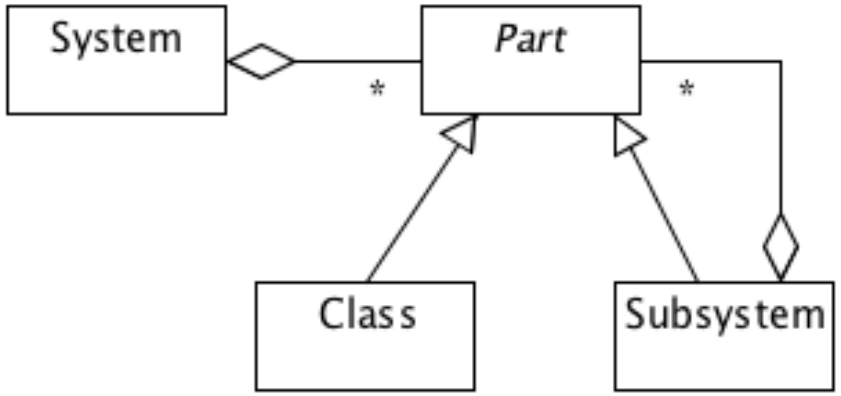
Subsystems

System design vs. object design

System design identifying large chunks of work that could be assigned to individual teams.

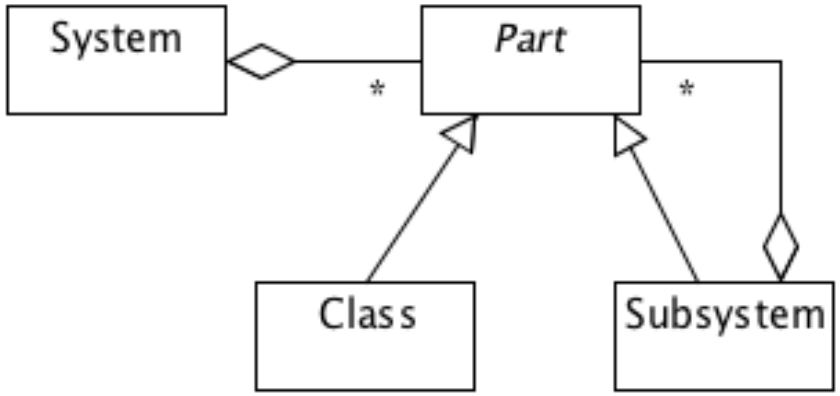
Object design specifying the boundaries between objects.

Subsystems



- Do you recognise this pattern?

Subsystems



- Do you recognise this pattern?
- What is the difference between a component and a subsystem?

Subsystems

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Component	Reusable encapsulated well defined software. Cannot stand alone.
Subsystem	Encapsulated well defined stand-alone software. Might or might not be an application by itself.
Application	An end-usable piece of software.

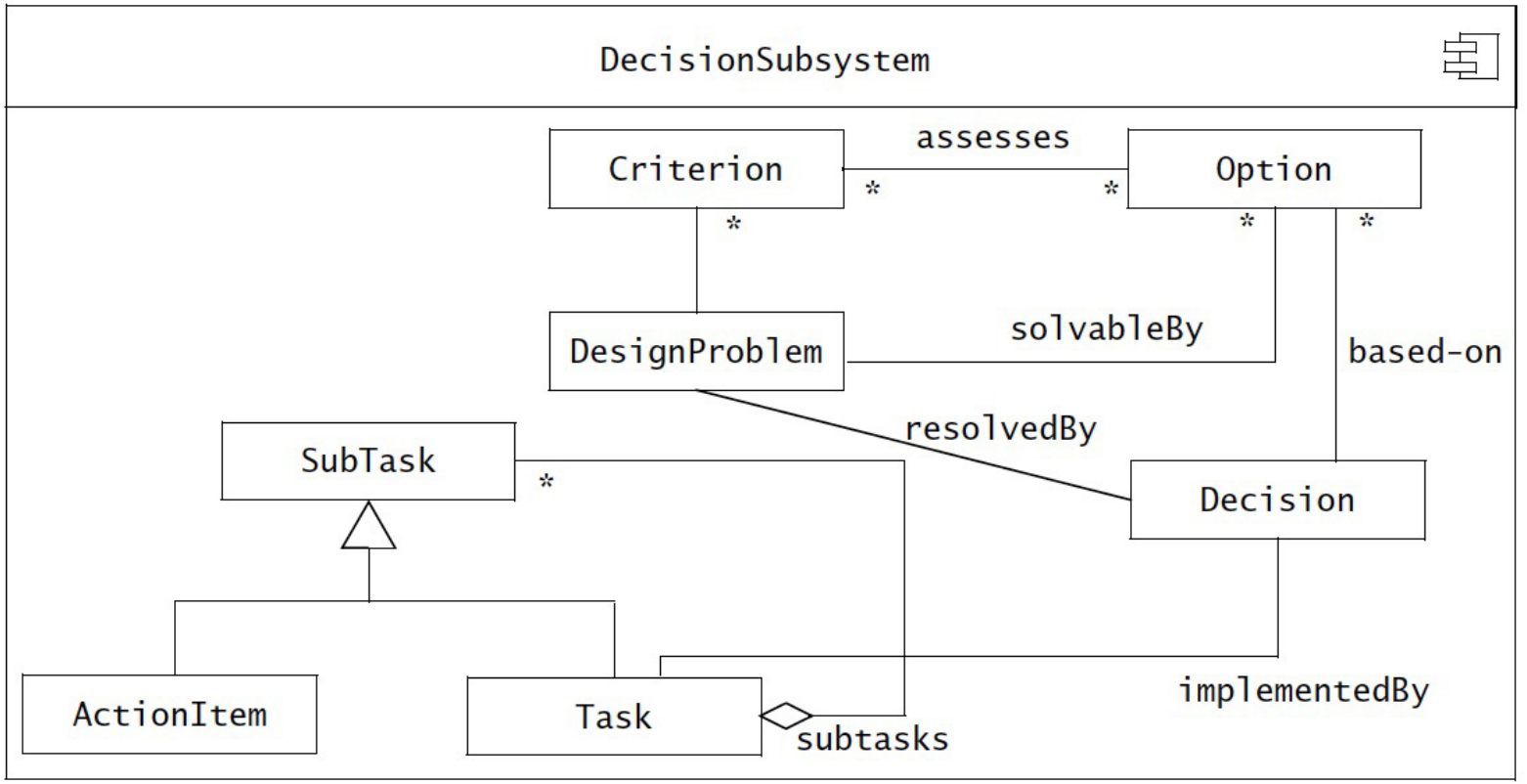


Figure 6-7 Decision tracking system (UML component diagram). The `DecisionSubsystem` has a low cohesion: The classes `Criterion`, `Option`, and `DesignProblem` have no relationships with `Subtask`, `ActionItem`, and `Task`.

