The Toolbox 工具箱

Distributed development

Lars Bogetoft - Anders Kalhauge
拉尔斯 - 安德斯

Spring 2013

Agenda

- Presentation - **who** we are
- Origin - **where** we come from
- Global distributed software development - **why** we are here
- Evolution of the tool box
- The tool box

cphbusiness
COPENHAGEN BUSINESS ACADEMY

拉尔斯 桦木生长的地方

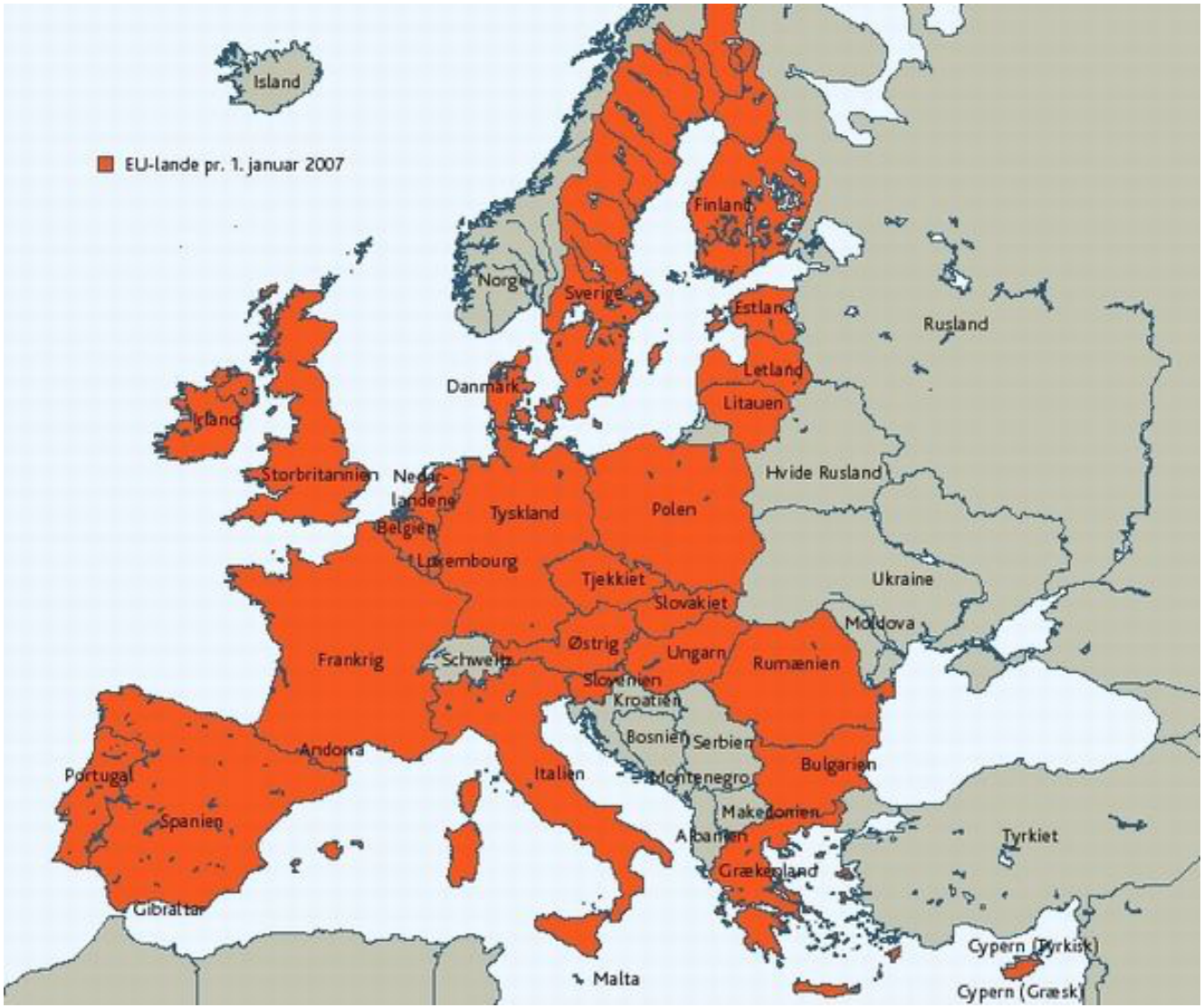
- Master in Computer science and public administration
- Teaching for 25 years at university level
- Main interests
  - Systems Development
  - Teaching approaches
  - Global distributed software development
- Also teach at RUC, DIKU, and DTU

