LTAT.05.003
Software Engineering

Lecture 04: Analysis

Fall 2017
ICS Day 2017 – Friday, 29 September

When? – 14:15

Where? – Paabel (Ülikooli 17)

What? –

To bring together students and academic/research staff
To introduce interesting research directions that are carried out in the institute
To network and find topics for Bachelor's and Master's theses

More info:
ICS Day 2017 – Friday, 29 September

When? – 14:15
Where? – Paabel (Ülikooli 17)
What? –
To bring together students and academic/research staff
To introduce interesting research directions that are carried out in the institute
To network and find topics for Bachelor's and Master's theses

More info:

Timetable for the event
14:15 - 14:30 Introductory lecture by prof. J.Vilo (in the atrium)
14:35 - 15:00 1st round of research groups' introductions
15:05 - 15:30 2nd round of research groups' introductions
15:35 - 16:00 3rd round of research groups' introductions
16:00 Open doors for questions and discussions
Labs Next Week

• Assessment of your Homework 2 progress

• Each project team must attend with all its members present

• If not all team members present, penalty applies
Labs This Week

Lab 2 Assignment:

1. Project Planning
   - Refinement of requirements into Tasks
   - Prioritization of Tasks (P1, P2, P3, P4)
   - Responsibility assignment
   - Effort estimation

2. Five Use Case Descriptions

3. (Initial) Domain Model
Schedule of Lectures (Tentative)

Week 01: Introduction to SE
Week 02: Requirements Engineering I
Week 03: Requirements Engineering II
Week 04: Analysis
Week 05: Development Infrastructure I
Week 06: Development Infrastructure II
Week 07: Architecture and Design
Week 08: Refactoring
Week 09: Verification and Validation I
Week 10: Crowdsourced Testing

Week 11: Continuous Development and Integration
Week 12: Agile/Lean Methods
Week 13: Software Craftsmanship
Week 14: Course wrap-up, review and exam preparation
Week 15: no lecture
Acknowledgement

Textbooks/Slides:

Structure of Lecture 04

• Preliminaries and Context
  • OO Development Background
  • UML: UC Diagram, Class Diagram, Sequence Diagram
  • Analysis versus Design
• Domain Analysis and Modelling Example
  • Identifying Concepts (Responsibilities)
  • Attributes
  • Associations
• Domain Analysis and Modelling Exercise
Object-Oriented (OO) Analysis

- **OO analysis is *claimed to be* ‘natural’**
  - When a system evolves, the functions it performs need to be changed more often than the objects on which they operate
  - A model based on objects (rather than functions) will be more stable over time
  - Hence the claim that object-oriented designs are more maintainable
Nearly anything can be an object...

External Entities ...
... that interact with the system being modeled: people, devices, other systems

Things ...
... that are part of the domain being modeled: reports, displays, signals, etc.

Occurrences or Events ...
... that occur in the context of the system: transfer of resources, a control action, etc.

Organisational Units ...
... that are relevant to the application: division, group, team, etc.

Places ...
... that establish the context of the problem being modeled: manufacturing floor, loading dock, etc.

Some things cannot be objects:

  procedures (e.g. print, invert, etc)
  attributes (e.g. blue, 50Mb, etc)
Object Example - ATM

Object:
ATM machine

method-1:
Accept card

method-2:
Read code

method-3:
Take selection
Object Interface

- **Interface** defines method “signatures”

- Method signature: name, parameters, parameter types, return type

Object hides its state (=attributes). The attributes are accessible only through the interface.
Objects send **messages** by calling methods

- **Client object**: sends message and asks for service
- **Server object**: provides “service” and returns result
Representation Styles

- Natural language (plus supporting tables and graphs)
- Structured natural language / Scenarios
  - e.g., use case descriptions, user stories, CRC cards, ...
- Semi-formal notations
  - e.g., UML diagrams (use case diagrams, class diagrams, state machine diagram, sequence diagram, etc.)
- Formal notations (with formal semantics)
  - e.g., abstract model-based (VDM, Larch, B, ...) or algebraic (OBJ, SBJ, ACT-ONE, ...)
UML Diagram Taxonomy

Structure
- Class Diagram
  - Profile Diagram
  - Composite Structure Diagram
- Component Diagram
- Object Diagram
- Deployment Diagram
- Package Diagram

Behavior
- Activity Diagram
- Use Case Diagram
  - Interaction Diagram
    - State Machine Diagram
    - Sequence Diagram
    - Communication Diagram
    - Interaction Overview Diagram
    - Timing Diagram

(Source: Wikipedia)
Sequence Diagram

• A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline.