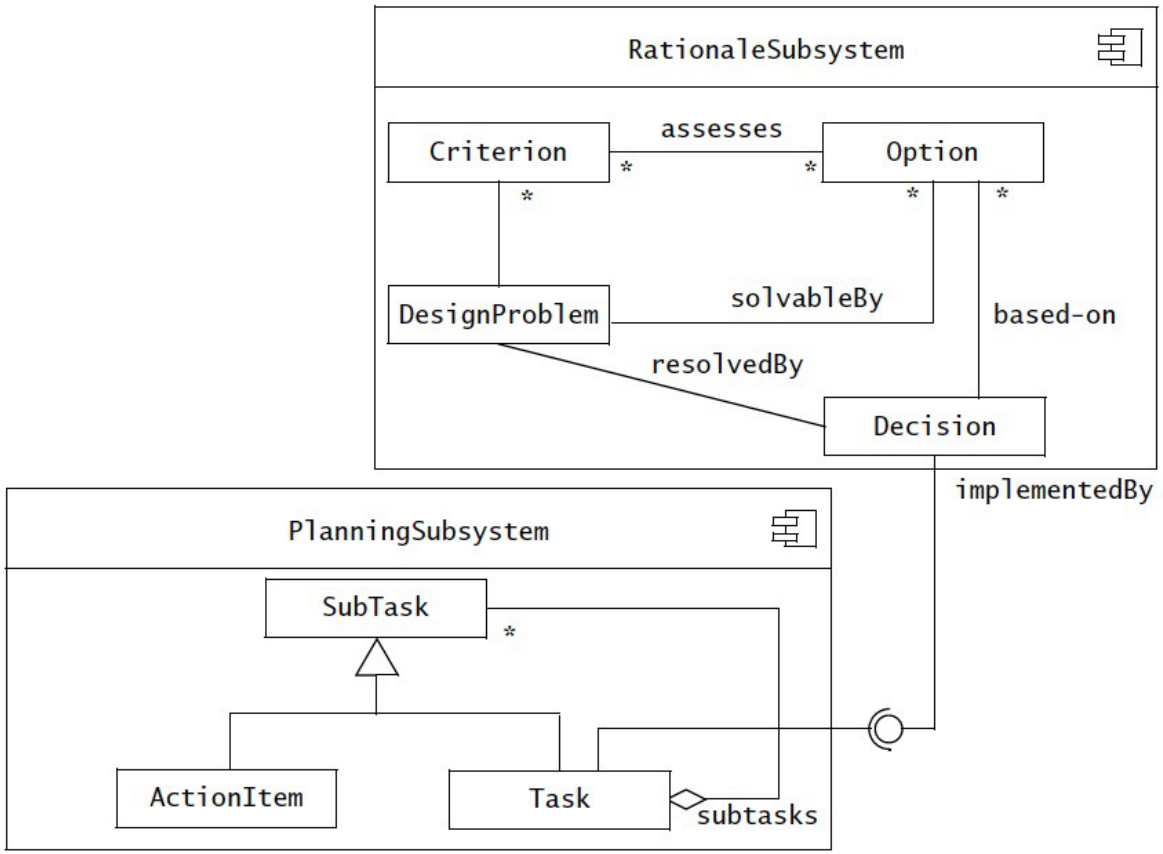


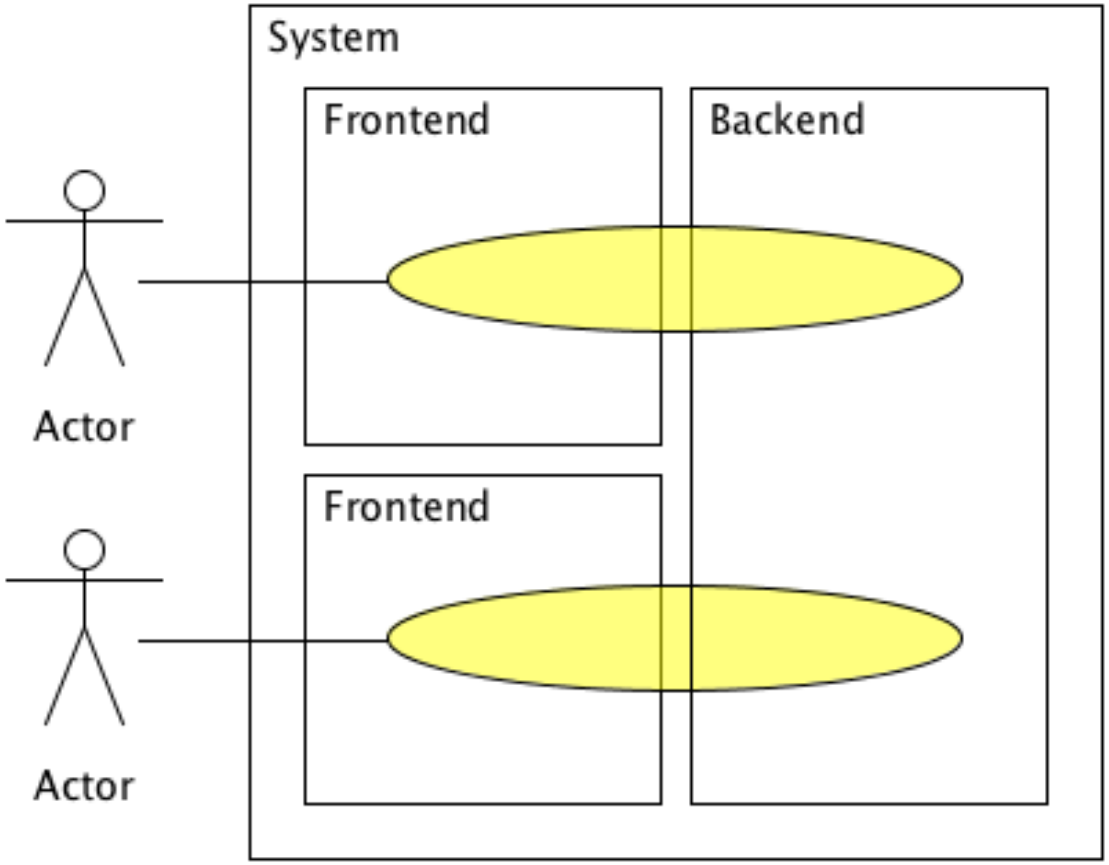
Figure 6-8 Alternative subsystem decomposition for the decision tracking system of Figure 6-7 (UML component diagram, ball-and-socket notation). The cohesion of the `RationaleSubsystem` and the `PlanningSubsystem` is higher than the cohesion of the original `DecisionSubsystem`. The `RationaleSubsystem` and `PlanningSubsystem` subsystems are also simpler. However, we introduced an interface for realizing the relationship between `Task` and `Decision`.

Strategies for dividing systems

- Layered/functional sub-systems (High Cohesion)
  - Fit to competences between developers
    and/or
  - Fit to distributions on machines
- Use-case based sub-systemer (Low coupling – primary on data level)
  - Fits requirements owner (users)
- Mixed or balanced division
  - Front ends – use-cases - presentation logic
  - Back end - technology - business logic

cphbusiness
COPENHAGEN BUSINESS ACADEMY

A balanced/mixed model



cphbusiness
COPENHAGEN BUSINESS ACADEMY

What's in the box?

Overview

What's in the box?

- Logical data model
- Use case model
  - Use case diagram(s)
  - Use case descriptions
  - System sequence diagram
  - System operation contracts
- Communication model
  - System operation contracts
  - Transfer objects
    - Data Transfer Objects (DTOs)
    - Exception Transfer Objects (ETOs)
- Verification strategy

Logical data model

cphbusiness
COPENHAGEN BUSINESS ACADEMY

What is a logical data model?

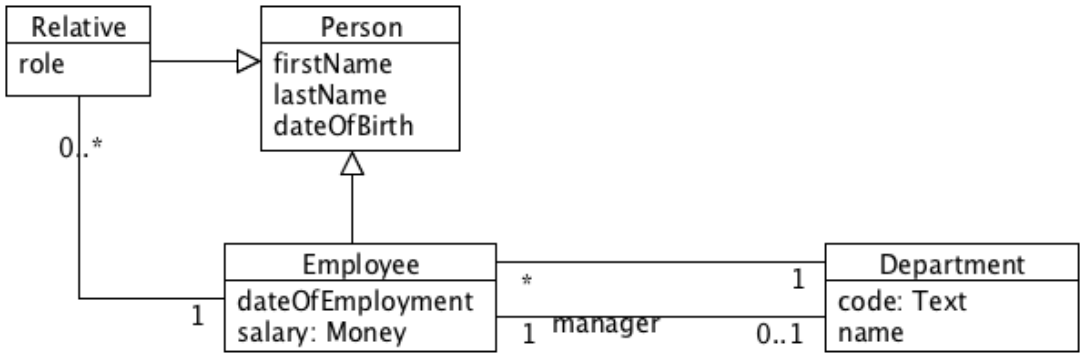
- It models the system state.
- Expresses valid pre- and postcondition states.
- Expresses possible system state changes.