cphbusiness
COPENHAGEN BUSINESS ACADEMY

安德斯 甘蓝生长的田地

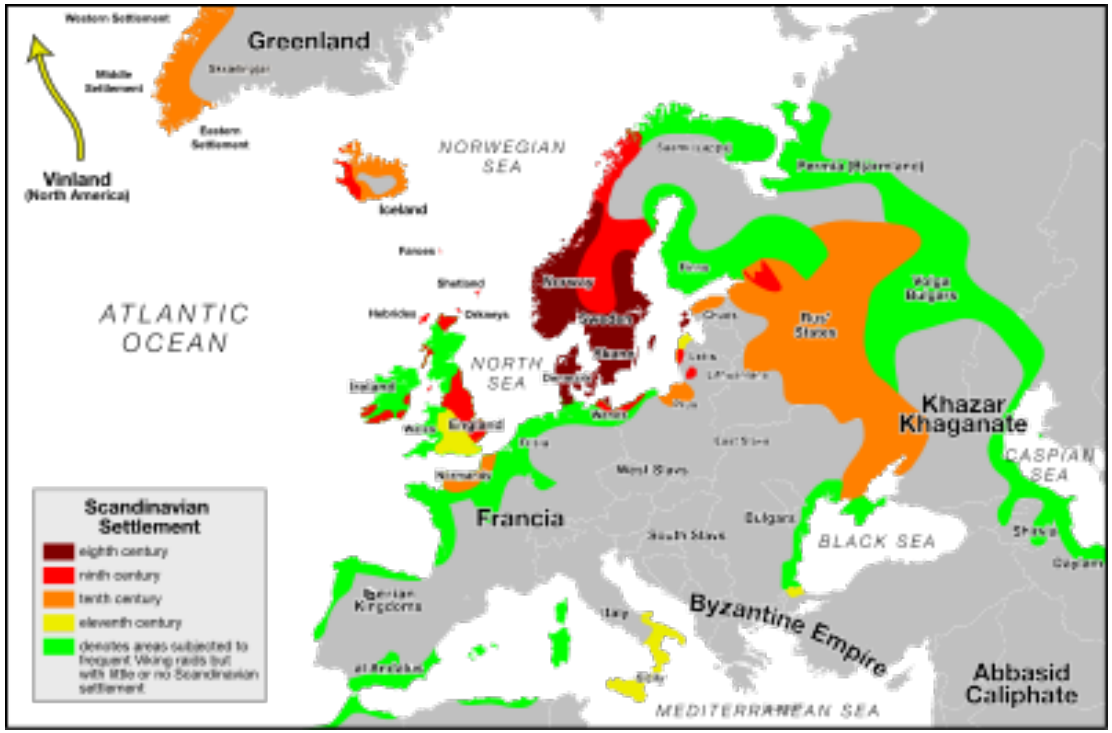
- 20 years experience as IT consultant in the private sector
- 10 years teaching computer science for students and private companies
- Main interests
  - Programming
  - Development of large scale systems
  - Software architecture

Where do we come from?
Denmark in Europe



Where do we come from?
Copenhagen in Denmark

Denmark in the viking ages



Our academy

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Copenhagen Business Academy is Denmark's largest business academy and consists of five campuses in Greater Copenhagen and Northern Zealand. Copenhagen Business Academy offers professional higher education programmes in which theory and practical skills always go together, and where the subjects taught can be seen in their proper perspective.

Our history extends back for more than a hundred years. Not in terms of the same educational programmes, but in terms of our focus on giving students the competences that are in demand in the world of business.

- Globally Distributed Software Development (GDSD):
  - Different development groups are responsible of different parts of the developed software system
  - The development groups are placed in different countries around the world
- Software Development is hard
- Distributed Software Development is harder
- Globally Distributed Software Development is . . .

- To save money
  - Lower wages in eastern and developing countries
- To get access to IT professionals
  - Negative growth in population in the west $\Rightarrow$ Fewer workers
  - Negative growth in IT students in the west
  - Growth of IT students in eastern and developing countries

- The IT systems are vital to our society
  - If we can't maintain and further develop our IT systems it is a threat to our existence
- We have a problem with "producing" enough IT professionals
  - The government and EU are making huge campaigns to recruit new IT students
- As a small country it is not profitable to have specialized IT professionals
  - We have to look abroad to find specialists

- Being an educational institution, we have an obligation to provide relevant skills to our students.
- Relevant skills will include ability to navigate in a globalized world also in the future.
- In the future globalization will demand IT professionals who can participate in Global Distributed Software Development

Evolution of the toolbox

cphbusiness
COPENHAGEN BUSINESS ACADEMY

2002 The curriculum for the 3rd semester on the Computer Science program is reformed, and the focus is now distributed systems.

2002-2008 Different models for the cooperation between the student groups are tried out with various success. The basic version of the toolbox is developed.

2008-2012 In cooperation with NENU, the toolbox is testet and improved dramatically in a true globalized setup.

2012- Contracting for distributed system is moved from 3rd to 6th semester, and the cooperation to a joined course.

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

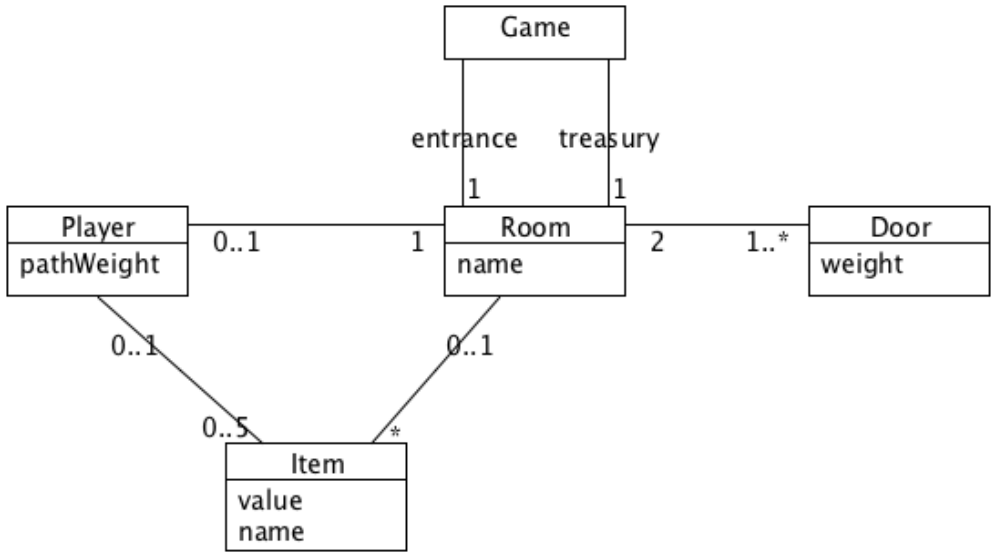
An example requirement
Dungeon Game

cphbusiness
COPENHAGEN BUSINESS ACADEMY

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.