• Sequence diagrams are good at showing which objects communicate with which other objects; and what messages trigger those communications.

• Sequence diagrams are not intended for showing complex procedural logic.
Sequence Diagram

- Message - synchronous
  - Source
  - Target
  - message(message\(p\), parameter)
  - message(message\(r\))

- Message - asynchronous
  - Source
  - Target
  - message(message\(r\))

- Self-message & Recursion
  - Source
  - self\(m\)
  - recursion

- Lifeline
- Execution Occurrence

Object (or Actor)
Use Case Diagram Example

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login

First tier use cases

Second tier use cases

Actor:
- Tenant
- Landlord

System boundary:
- Timer
- LightSwitch
- LockDevice

Communication:
- «initiate»
- «participate»
- «initiate + participate»

Use case:
- UC1: Unlock
- UC2: Lock
- UC7: AuthenticateUser
# Use Case 1: Unlock

<table>
<thead>
<tr>
<th>Use Case UC-1:</th>
<th>Unlock</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Related Requirem’ts:</strong></td>
<td>REQ1, REQ3, REQ4, and REQ5</td>
</tr>
<tr>
<td><strong>Initiating Actor:</strong></td>
<td>Any of: Tenant, Landlord</td>
</tr>
<tr>
<td><strong>Actor’s Goal:</strong></td>
<td>To disarm the lock and enter, and get space lighted up automatically.</td>
</tr>
<tr>
<td><strong>Participating Actors:</strong></td>
<td>LockDevice, LightSwitch, Timer</td>
</tr>
</tbody>
</table>

### Preconditions:
- The set of valid keys stored in the system database is non-empty.
- The system displays the menu of available functions; at the door keypad the menu choices are “Lock” and “Unlock.”

### Postconditions:
The auto-lock timer has started countdown from `autoLockInterval`.

### Flow of Events for Main Success Scenario:

1. **Tenant/Landlord** arrives at the door and selects the menu item “Unlock”
2. include::{**AuthenticateUser**} (UC-7)
3. **System** (a) signals to the **Tenant/Landlord** the lock status, e.g., “disarmed,” (b) signals to **LockDevice** to disarm the lock, and (c) signals to **LightSwitch** to turn the light on
4. **System** signals to the **Timer** to start the auto-lock timer countdown
5. **Tenant/Landlord** opens the door, enters the home [and shuts the door and locks]
Sequence Diagram

Can be used to:

- Represent usage scenarios of actors interacting with the system-to-be (analysis)
- Represent interaction between objects within the system (design)

UC-1: unlock
State Machine Diagrams (Statecharts)

- SCs are used to detail the transitions (or changes of state) an object can go through in the system.
- SCs show how an object moves from one state to another and the rules that govern that change.
- SCs typically have a start and end condition.

Example: Statechart of class 'Controller'

Useful mainly in design and for testing
Statecharts and Testing

SCs can be used to design test cases

<table>
<thead>
<tr>
<th>Input (Event)</th>
<th>State</th>
<th>Locked (S1)</th>
<th>Accepting (S3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid-key</td>
<td>Locked</td>
<td>S2 (signal-success)</td>
<td>S2 (signal-success)</td>
</tr>
<tr>
<td>Invalid-key</td>
<td>Locked</td>
<td>S3 (signal-failure)</td>
<td>-</td>
</tr>
<tr>
<td>Invalid-key [numOfAttempts &lt;= maxNumOfAttempts]</td>
<td>-</td>
<td>S3 (signal-failure)</td>
<td></td>
</tr>
<tr>
<td>Invalid-key [numOfAttempts &lt;= maxNumOfAttempts]</td>
<td>-</td>
<td>S4 (sound-alarm)</td>
<td></td>
</tr>
</tbody>
</table>
UML Diagram Taxonomy

