Programming Embedded Systems

***Project overview***

Monday Mar 28, 2011

Philipp Rümmer
Uppsala University
Philipp.Ruemmer@it.uu.se

## What happened so far

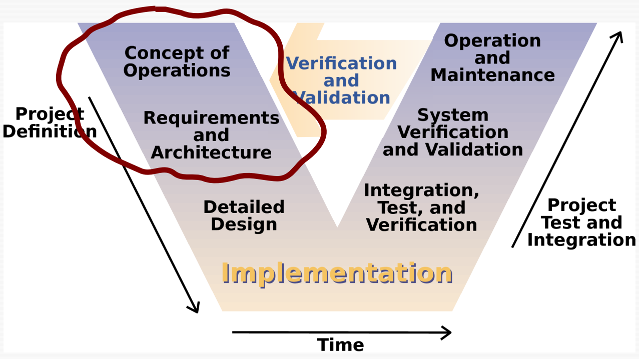
- Decided about groups
- Thought/decided about project topics

## Plan for today

- Discuss the individual project topics
- On proposal writing (“pre-study”)

- (borrowed some slides from Rogardt Heldal)

## Development process


Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

## Contents of pre-study

- **Analysis**
  - How does the world look like?
    (in which our system has to operate)
  - Possibly: domain model, vocabulary
- **Requirements engineering**
  - What is the system supposed to do?
  - Use cases, requirements
- **Architecture**
  - Hardware + software components

## Required here

- **Just a short, textual project proposal (2-3 pages)**
- But: should cover
  analysis + requirements + architecture
- To be finished by **April 6th**
  and uploaded to student portal

## Analysis in the large

- For our purposes:
  textual description of project setting is sufficient
- Common technique for larger projects:
  **concepts + domain models**,
  formulated in UML
  (to be taken with a grain of salt)

- Really: more topic of Bengt's course

## Goal of analysis phase

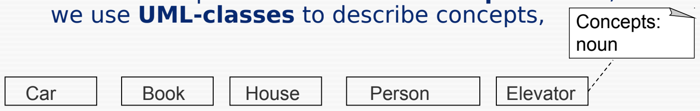
- Understand the problem (domain)
- Eliminate ambiguities by creating a well-defined vocabulary

## Concepts

We use **concepts** to denote things in the real world.

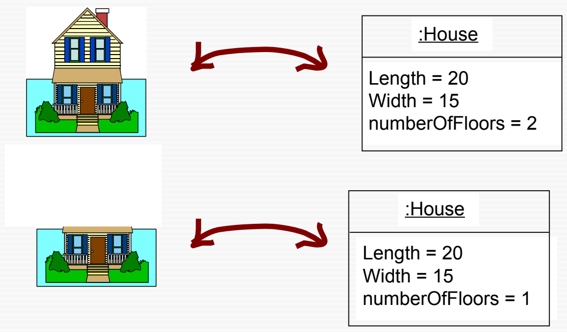
For instance:
Car, Book, House, Person and Elevator.

There is no special notation for **concepts** in UML, so we use **UML-classes** to describe concepts,



A concept is a description of a group of concept-instances with the same properties.

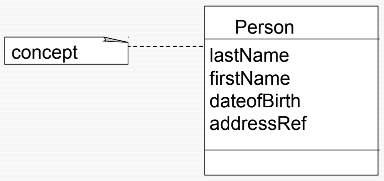
## Example: Concept-instances



Real or imaginary object

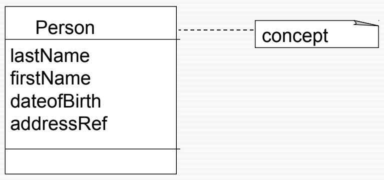
## Attributes

- Attributes describe properties of concepts.
- Attributes are needed to store the information that the concept-instance must remember.
- Candidate attributes are concepts which do not have an independent role, but rather fit as attributes in some of the other concepts.