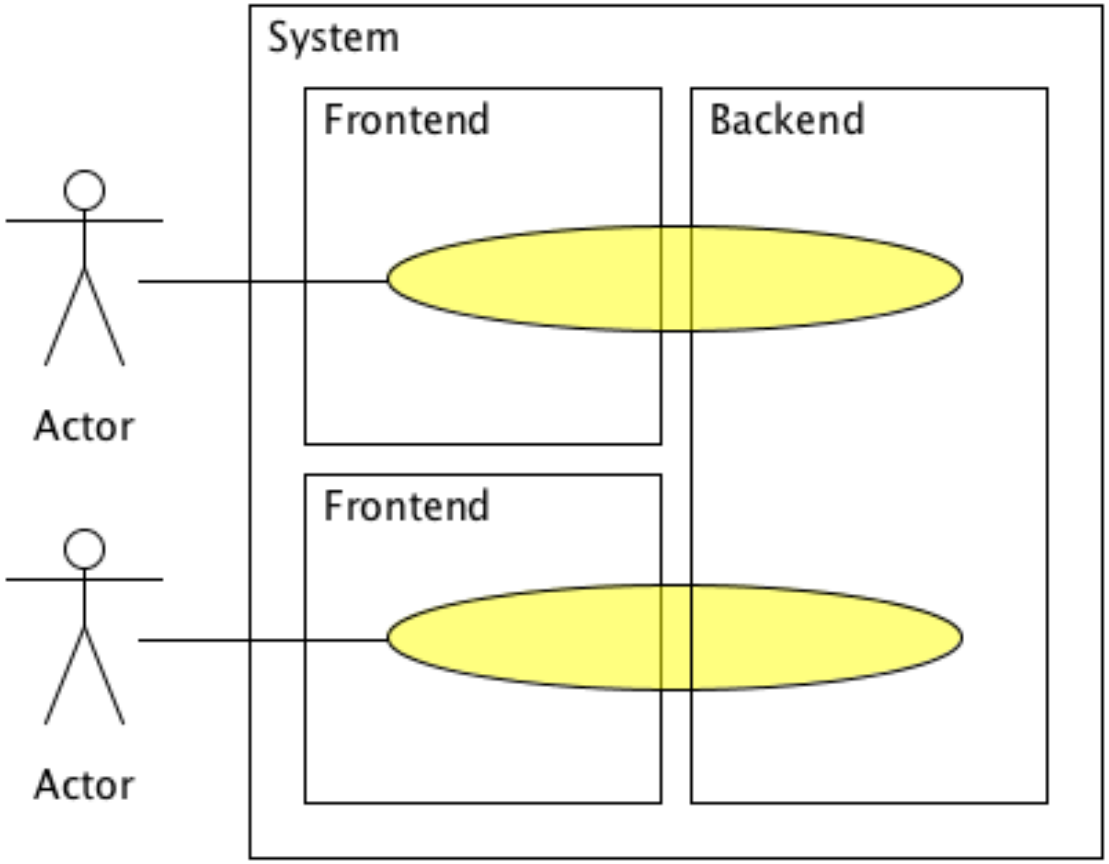
- It models the system state.
- Expresses valid pre- and postcondition states.
- Expresses possible system state changes.

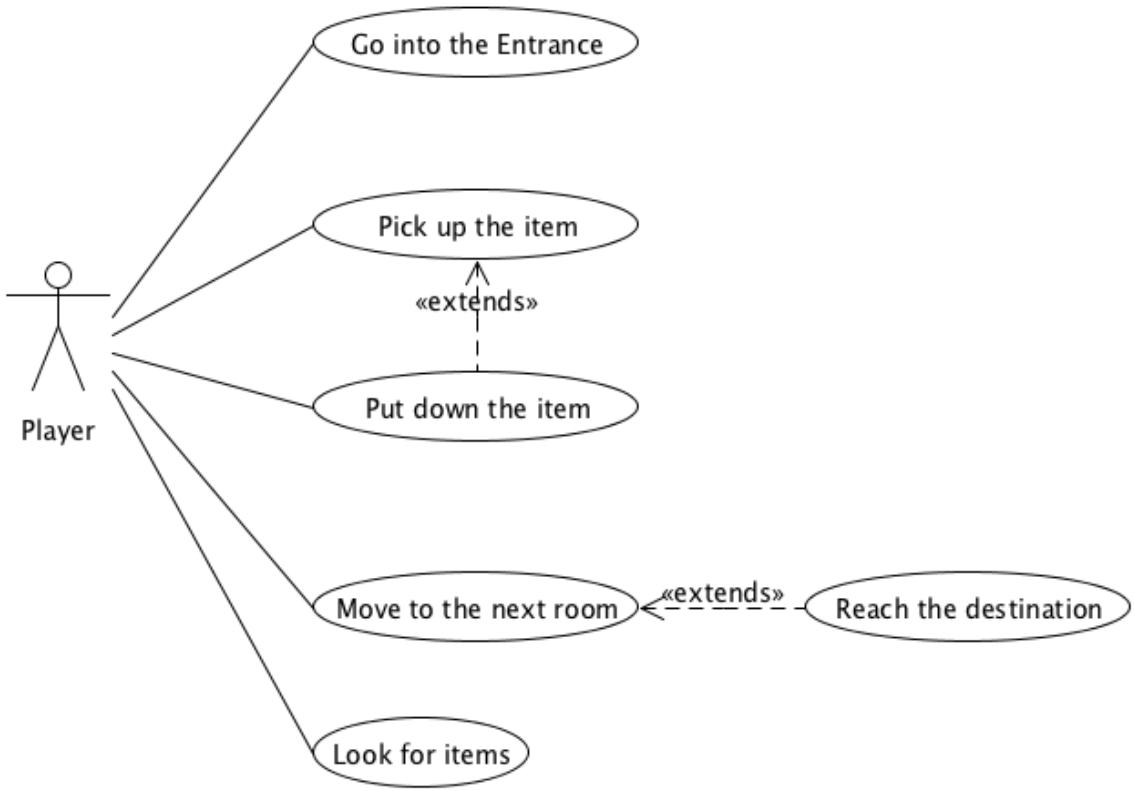
Logical data model
An example



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

Use Case Model
Subsystems and Use cases




Go into the Entrance
Pick up the item
«extends»
Player
Put down the item
Move to the next room
«extends»
Reach the destination
Look for items