Structure

- Class Diagram
  - Profile Diagram
  - Composite Structure Diagram
- Component Diagram
- Object Diagram
- Deployment Diagram
- Package Diagram

Behavior

- Behaviour Diagram
  - Activity Diagram
  - Use Case Diagram
- Interaction Diagram
  - State Machine Diagram
- Interaction Overview Diagram
- Timing Diagram

(Source: Wikipedia)
Classes

- A class describes a group of objects with
  - similar properties (attributes),
  - common behaviour (operations),
  - common relationships to other objects,
  - and common meaning (“semantics”).

Example

employee:
- has a name, employee# and department;
- an employee is hired, and fired; an employee works in one or more projects
Classes

- A class describes a group of objects with
  - similar properties (attributes),
  - common behaviour (operations),
  - common relationships to other objects,
  - and common meaning ("semantics").

- Example

  employee:
  - has a name, employee# and department;
  - an employee is hired, and fired; and an employee works in one or more projects.
Objects vs. Classes

• The instances of a class are called objects

<table>
<thead>
<tr>
<th>Fred_Bloggs:employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: Fred Bloggs</td>
</tr>
<tr>
<td>Employee #: 234609234</td>
</tr>
<tr>
<td>Department: Marketing</td>
</tr>
</tbody>
</table>

In diagrams, write :employee in the name field to distinguish object from class

• Two different objects may have identical attribute values (like two people with identical name and address)

• Objects have associations with other objects
  • E.g. Fred_Bloggs:employee is associated with the KillerApp:project object
  • But we will capture these relationships at the class level (why?)
Associations

- Objects do not exist in isolation from one another
  - A relationship represents a connection between things

- Types of relationships
  - Association
  - Aggregation and Composition
  - Generalization

- Class diagrams show classes and their relationships
Associations

A staff member has zero or more clients on his/her clientList.

A client has exactly one staffmember as a contact person.

- **Multiplicity**
  - A staff member has zero or more clients on his/her clientList.
  - A client has exactly one staffmember as a contact person.

- **Name of the association**
  - liaises with

- **Role**
  - The staffmember’s role in this association is as a contact person.
  - The clients’ role in this association is as a clientList.

- **Direction**
  - The “liaises with” association should be read in this direction.

- **Multiplicity**
  - A staff member has zero or more clients on his/her clientList.
  - A client has exactly one staffmember as a contact person.

Client

- companyAddress
- companyEmail
- companyFax
- companyName
- companyTelephone

ClientList

staffName
staff#
staffStartDate
contact person
liaises with
Client

Role

Multiplicity

Name of the association

Direction
Multiplicity

- **Optional** (0 or 1) 0..1
- **Exactly one** 1 (alternative: 1..1)
- **Zero or more** 0..* (alternative: *)
- **One or more** 1..*
- **A range of values** 1..6
- **A set of ranges** 1..3, 7..10, 15, 19..*
Association classes

Sometimes the association is itself a class

...because we need to retain information about the association

...and that information doesn’t naturally live in the classes at the ends of the association

E.g. a “title” is an object that represents information about the relationship between an owner and her car

Car
VIN(vehicle Id Number)  yearMade  mileage

owns

Person
name
address
driversLicenceNumber
permittedVehicles

owns

Title
yearBought
initialMileage
pricePaid
LicencePlate#
Aggregation and Composition

• Aggregation
  • This is the “Has-a” or “Whole/part” relationship

• Composition
  • Strong form of aggregation that implies ownership:
    • if the whole is removed from the model, so is the part
    • the whole is responsible for the disposition of its parts
Generalisation

- Sub-classes **inherit** attributes, associations, & operations from the superclass
- A sub-class may override an inherited aspect
- Super-classes may be declared **{abstract}**, meaning they have no instances
  - Implies that the sub-classes cover all possibilities
Class Diagram