Get inspired from **nouns** in textual requirements

Logical data model

cphbusiness
COPENHAGEN BUSINESS ACADEMY

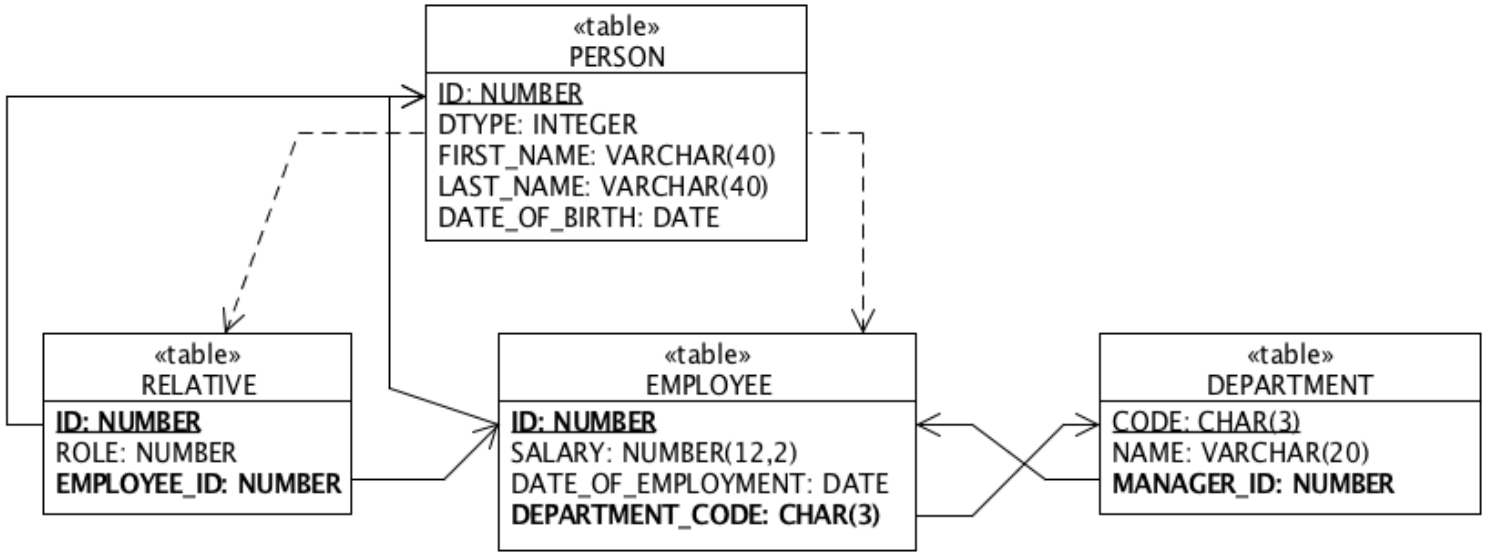
An example



- What should be persisted
- Only entities
- No implementation details
  - No ids unless they contain data (not necessarily wise)
  - Only abstract types

Logical data model

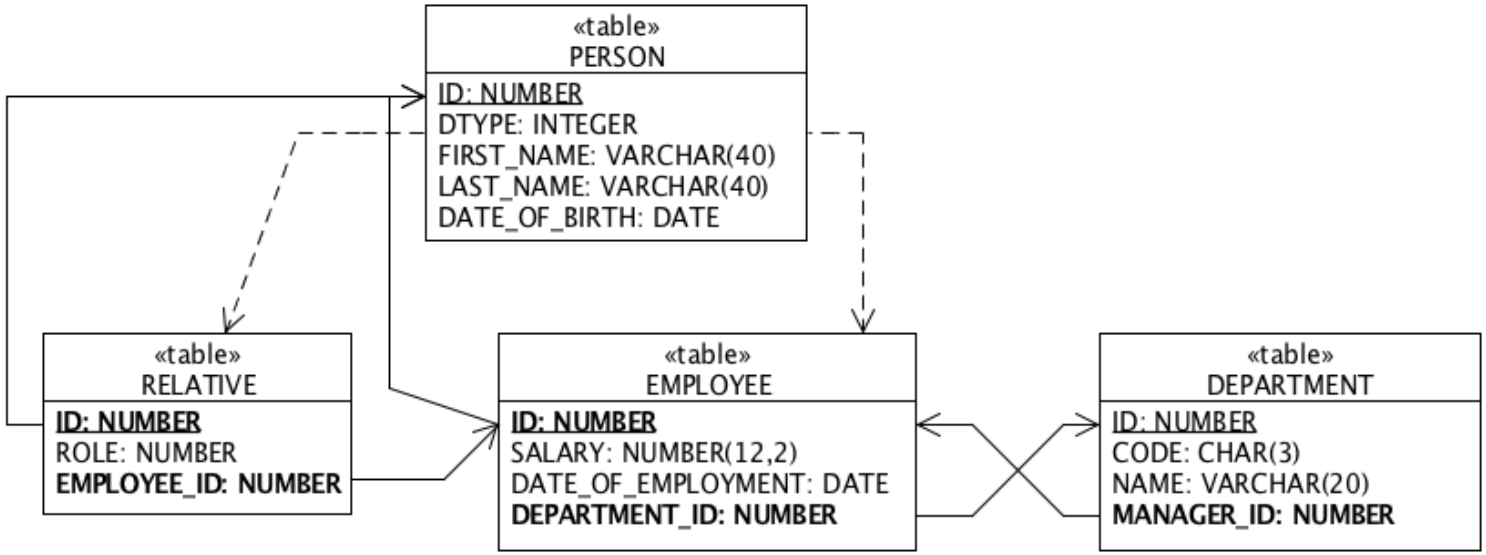
A relational implementation I



- Primary (underlined) and foreign (**boldfaced**) keys shown.
- Joined tables inheritance strategy, `DTYPE` discriminates between types.

Logical data model

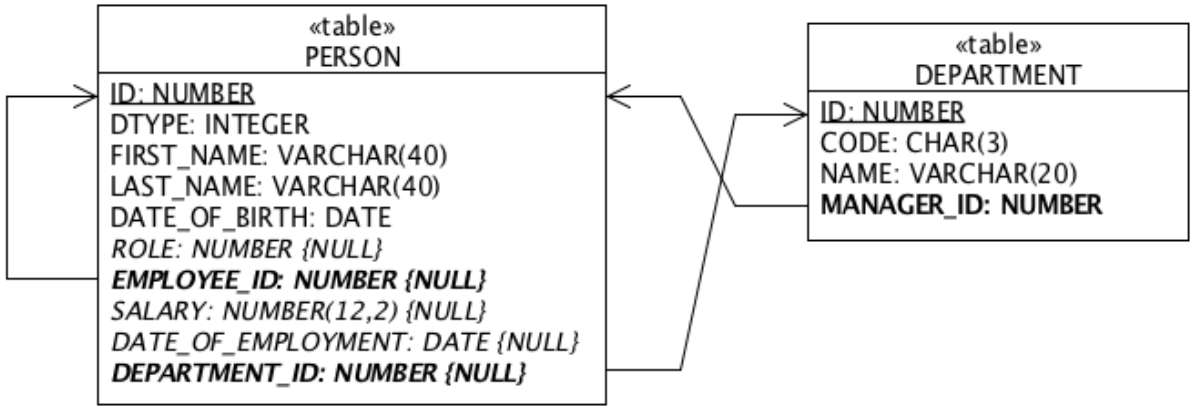
A relational implementation II



- Same as I, with no data bearing primary keys ☺

Logical data model

A relational implementation III

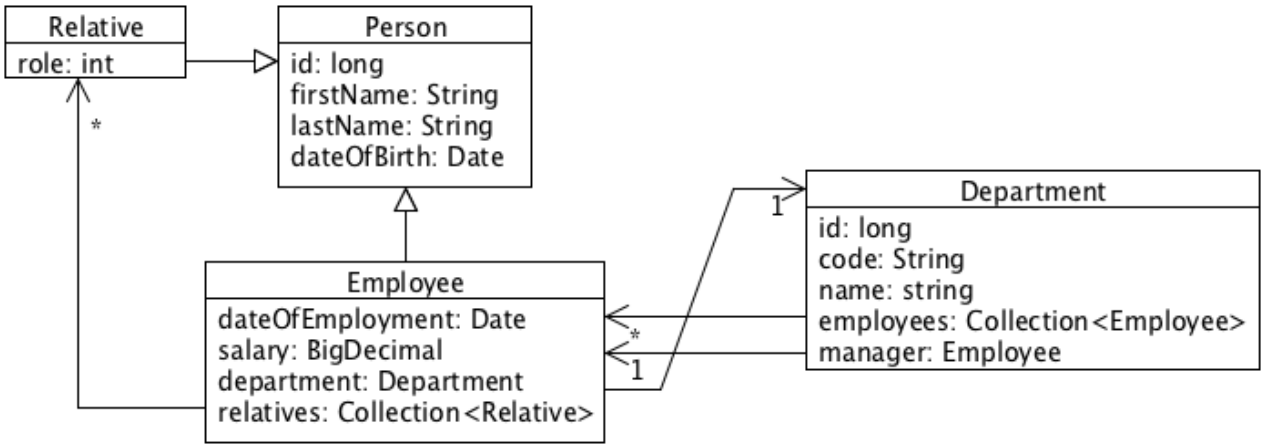


- Single table inheritance strategy
- “Irrelevant fields are nulled

Logical data model

cphbusiness
COPENHAGEN BUSINESS ACADEMY

An implementation with objects



- No associations, only references
- Id's to support "Object Relational Mapping"