Detailed Use Case Description 1

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

Detailed Use Case Description 2

cphbusiness
COPENHAGEN BUSINESS ACADEMY

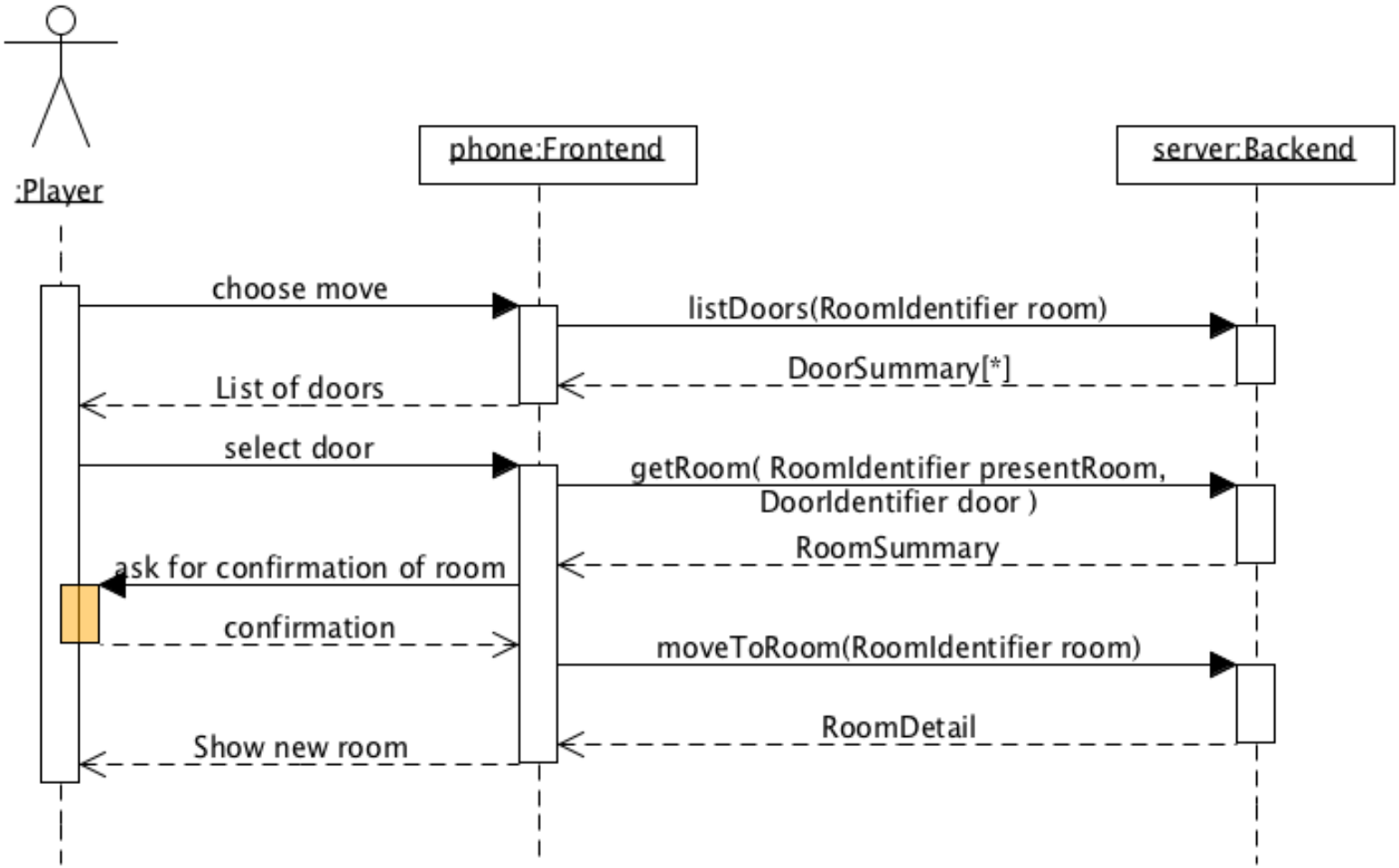
Name Move to the next room

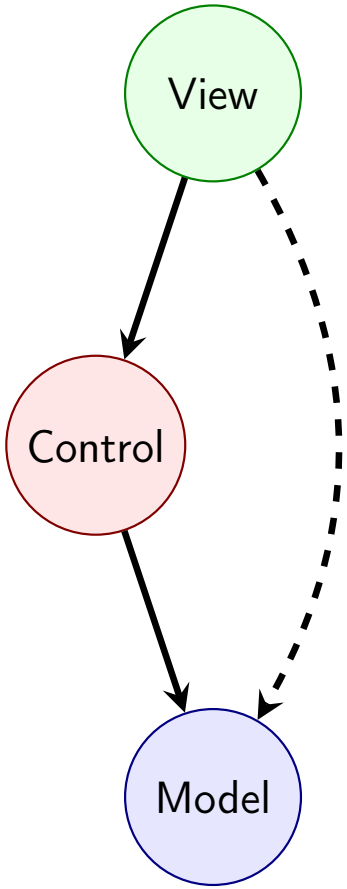
...