Aggregation
This is the “Has-a” or “Whole/part” relationship

Composition
Strong form of aggregation that implies ownership:
if the whole is removed from the model, so is the part
the whole is responsible for the disposition of its parts

Can be used to model the domain structure, i.e. concepts and their properties and relationships
Analysis versus Design

• During Analysis
  • we want to know about the application domain and the requirements
  • …so we develop a coarse-grained model to show where responsibilities are, and how objects interact
    • Our models show a message being passed, but we don’t worry too much about the contents of each message
    • To keep things clear, use icons to represent external objects and actors, and boxes to represent system objects

• During Design
  • we want to say how the software should work
  • … so we develop fine-grained models to show exactly what will happen when the system runs
    • e.g. show the precise details of each method call

What vs. How
Domain Models

• The Domain Model illustrates noteworthy concepts in a domain
  • The domain model is also called conceptual model, domain object model or analysis object model.

• To visualize domain models the UML class diagram notation is used
  • However, no operations are defined in domain models
  • Only ...
    • domain objects and conceptual classes
    • associations between them
    • attributes of conceptual classes
Example: Domain Model (snippet)

Domain model for UC-1: Unlock

```
<table>
<thead>
<tr>
<th>Controller</th>
<th>KeyChecker</th>
<th>KeyStorage</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>numAttempts</td>
<td>ArrayOfAttempts</td>
<td>conveys requests</td>
<td>retrieves valid keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>verifies</td>
</tr>
<tr>
<td></td>
<td>KeyChecker</td>
<td>KeyStorage</td>
<td>Attributes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>userIdentityCode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>timestamp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>doorLocation</td>
</tr>
</tbody>
</table>

```

“Reading direction arrow. ”
Has no meaning; it only helps reading the association label, and is often left out.
Sequence Diagram

Can be used to:

• Represent usage scenarios of actors interacting with the system-to-be (analysis)

• Represent interaction between objects within the system (design)

UC-1: unlock
Design: Object Interactions

System Sequence Diagram

User → System:
- select function("unlock")
- prompt for the key
- enter key
- verify key
- signal: valid key, lock open
- open the lock, turn on the light
- start ("duration")

System → User:
- System

Design Sequence Diagram

Controller: LockCtrl
- checkKey() sk := getNext()
- alt val != null setOpen(true)
- [else] val == null : setLit(true)

Checker: : Checker
- checkKey()
- sk := getNext()
- setOpen(true)

KeyStorage: : KeyStorage
- val
- val != null

Diagram elements:
- User (initiating actor)
- System (offstage actor)
- Controller: LockCtrl
- Checker: : Checker
- KeyStorage: : KeyStorage
- System (controller actor)
System Function 'Enter Key'

Much more detail added in each iteration

Alternative solutions are possible and must be discussed

Figure 2-33: Sequence diagram for the system function “enter key” (Figure 2-20). Several UML interaction frames are shown, such as “loop,” “alt” (alternative fragments, of which only the one with a condition true will execute), and “opt” (optional, the fragment executes if the condition is true).
Summary

Behavior

Use Cases

System Sequence Diagrams

Interaction Diagrams

Structure

Domain Model

Class Diagram

System Description

Implementation Program

import javax.rmi
import java.io.
import java.util

public class Hom
    implements Co
    protected In
    protected St
    public static
    public HomeA
    KeyStore
    }{try {
        ingu
    } catch

LockCtrl
LightCtrl
PhotoObs
Structure of Lecture 04

- Preliminaries and Context
  - OO Development Background
  - UML: UC Diagram, Class Diagram, Sequence Diagram
  - Analysis versus Design
- Domain Analysis and Modelling Example
  - Identifying Concepts (Responsibilities)
  - Attributes
  - Associations
- Domain Analysis and Modelling Summary
Domain Models

• Captures the most important concepts of the domain and their associations

• The domain is the background knowledge of the users of the system, e.g. the domain of the librarian includes books, loans, returns, patrons, …