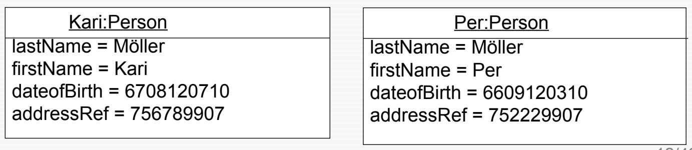


## Instances

Example:



Instances:



## The Domain Model

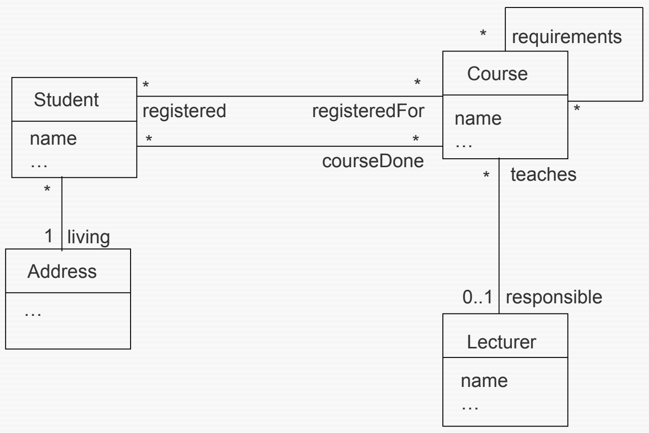
Construct a model of the problem (*problem domain model*).
I.e. not a model of the system/software.

- A collection of system-relevant concepts and their static interconnection.
- Often the model is presented just as a number of concepts (“Vocabulary”)

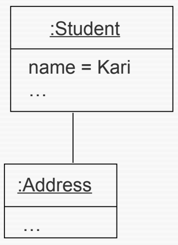
## The Domain model shows

- concept-names
- associations between concepts
- concept attributes
- etc

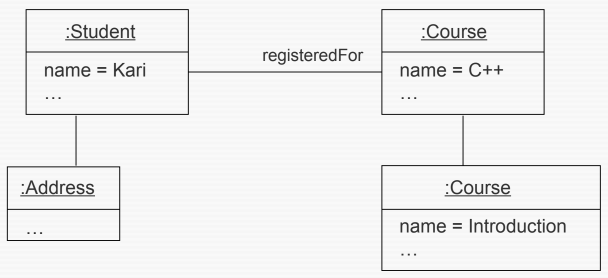
## Example: Domain model



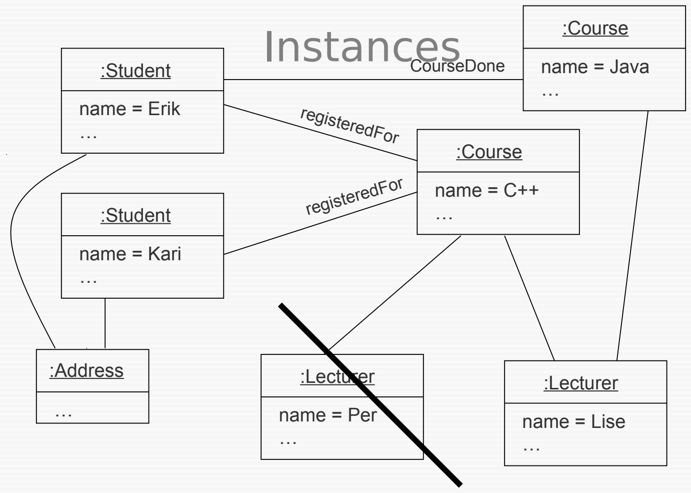
## Instances



## Instances



## Instances



## Requirements engineering

- Two main formalisms:
  - Use cases
  - Requirements (textual)

## Requirement analysis

- Often these kinds of requirements have to be identified (FURPS+):
  - Functionality
    - Features
    - Security
  - Usability
    - Auxiliary functions
    - Documentation
  - Reliability
    - Frequency of failure
    - Predictability

## Requirement analysis (2)

- Performance
  - Response times
- Supportability
  - Adaptability
  - Configurability
- ”+” represents further requirements/documents that are possible, e.g., implementation, user interface, licensing