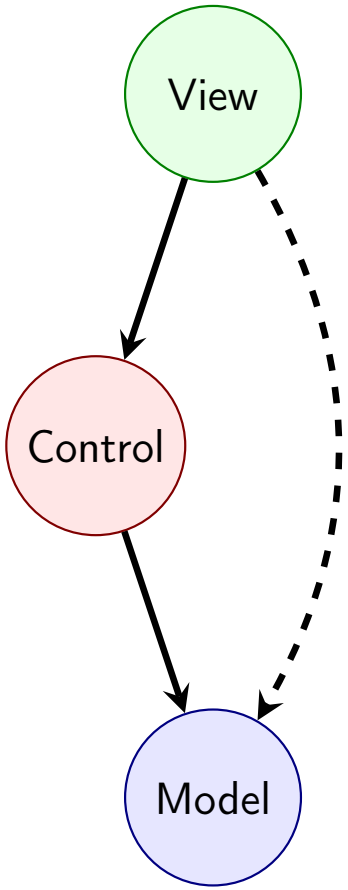
Use case model

More here next week . . .

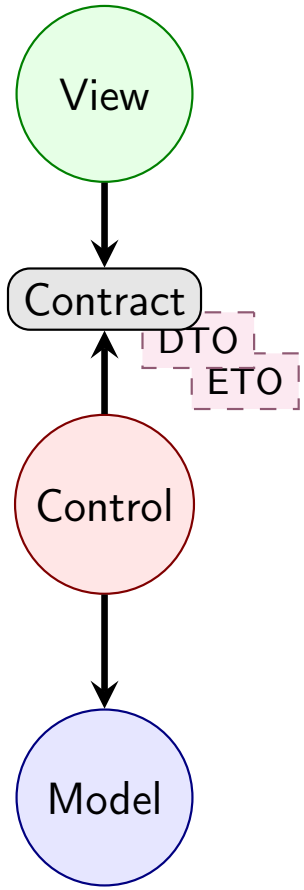
Get inspired from **verbs** in textual requirements

Communication model

System operation contracts - MVC pattern reviewed



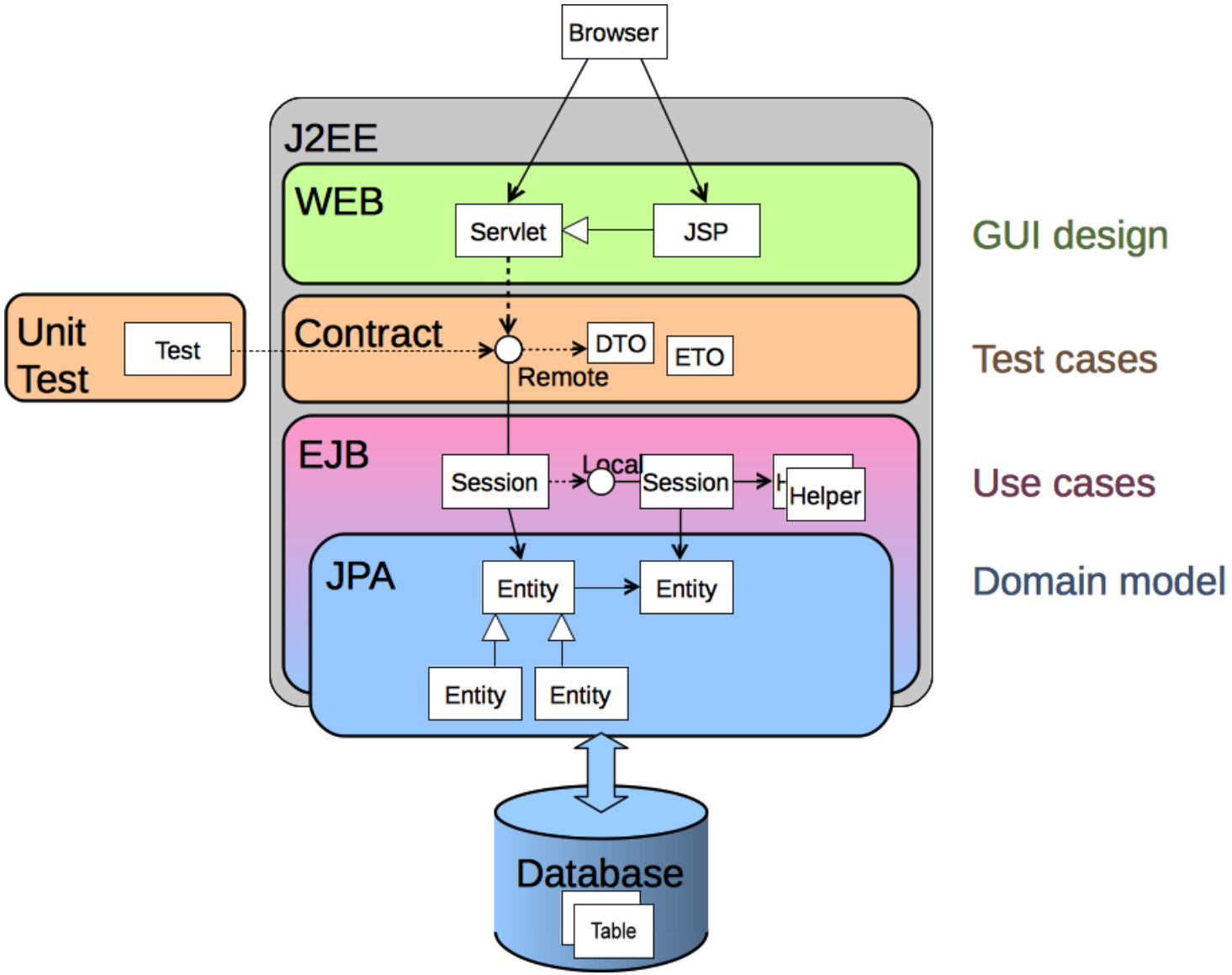
(a)



(b)

Communication model

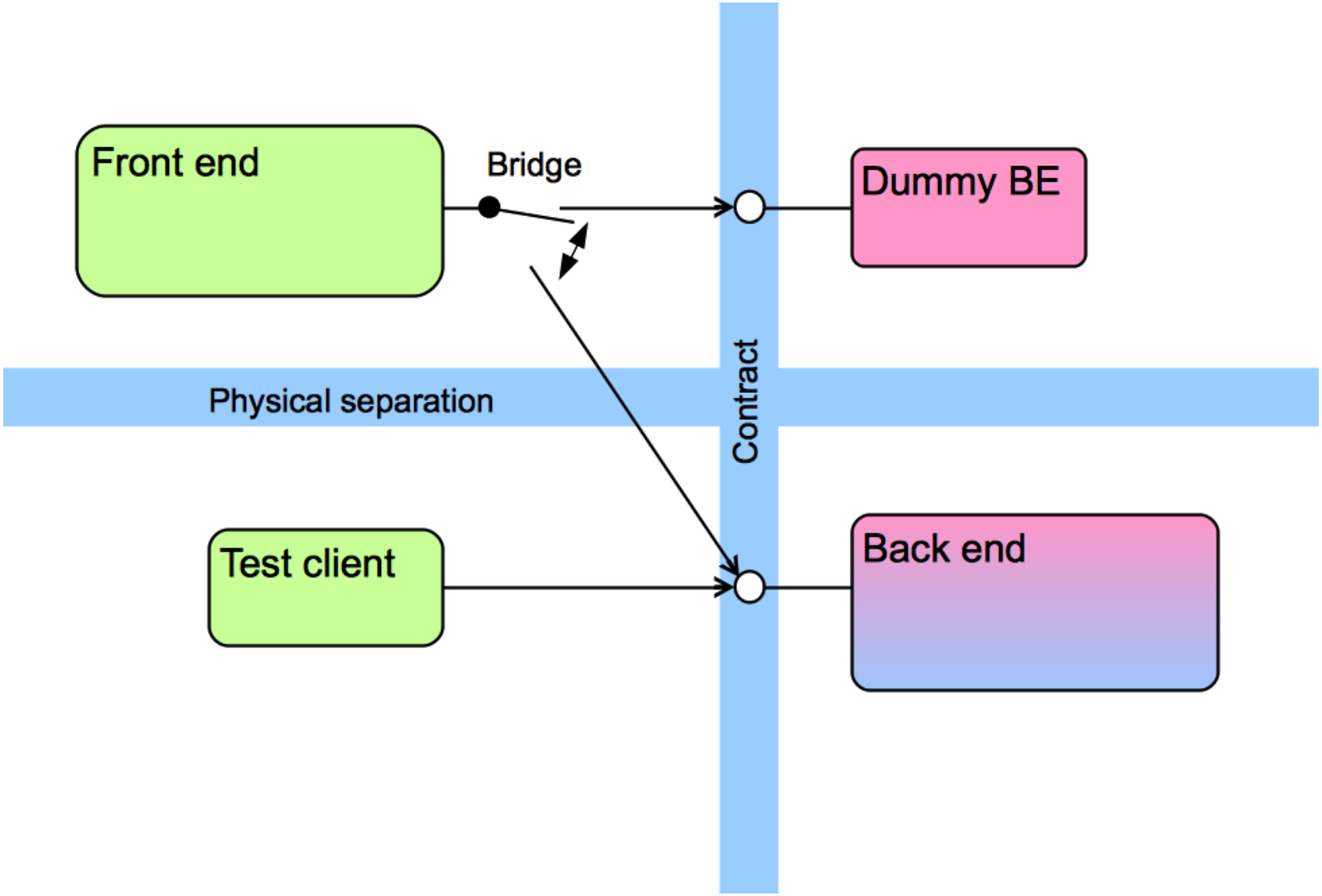
System operation contracts - Layers in EJB