• Helps us to understand and think about the concepts we will use in the project

• Provides a useful “glossary” for the project
Domain Model Relationships

- Conceptual Class Diagram: Classes, attributes, associations
- Use Case Model
- Domain Model
- Interaction Diagrams (Sequence Diagrams)
- Glossary
- Functional Requirements
- Dynamic Behavior

What do you learn about when and how to create these models?
Elements of a Domain Model

- The following elements enable us to express time invariant static business rules for a domain:
  - **Domain classes** – each domain class denotes a concept (type of object).
  - **Attributes** – an attribute is the description of a named slot of a specified type in a domain class; each instance of the class separately holds a value.
  - **Associations** – an association is a relationship between two (or more) domain classes that describes links between their object instances. Associations can have roles, describing the multiplicity and participation of a class in the relationship.
  - **Additional rules** – complex rules that cannot be shown with symbols can be shown with attached notes.
Simple Domain Model Example

Domain class

Person
- first name : String
- last name : String
- salary
- +owner

Company
- name
- +employer 0..1

Car
- type
- model name

<<Rule>>
If a person is not employed by a company then they do not have a car.

<<Rule>>
If a person is not employed by a company then they do not have a car.
What are Domain Classes?

- Each domain class denotes a concept.
  - Concept = Descriptor for a set of things that share common properties.
- Domain Classes can be:
  - Business objects - things that are manipulated in the business e.g. Order.
  - Real world objects - things that the business keeps track of e.g. Contract, Site.
  - Actors/Workers/Persons - e.g. Controller and Customer.
  - Events that transpire - e.g. Sale and Payment.
- A domain class has attributes and associations with other classes.
How to create a Domain Model?

Perform the following in very short iterations:

• Make a list of candidate domain classes.
• Draw these classes in a UML class diagram.
• If possible, add brief descriptions for the classes.
• Identify any associations that are necessary.
• Decide whether some domain classes are really just attributes.
• Where helpful, identify role names and multiplicity for associations.
• Add any additional static rules as UML notes that cannot be conveyed with UML symbols.
• Group diagrams/domain classes by category into packages.
• Concentrate more on just identifying domain classes in early iterations!
How to identify Domain Classes?

• An obvious way to identify domain classes is to identify **nouns** and **phrases** in textual descriptions of a domain.

Consider a use case description as follows:

1. **Customer** arrives at a **checkout** with **goods** and/or **services** to purchase.

2. **Cashier** starts a new **sale**.

3. **Cashier** enters **item identifier**.

4. System records the **sale line item** and presents the **item description**, **price** and running **total**.

   Class or Attribute?
Example: ATM Machine

(a) Actor (Bank customer) 
(ATM machine) 

(b) Domain Model 

Actor 

Concept 1 
Concept 2 
Concept 3 
Concept n 

System 

Deposit 
Withdraw 
Transfer 

Actor (Remote datacenter)
Consider the following problem description, analyzed for Subjects, Verbs, Objects:

The ATM verifies whether the customer's card number and PIN are correct. If it is, then the customer can check the account balance, deposit cash, and withdraw cash. Checking the balance simply displays the account balance. Depositing asks the customer to enter the amount, then updates the account balance. Withdraw cash asks the customer for the amount to withdraw; if the account has enough cash, the account balance is updated. The ATM prints the customer’s account balance on a receipt.

Analyze each subject and object as follows:

- Does it represent a person performing an action? Then it’s an actor, ‘R’.
- Is it also a verb (such as ‘deposit’)? Then it may be a method, ‘M’.
- Is it a simple value, such as ‘color’ (string) or ‘money’ (number)? Then it is probably an attribute, ‘A’.
- Which NPs are unmarked? Make it ‘C’ for class.
- Verbs can also be classes, for example: Deposit is a class if it retains state information.
Identifying Concepts (Domain Classes) from noun phrases

• Vision and Scope, Glossary and Use Cases are good for this type of linguistic analysis

• However:
  • Words may be ambiguous or synonymous
  • Noun phrases may be attributes or parameters rather than classes:
    • If it stores state information or it has multiple behaviors, then it’s a class
    • If it’s just a number or a string, then it’s probably an attribute
How to identify Attributes?