## Functional requirements

- Use cases capture most functional requirements (details later)
- But: Some functionality can be ”hidden” in several/all use cases
  - For instance: Logging occurring events

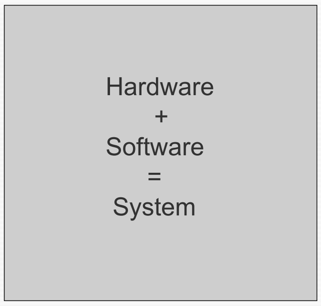
## Non-functional requirements

- Use cases are not suitable here, do not capture non-functional issues
- But: Use cases are a context to which non-functional requirements can be attached:
  - For instance: ”Dispensing money takes at most X sek” is added to use case ”Withdraw Money”
- Other requirements are more difficult, can’t be assigned naturally to particular use cases

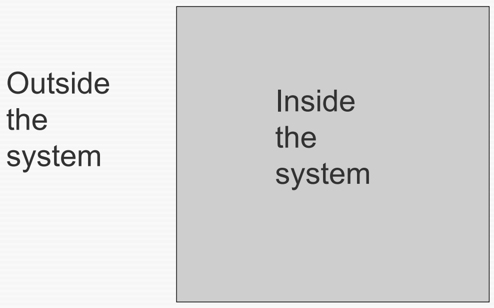
## Example

- Usability
  - ATM should be usable for visually impaired persons
  - ATM should be usable for colour blind persons
- Reliability
  - Frequence of failure
    - At most one failure per year (or per 10 sek)
  - Restart after an error
    - When restarting, account balance should be checked against bank to ensure right value (in case of unfinished transactions)