Communication model

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Project setup with bridge



System operation contracts - "Remote" interface

The interface is the code based operation contract.

- Use strong typing.
  - use DTOs instead of simple data types.
- Make inline documentation (JavaDoc)
  - have documentation close to code - easier to update.
  - generates written code contracts.
- Implement the interface with a Remote facade in the "backend".
  - Changes to the backend code or to the interface will have less impact.
- Reference the interface from a Factory in the "frontend".
  - Change of backend can be done with practically no code changes in "frontend"

cphbusiness
COPENHAGEN BUSINESS ACADEMY

System operation contracts - Data Transfer Objects

Data transfer objects should be as abstract as possible when still being concrete. Use DTOs for request and return values.

- Efficiency
  - Packing related data together
  - reducing calls - network calls are expensive to establish
  - reducing data - bandwidth is still an issue
- Encapsulation
  - by hiding irrelevant or secret data
  - **by hiding actual implementation**
- Serializable

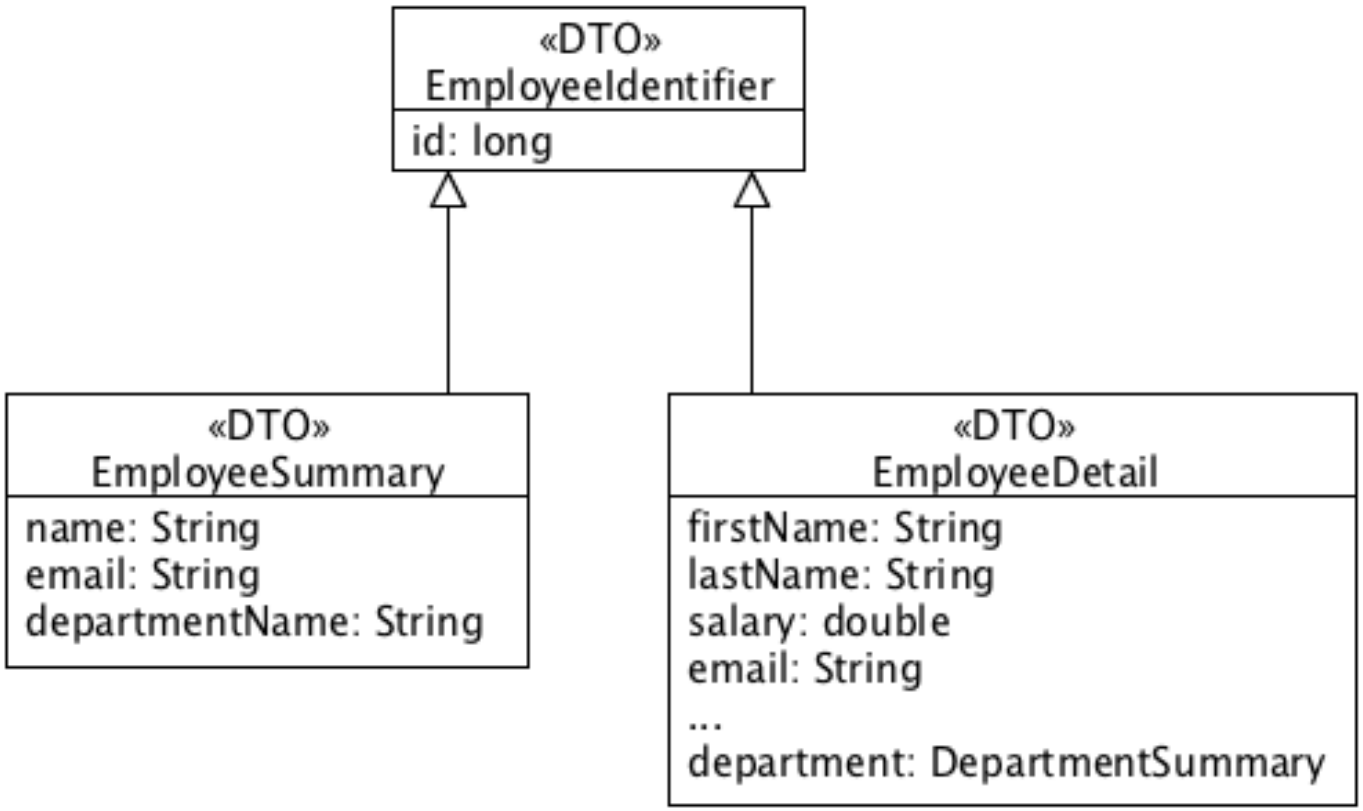
System operation contracts - Exception Transfer Objects

Exceptions are as valid, even less happy, return values from operations.

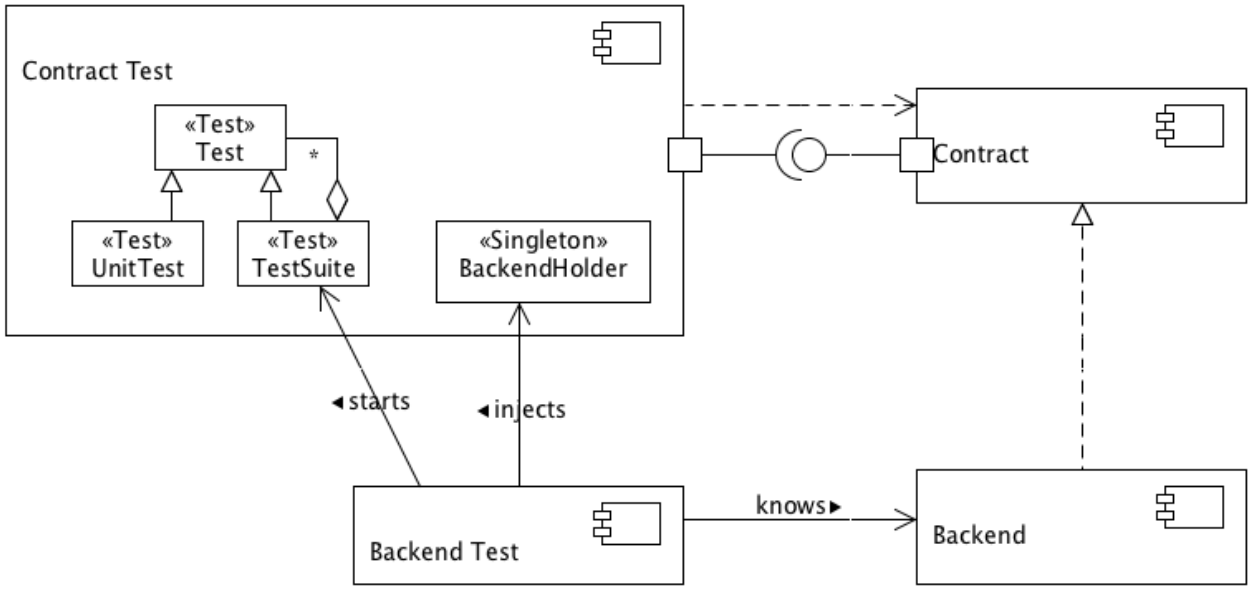
- User friendly - return only relevant information.
  - Preconditions: What precondition was violated (unchecked).
  - Postconditions: What went wrong (checked).
- Encapsulation
  - by hiding actual implementation
  - **revealing errors and their precise cause, is pleasing hackers**
- Serializable - in Java Exceptions are already Serializable