A domain class sounds like an attribute if …

• It relies on an associated class for its identity – e.g. ‘order number’ class associated to an ‘order’ class.
  • The ‘order number’ sounds suspiciously like an attribute of ‘order’.

• It is a simple data type – e.g. ‘order number’ is a simple integer.
  • Now it really sounds like an attribute!
Building a Domain Model

Step 1: Identifying the boundary concepts

Step 2: Identifying the internal concepts
Domain Modelling Strategies for Concept Identification

- 'Outside-In' Approach: First identify boundary concepts, then internal concepts
  - Internal concepts might be further classified into control and entity concepts

- 'Setting-up-an-enterprise' Approach: What workers need to be hired and what things acquired?
  - Start with 'worker' concepts and their responsibilities
    - Usually (at least) one 'Controller'
  - Distinguish between 'doing' (D) and 'knowing' (K) responsibilities
    - Usually: D=worker and K=thing; but not always clear
Use Cases vs. Domain Model

In **use case analysis**, we consider the system as a “**black box**”

In **domain analysis**, we consider the system as a “**transparent box**”
Use Case Diagrams

UC1: Unlock
UC2: Lock
UC3: AddUser
UC4: RemoveUser
UC5: InspectAccessHistory
UC6: SetDevicePrefs
UC7: AuthenticateUser
UC8: Login

Subsystem 1: Device Control

Subsystem 2: Account Management
## Use Case 1: Unlock

<table>
<thead>
<tr>
<th><strong>Use Case UC-1:</strong></th>
<th>Unlock</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Related Requirements:</strong></td>
<td>REQ1, REQ3, REQ4, and REQ5 stated in Table 2-1</td>
</tr>
<tr>
<td><strong>Initiating Actor:</strong></td>
<td>Any of: Tenant, Landlord</td>
</tr>
<tr>
<td><strong>Actor's Goal:</strong></td>
<td>To disarm the lock and enter, and get space lighted up automatically.</td>
</tr>
<tr>
<td><strong>Participating Actors:</strong></td>
<td>LockDevice, LightSwitch, Timer</td>
</tr>
</tbody>
</table>
| **Preconditions:** | • The set of valid keys stored in the system database is non-empty.  
• The system displays the menu of available functions; at the door keypad the menu choices are "Lock" and "Unlock." |
| **Postconditions:** | The auto-lock timer has started countdown from autoLockInterval. |

### Flow of Events for Main Success Scenario:

1. Tenant/Landlord arrives at the door and selects the menu item “Unlock”
2. include:: AuthenticateUser (UC-7)
3. System (a) signals to the Tenant/Landlord the lock status, e.g., “disarmed,” (b) signals to LockDevice to disarm the lock, and (c) signals to LightSwitch to turn the light on
4. System signals to the Timer to start the auto-lock timer countdown
5. Tenant/Landlord opens the door, enters the home [and shuts the door and locks]
### Example NL Requirements

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>5</td>
<td>The system shall keep the door locked at all times, unless commanded otherwise by authorized user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed).</td>
</tr>
<tr>
<td>REQ2</td>
<td>2</td>
<td>The system shall lock the door when commanded by pressing a dedicated button.</td>
</tr>
<tr>
<td>REQ3</td>
<td>5</td>
<td>The system shall, given a valid key code, unlock the door and activate other devices.</td>
</tr>
<tr>
<td>REQ4</td>
<td>4</td>
<td>The system should allow mistakes while entering the key code. However, to resist “dictionary attacks,” the number of allowed failed attempts shall be small, say three, after which the system will block and the alarm bell shall be sounded.</td>
</tr>
<tr>
<td>REQ5</td>
<td>2</td>
<td>The system shall maintain a history log of all attempted accesses for later review.</td>
</tr>
<tr>
<td>REQ6</td>
<td>2</td>
<td>The system should allow adding new authorized persons at runtime or removing existing ones.</td>
</tr>
<tr>
<td>REQ7</td>
<td>2</td>
<td>The system shall allow configuring the preferences for device activation when the user provides a valid key code, as well as when a burglary attempt is detected.</td>
</tr>
<tr>
<td>REQ8</td>
<td>1</td>
<td>The system should allow searching the history log by specifying one or more of these parameters: the time frame, the actor role, the door location, or the event type (unlock, lock, power failure, etc.). This function shall be available over the Web by pointing a browser to a specified URL.</td>
</tr>
<tr>
<td>REQ9</td>
<td>1</td>
<td>The system should allow filing inquiries about “suspicious” accesses. This function shall be available over the Web.</td>
</tr>
</tbody>
</table>
## Extracting the Responsibilities