Main succes scenario

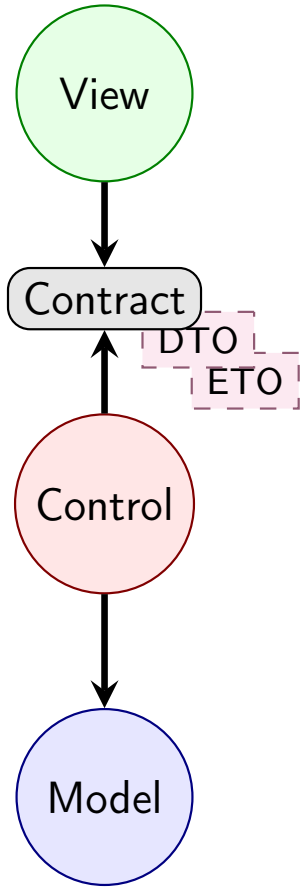
1. Player chooses “move”
2. System shows a list with all doors to other rooms
3. Player selects the door he/she wants to move through
4. System shows the room name, and asks the player to confirm
5. Player confirms the selection of door
6. System moves the player to the room behind the selected door

...


:Player
phone:Frontend
server:Backend
choose move
listDoors(RoomIdentifier room)
DoorSummary[*]
List of doors
select door
getRoom(RoomIdentifier presentRoom,
DoorIdentifier door)
RoomSummary
ask for confirmation of room
confirmation
moveToRoom(RoomIdentifier room)
RoomDetail
Show new room



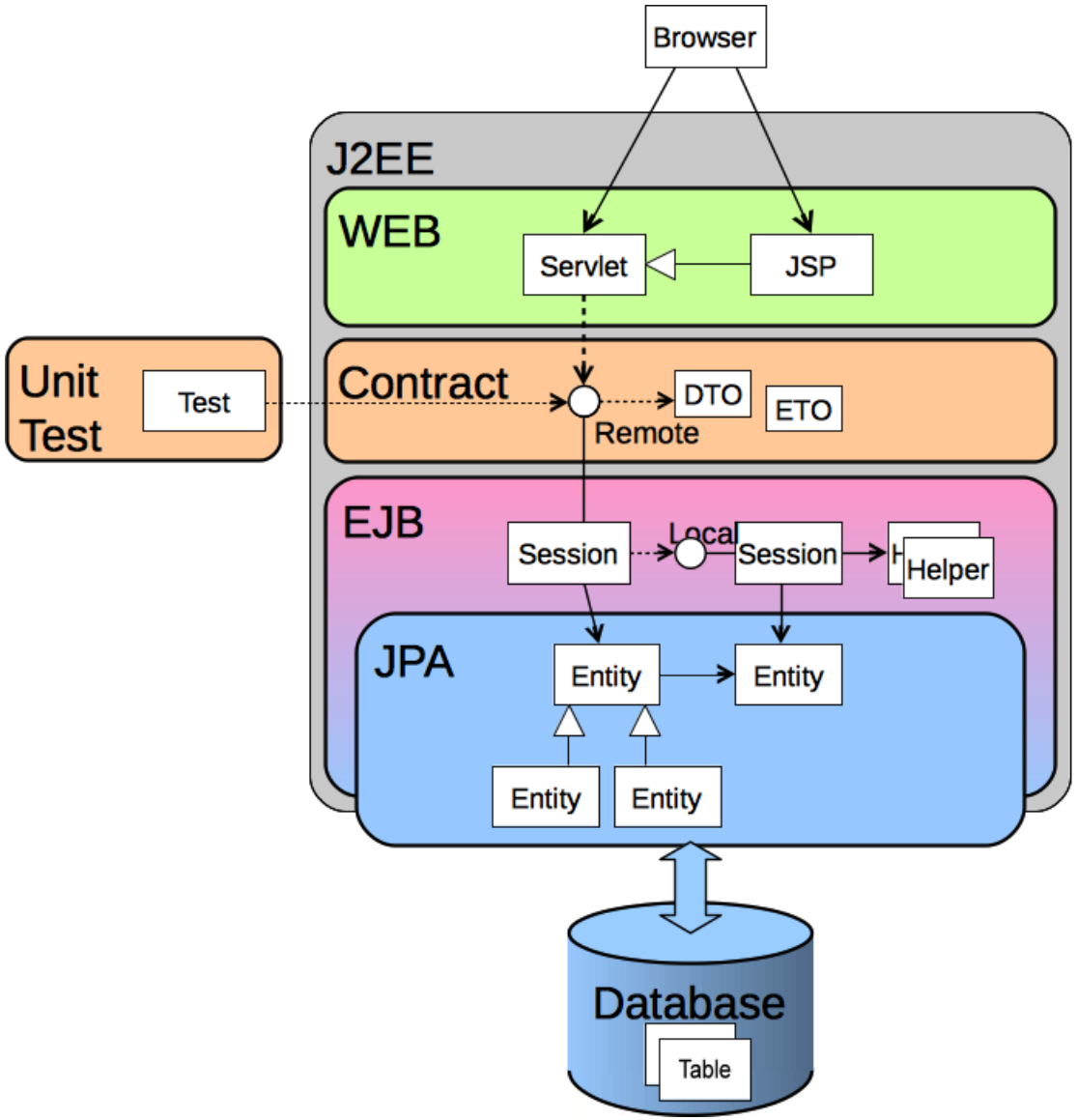
(a)



(b)