Communication model

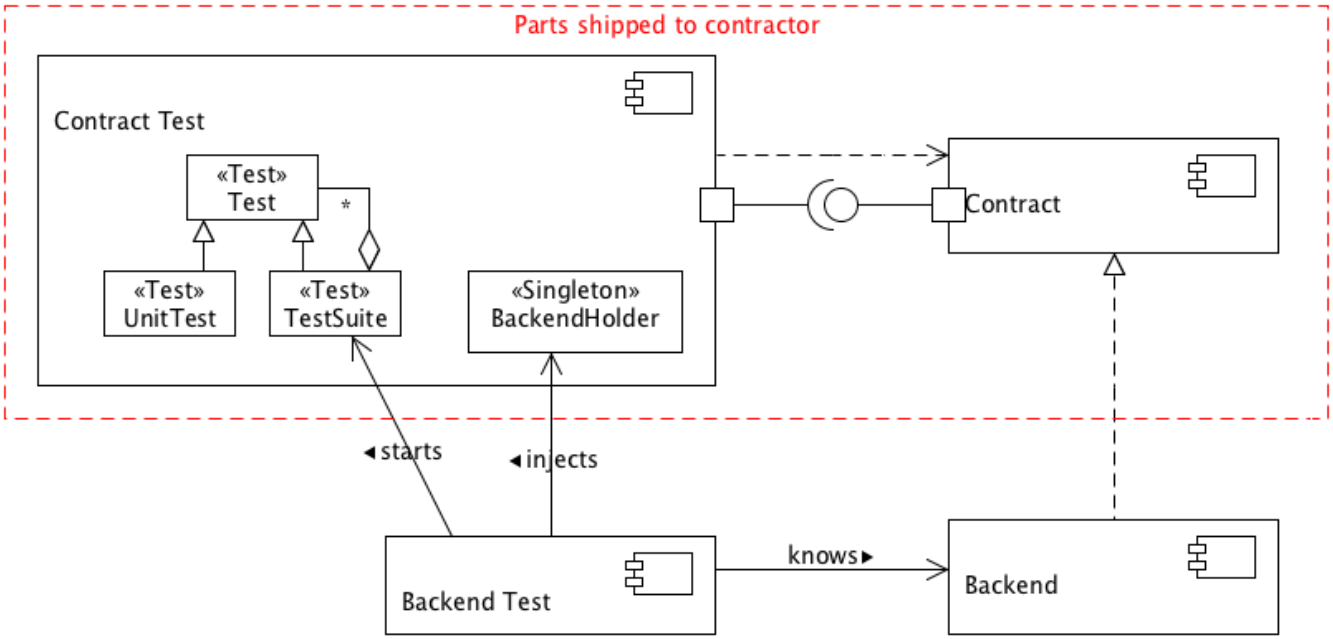
System operation contracts - Data Transfer objects



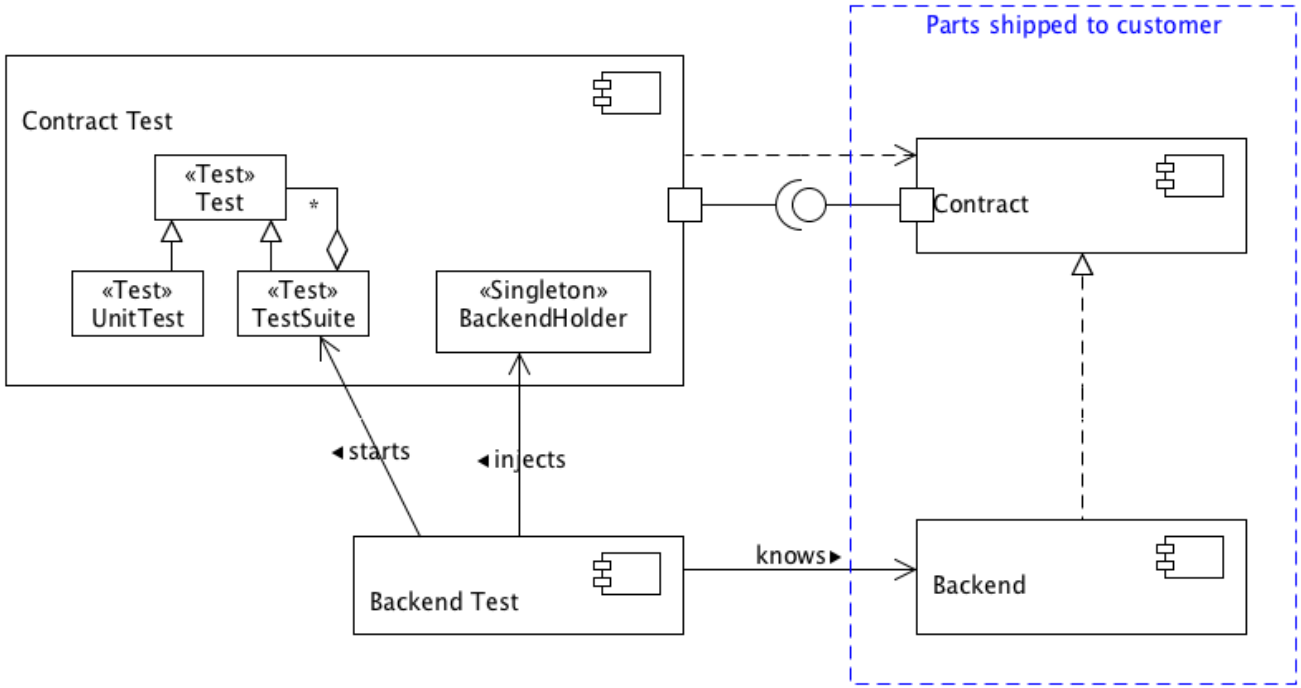
Verification



Verification



Verification



Dungeon game example

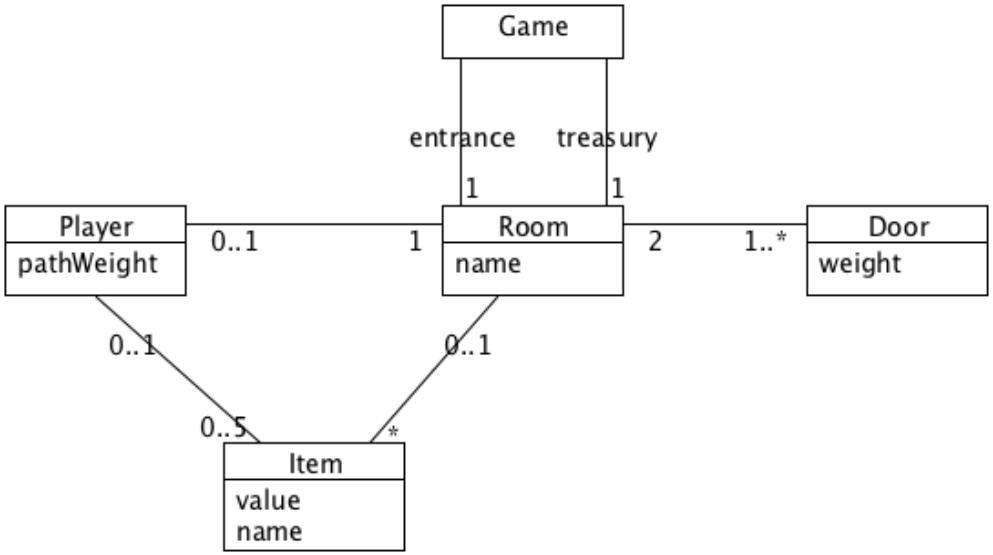
cphbusiness
COPENHAGEN BUSINESS ACADEMY

Requirements

We want a Dungeon game. The scenario of the game is a number of connected rooms or dungeons in a mountain. The player enters the mountain from the entrance room, and he/she should travel from dungeon to dungeon until he/she reaches the treasury room. All dungeon has doors that leads to at least one other dungeon. A dungeon can contain an unlimited number of items. Items have values. When a player is in the room he/she can see the items in the room, and he/she can see the doors leading from the room to other dungeons. The player can pick up and lay down items when he/she is in a room. But he/she can keep at most five items at a time. The quest is to reach the treasury room with the most expensive items through the shortest path. The game should run on a central server, and played throug a mobile phone connected to the server.