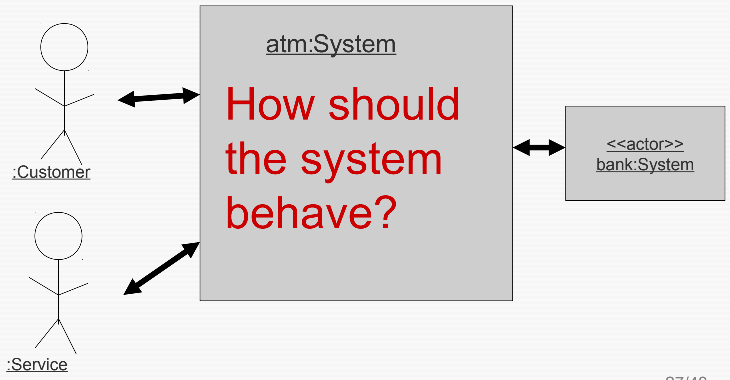
## System



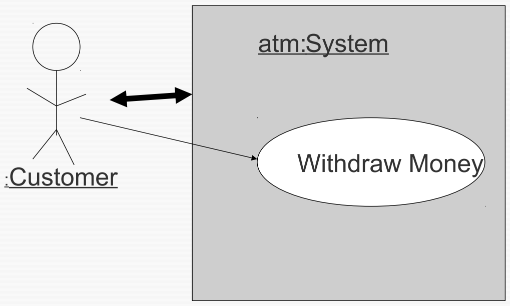
## System



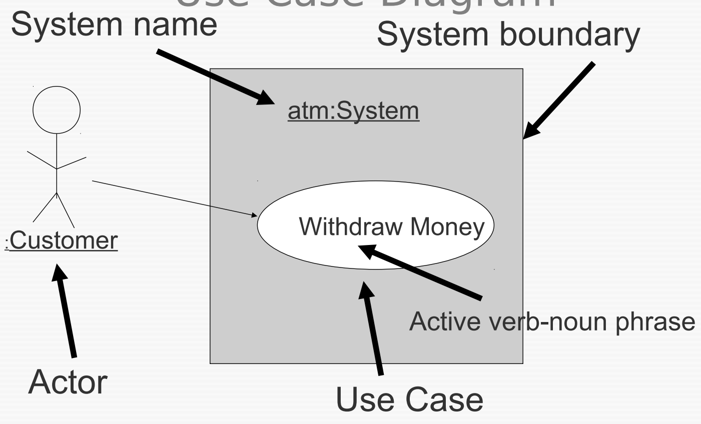
## ATM



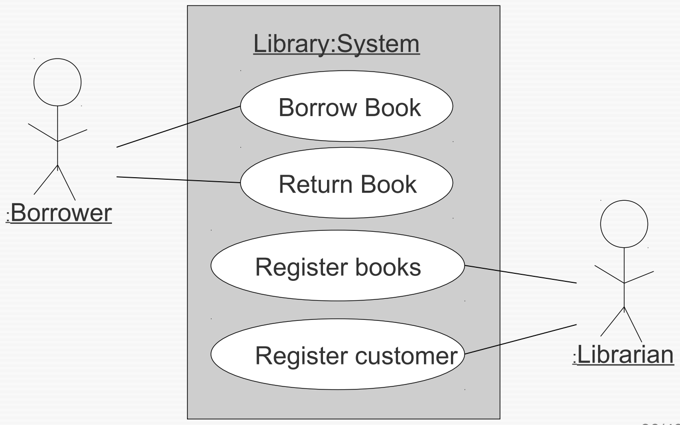
## ATM



## Use Case Diagram



## Solution: Library



## Brief Use Cases

- A short description of the use case, for example:
  - Name: Withdraw Money
  - Actor: Customer
  - Goal: Take out money from an account
  - Description: The customer identifies himself and requests an amount of money. The ATM gives out money if the customer has sufficient funds in his account.

## Brief Use Cases are Good

- These perfectly catch the informal behaviour of a system in an abstract way.
  - Use case name
  - Primary actor
  - Goal of the actor for this use case
  - Brief description
- Complete use case: a use case containing event flows (action steps).

## Complete Use Cases

- Name: Withdraw Money
- Actor: Customer
- Main Flow of Event:
  - ...
  - 7. User requests withdrawal of an amount of money
  - 8. System checks that the account balance is high
  - 9. System subtracts from account the amount taken out from
  - 10. System gives back card and dispenses cash
  - ...

## Action Steps

- Use cases without good action steps are worthless.
- The event flows are the important part of use cases

## Towards formal action steps

- When moving towards complete use cases one should make action steps more precise.
- Too formal: not understandable

$\updownarrow$

- Too informal: not understandable either

## Result in a given Goal

- The main concern is not only
  - to be a complete process
- But:
  - To result in a given goal
- The primary actor wants to achieve something which is of importance, for example:
  - Obtain money from an ATM
- This means often an observable result for the primary actor.

## Primary Actor

- Always an actor starts the use case: most often the primary actor
- There are also other types of actors:
  - Secondary actor
  - Helper actor
  - Time

## Withdraw Money

Only main flow:

- user identifies himself by a card
- system reads the bank ID and account number from card and validates them
- user authenticates by PIN
- system validates that PIN is correct
- user requests withdrawal of an amount of money
- system checks that the account balance is high enough
- system subtracts the requested amount of money from account balance
- system returns card and dispenses cash

## Alternative flows

- Most use cases do not have just one flow, but several alternative flows.
  - Another frequent behaviour of the system
  - Another possible behaviour of the system
  - An error case
- The alternative ways depend on the input given by the actor and the system state.

## Example: Alternative Flow

Fragment of the use case “Withdraw Money”

...

7. User requests withdrawal of an amount of money
8. System checks that the account balance is high enough
9. System subtracts from account the amount taken out from the ATM
10. System gives back card and dispenses cash

8-10a: Not enough money on account:

1. System does not change the account
2. System returns card

## Architectural design

- Hardware aspects
  - Which components are needed for system?
  - Communication?
- Software aspects
  - Which software components? (here: architectural level, not too detailed)
- Deployment aspects
  - Which software on which hardware components?

## Architectural design (2)

- Dedicated architectural specification languages exist
  - SysML
  - AADL
  - EAST-ADL
- Beyond scope of this course
  (project architectures will be rather simple as well)

## Next steps

- Short project proposal covering analysis + requirements + architecture
- To be finished by **April 6th** and uploaded to student portal