Communication model

System operation contracts - MVC pattern in EJB



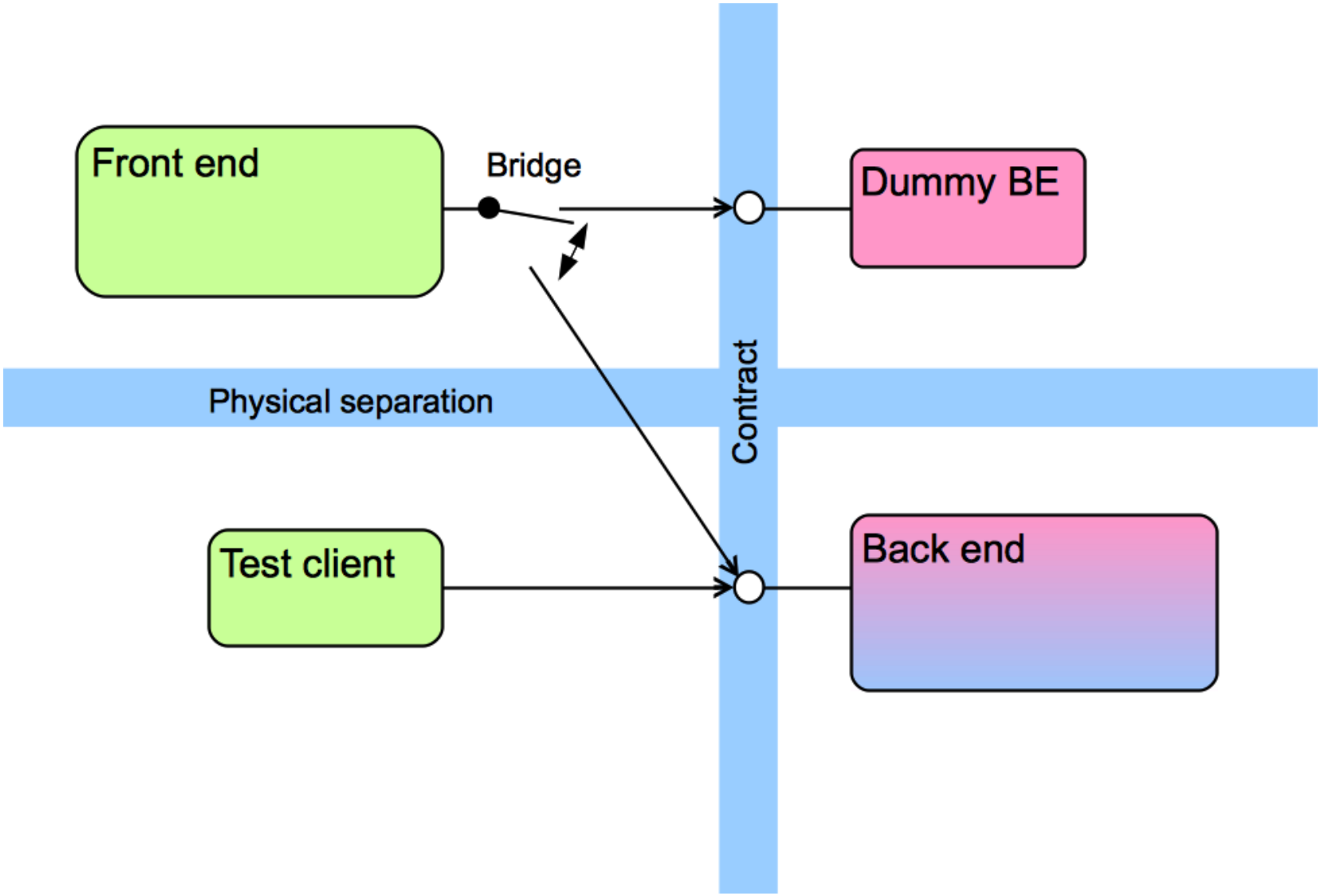
GUI design

Test cases

Use cases

Domain model

Project setup with bridge



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 Façade in the "backend".
  - Changes to the backend code or to the interface will have less impact.
- Reference the interface from a Bridge in the "frontend".
  - Change of backend can be done with practically no code changes in "frontend"