Dungeon game example

Logical data model

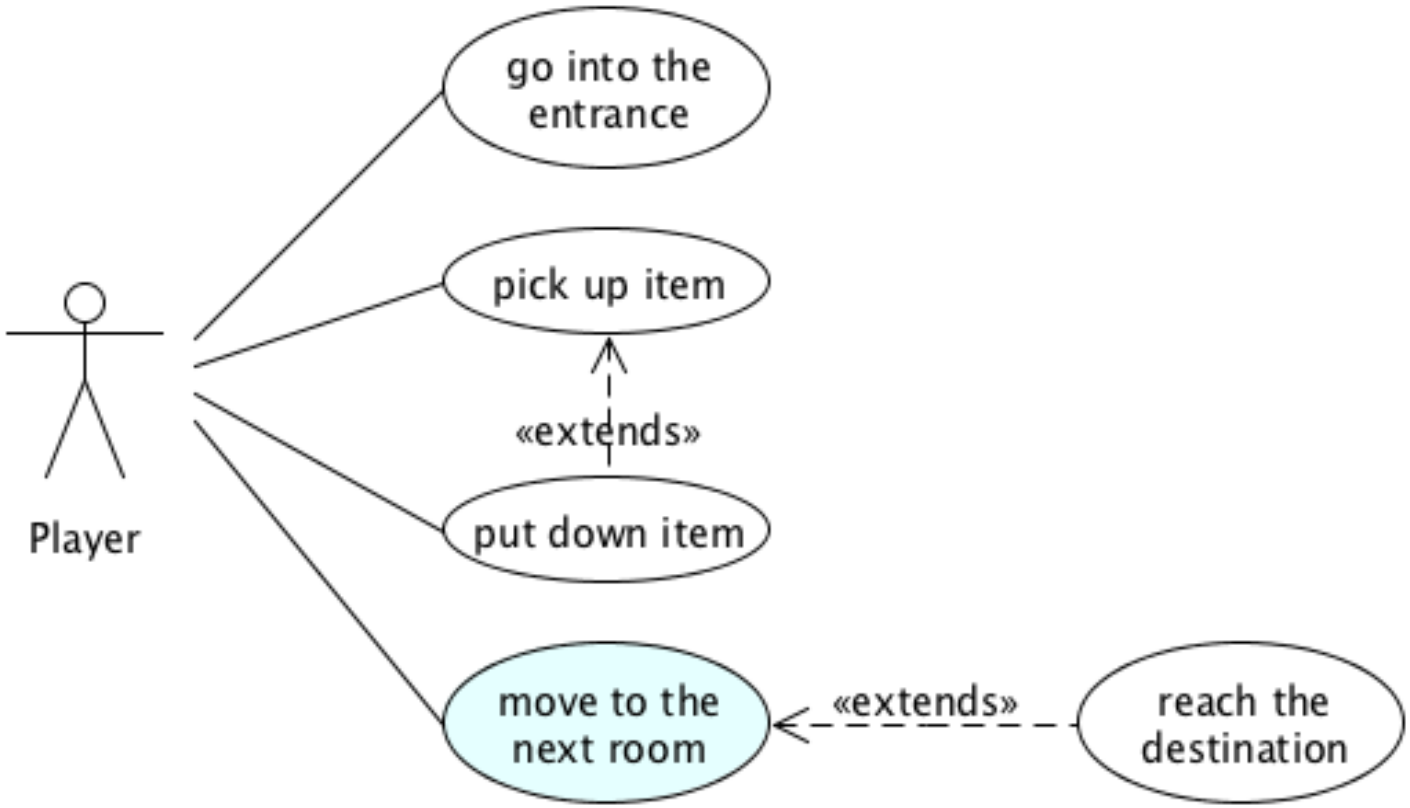


Again:

- Nouns from the requirements (glosary) are candidates
- What should be persisted
- Only entities
- No implementation details

Dungeon game example

Use Case Model - Use Case Diagram



Dungeon game example

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Use Case Model - Detailed Use Case Description...

- **Name** Move to the next room
- **Scope** System under design (SuD)
- **Level** Goal: Move to the next room
- **Primary Actor** Player
- **Precondition** The player is in a room
- **Main succes scenario** ...
- **Success guaratees** The player is in a new room
- **Extensions** Reach the destination if room is treasury room
- **Special Requirements** NONE

Dungeon game example

Use Case Model - . . . Detailed Use Case Description

- **Name** Move to the next room

. . .

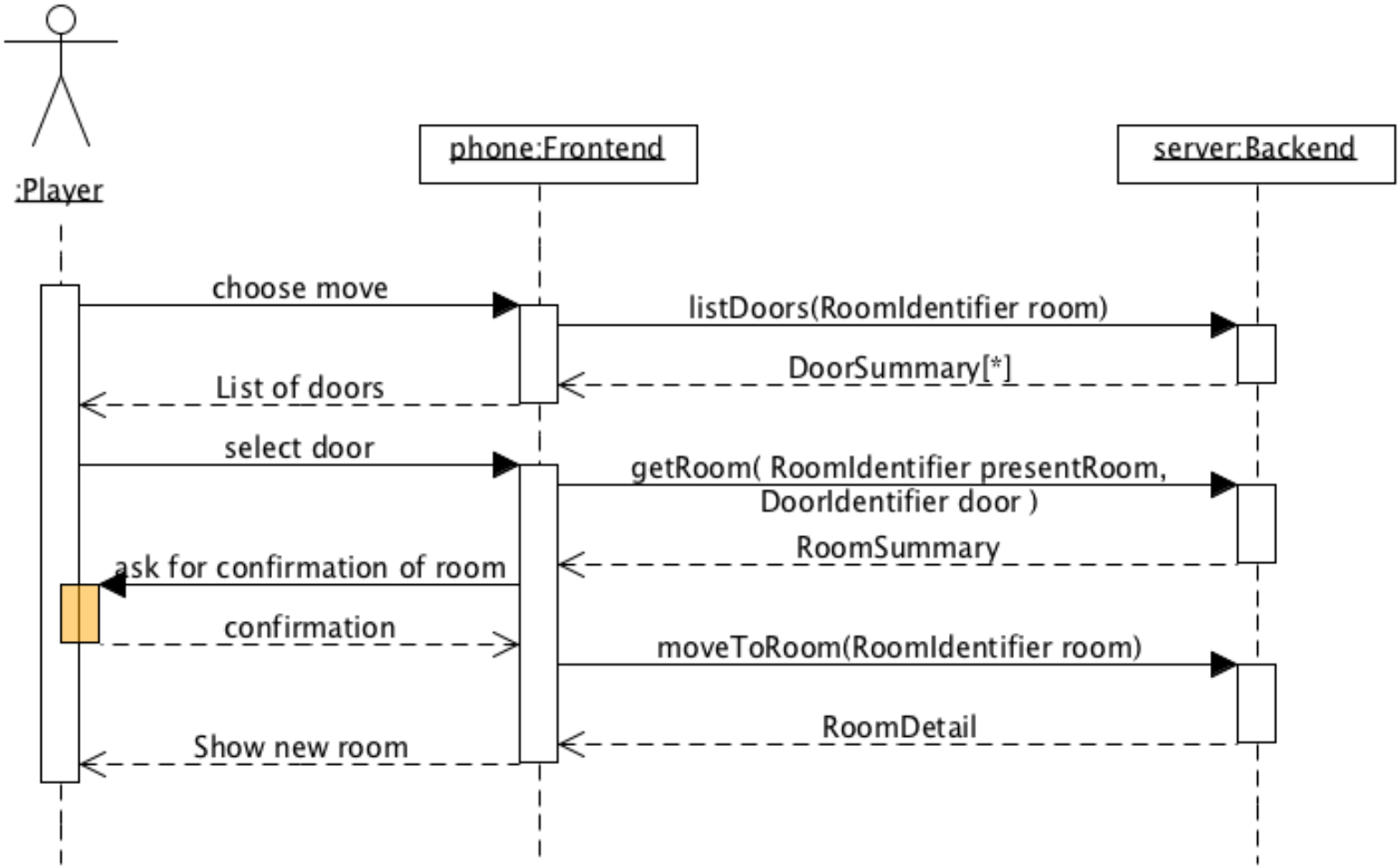
- **Main succes scenario**

1. Player chooses “move”
2. System shows a list with all doors to other rooms
3. Player selects the door he/she wants to move through
4. System shows the room name, and asks the player to confirm
5. Player confirms the selection of door
6. System moves the player to the room behind the selected door

. . .

Dungeon game example

Use Case Model - System Sequence Diagram



Dungeon game example

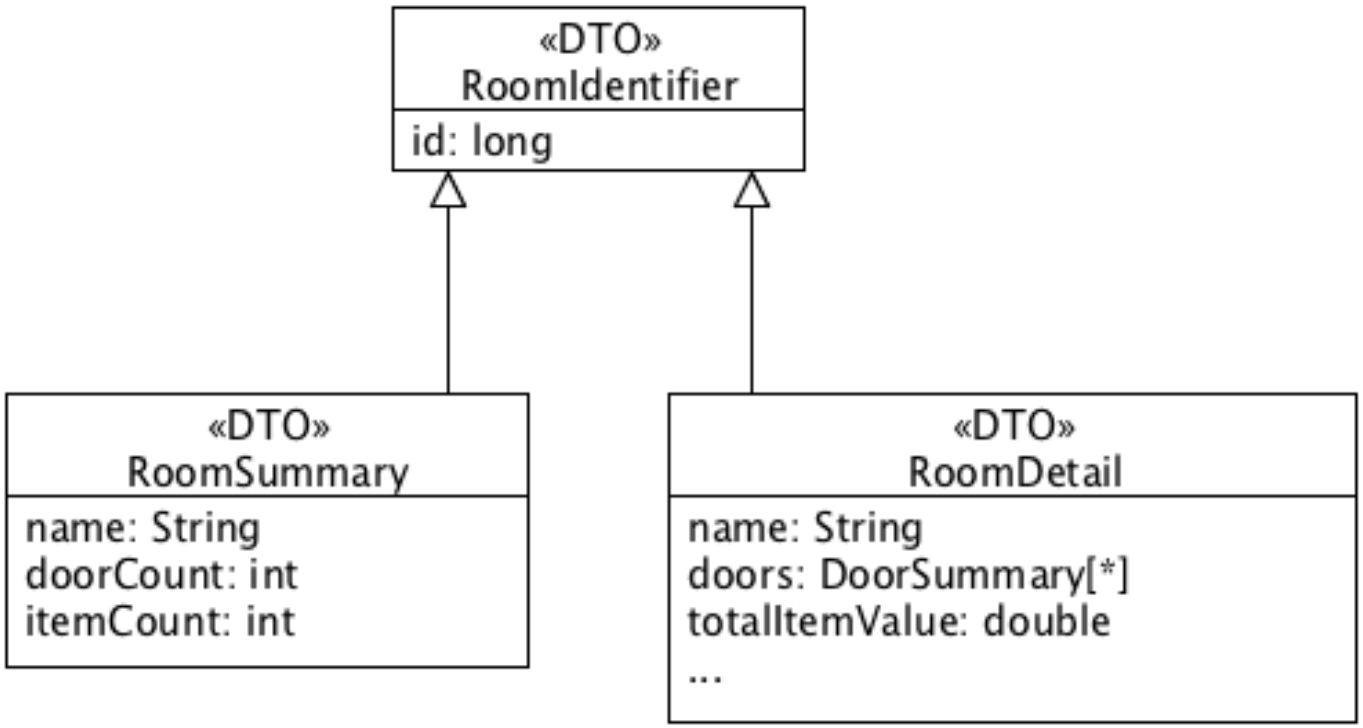
cphbusiness
COPENHAGEN BUSINESS ACADEMY

Communication model - "Remote" interface

```
@Remote
public interface DungeonManager {
  ...
  /**
   * List the doors leading from a given room.
   * @pre the room cannot be null and must exist.
   *    @throws NoSuchRoomException room doesn't exist.
   *    @param room the given room.
   * @post the doors in the given room is returned
   *    @return A collection of door summaries.
   */
  Collection<DoorSummary> listDoors(
      RoomIdentifier room
      );
  RoomSummary getRoom(
      RoomIdentifier room, DoorIdentifier door
      );
  RoomDetail moveToRoom(RoomIdentifier room);
  }
```

Dungeon game example

Communication model - Data Transfer objects



Communication model - Data Transfer objects

```
public class RoomIdentifier implements Serializable {
  private long id;

  public RoomIdentifier(long id) {
    this.id = id;
    }

  public long getId() { return id; }
  }
```

Dungeon game example

Communication model - Data Transfer objects

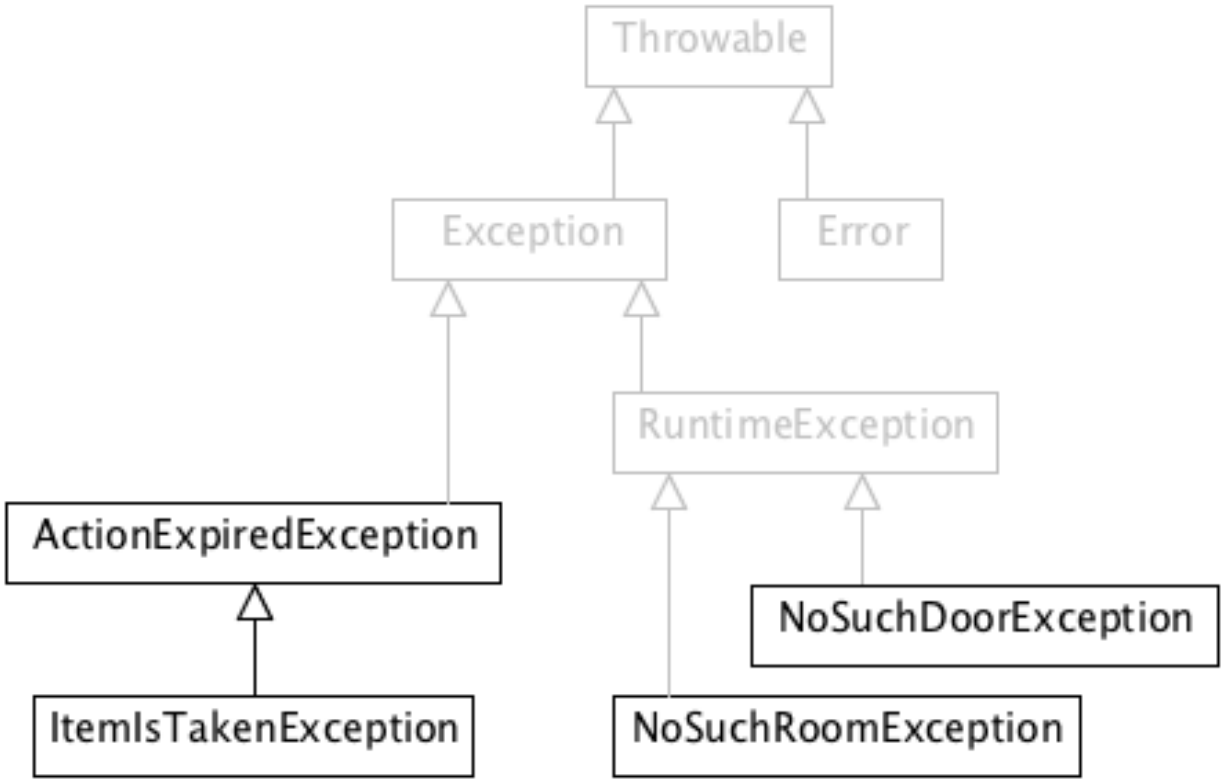
```
public class RoomSummary extends RoomIdentifier {
  private String name;
  private int doorCount;
  private int itemCount;

  public RoomSummary(
      long id, String name,
      int doorCount, int itemCount
      ) {
    super(id);
    this.name = name;
    this.doorCount = doorCount;
    this.itemCount = itemCount;
    }

  public long getName() { return name; }
  public long getDoorCount() { return doorCount; }
  public long getItemCount() { return itemCount; }
  }
```

Dungeon game example

Communication model - Exception Transfer objects



Dungeon game example

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Communication model - Exception Transfer objects

```
public class ActionExpiredException
             extends Exception {

  public ActionExpiredException(String message) {
    super(message);
    }

  }
```

```
public class NoSuchRoomException
             extends RuntimeException {

  public NoSuchRoomException(String message) {
    super(message);
    }

  }
```