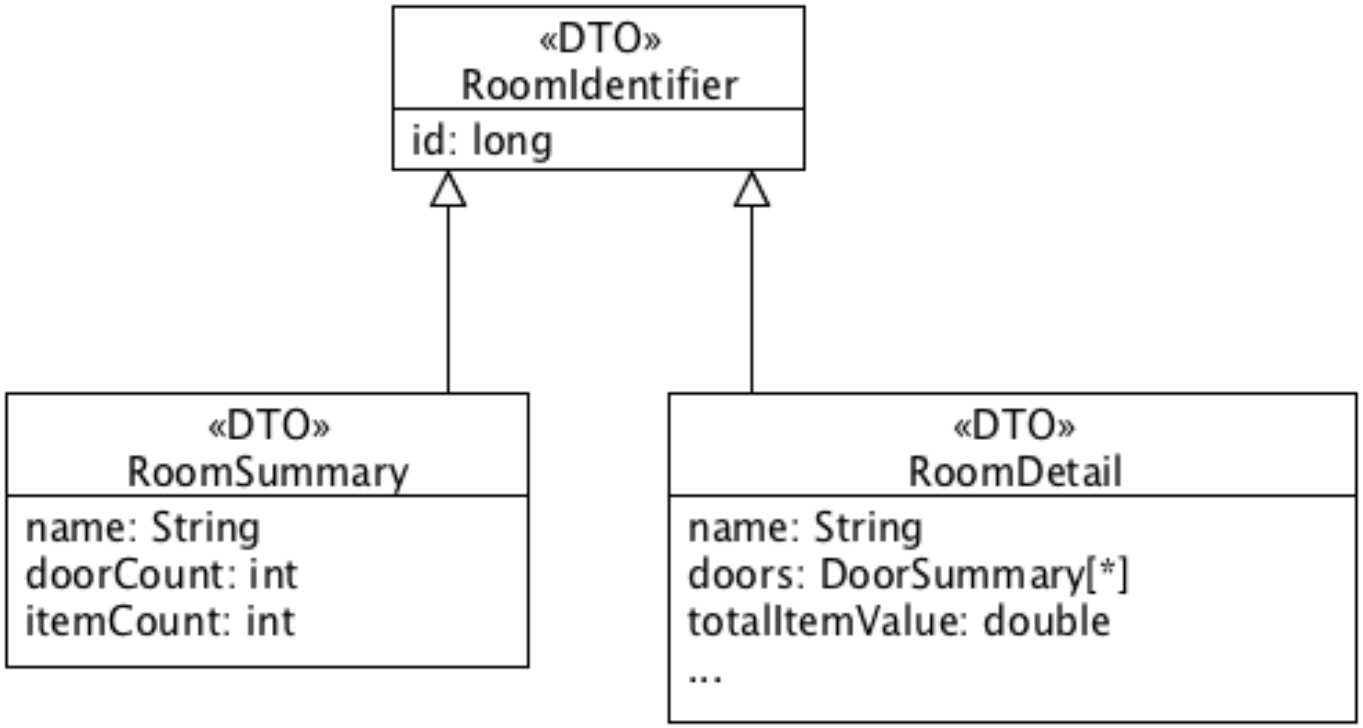
```
@Remote
public interface DungeonManager {
  ...
  /**
   * List the doors leading from a given room.
   * @pre the room cannot be null and must exist.
   *    @throws NoSuchRoomException room doesn't exist.
   *    @param room the given room.
   * @post the doors in the given room is returned
  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

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

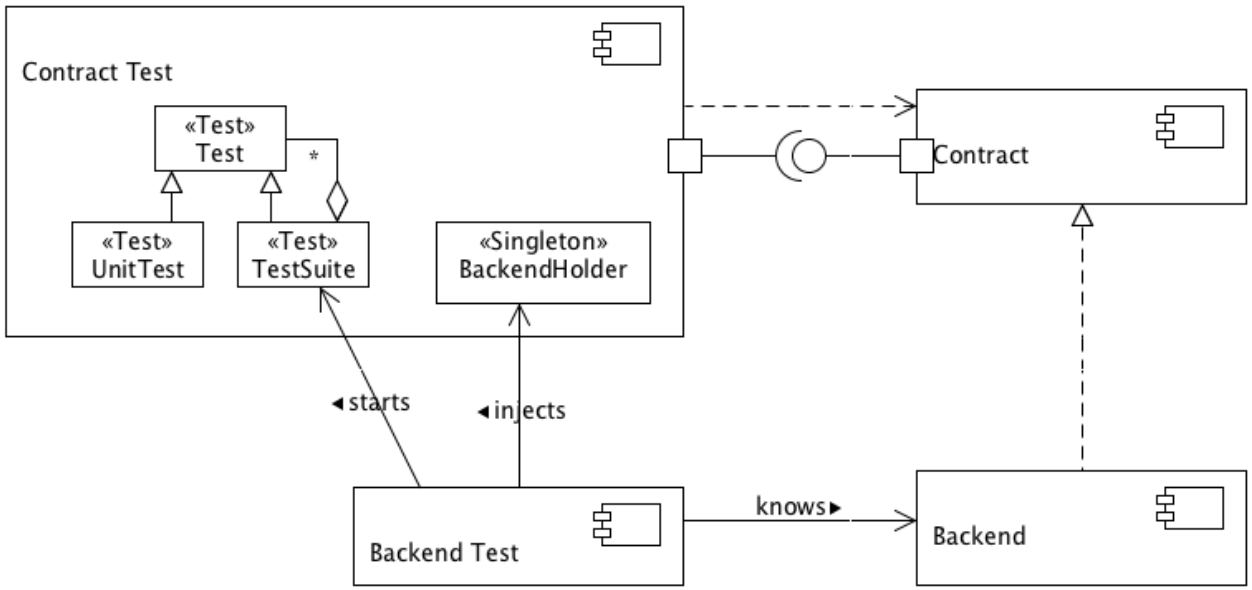
Exceptions are as valid, even less happy, return values from operations.

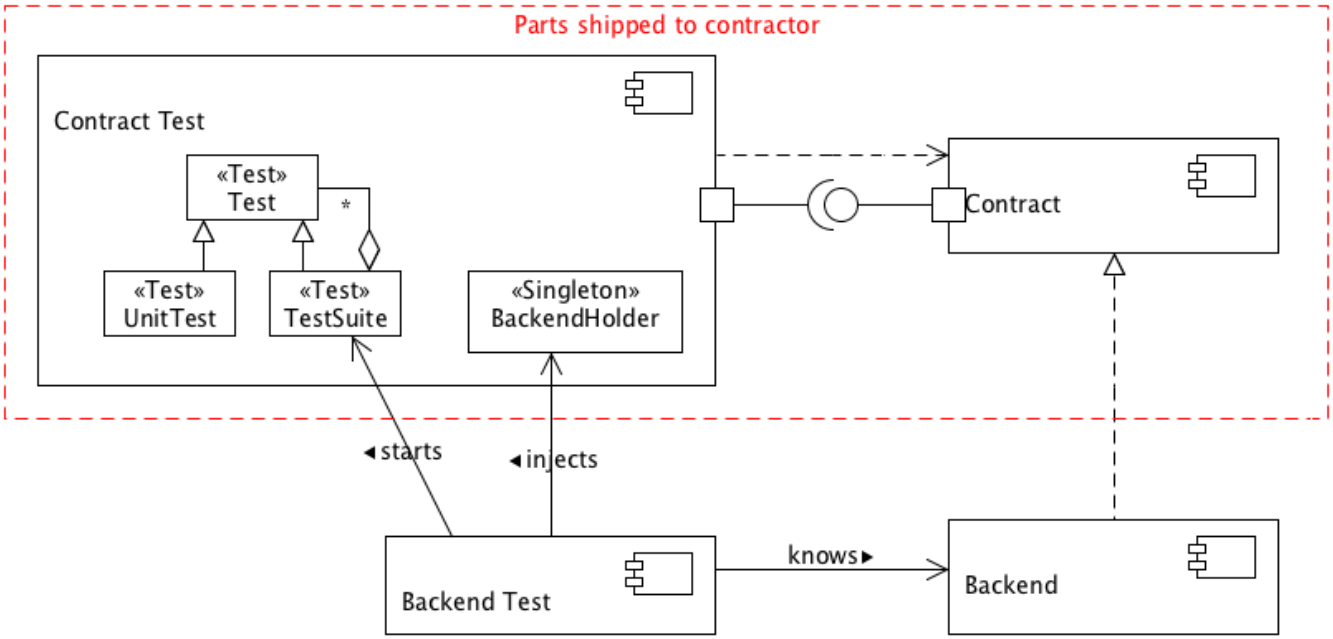
- User friendly - return only relevant information.
  - Preconditions: What precondition was violated (unchecked).
  - Postconditions: What went wrong (checked).
- Encapsulation
  - by hiding actual implementation
  - **revealing errors and their precise cause, is pleasing hackers**
- Serializable -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

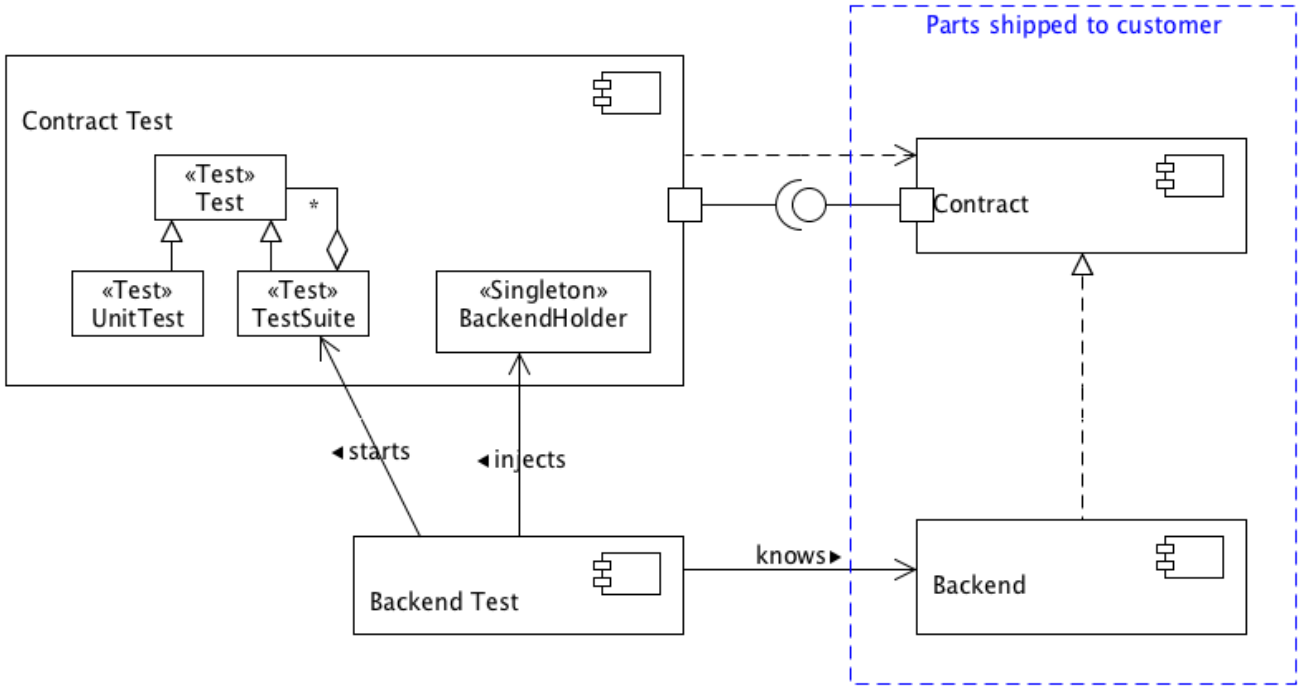

«DTO»
RoomIdentifier
id: long
«DTO»
RoomSummary
name: String
doorCount: int
itemCount: int
«DTO»
RoomDetail
name: String
doors: DoorSummary[*]
totalItemValue: double
...

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

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

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

```
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; }
  }
```


Contract Test
«Test»
Test
*
«Test»
UnitTest
«Test»
TestSuite
«Singleton»
BackendHolder
Contract
starts
injects
Backend Test
knows
Backend


Parts shipped to contractor
Contract Test
«Test»
Test
*
«Test»
UnitTest
«Test»
TestSuite
«Singleton»
BackendHolder
Contract
starts
injects
Backend Test
knows
Backend


Parts shipped to customer
Contract Test
«Test»
Test
*
«Test»
UnitTest
«Test»
TestSuite
«Singleton»
BackendHolder
Contract
starts
injects
knows
Backend Test
Backend

Summing up

- Globalization in Software Development is only in its beginning
- Only experiments in practice will find feasable solutions
- NENU and cphbusiness are important contributors
- Simple tools can be very powerful in the right combination
- We look forward to the next five years of cooperation with NENU