<table>
<thead>
<tr>
<th>Responsibility Description</th>
<th>Type</th>
<th>Concept Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinate actions of all concepts associated with a use case, a logical grouping of use cases, or the entire system and delegate the work to other concepts.</td>
<td>D</td>
<td>Controller</td>
</tr>
<tr>
<td>Container for user’s authentication data, such as pass-code, timestamp, door identification, etc.</td>
<td>K</td>
<td>Key</td>
</tr>
<tr>
<td>Verify whether or not the key-code entered by the user is valid.</td>
<td>D</td>
<td>KeyChecker</td>
</tr>
<tr>
<td>Container for the collection of valid keys associated with doors and users.</td>
<td>K</td>
<td>KeyStorage</td>
</tr>
<tr>
<td>Operate the lock device to armed/disarmed positions.</td>
<td>D</td>
<td>LockOperator</td>
</tr>
<tr>
<td>Operate the light switch to turn the light on/off.</td>
<td>D</td>
<td>LightOperator</td>
</tr>
<tr>
<td>Operate the alarm bell to signal possible break-ins.</td>
<td>D</td>
<td>AlarmOperator</td>
</tr>
<tr>
<td>Block the input to deny more attempts if too many unsuccessful attempts.</td>
<td>D</td>
<td>Controller</td>
</tr>
<tr>
<td>Log all interactions with the system in persistent storage.</td>
<td>D</td>
<td>Logger</td>
</tr>
</tbody>
</table>

[Note: Incomplete, e.g., 'Timer' missing ...]
Domain Model (1a)

Domain concepts for subsystem #1 of safe home access

[Note: Incomplete, e.g., Logger, Timer missing ...]
Domain Model (1b)

- «boundary» HouseholdDeviceOperator
  - «boundary» LockOperator
  - «boundary» LightOperator
  - «boundary» MusicPlayerOperator
  - «boundary» AlarmOperator
# Extracting Associations

<table>
<thead>
<tr>
<th>Concept pair</th>
<th>Association description</th>
<th>Association name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller ↔ StatusDisplay</td>
<td>Controller sends current status of locks etc. to StatusDisplay for display</td>
<td>…</td>
</tr>
<tr>
<td>Controller ↔ KeycodeEntry</td>
<td>KeycodeEntry device reads Input and sends it to Controller for further processing</td>
<td>…</td>
</tr>
<tr>
<td>Key ↔ Controller</td>
<td>Controller obtains Key based on Input and metadata (door, time)</td>
<td>obtains</td>
</tr>
<tr>
<td>KeyChecker ↔ Controller</td>
<td>Controller sends Key for checking to KeyChecker and receives check result for further action</td>
<td>conveys requests</td>
</tr>
<tr>
<td>KeyStorage ↔ KeyChecker</td>
<td>KeyChecker retrieves valid keys from KeyStorage for comparison with Key</td>
<td>retrieves valid keys</td>
</tr>
<tr>
<td>Key ↔ KeyChecker</td>
<td>KeyChecker verifies Key</td>
<td>verifies</td>
</tr>
<tr>
<td>HouseholdDeviceOperator ↔ Controller</td>
<td>Controller activates Devices through requests to HouseholdDeviceOperator</td>
<td>conveys requests</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

...
## Extracting Attributes

<table>
<thead>
<tr>
<th>Concept</th>
<th>Attributes</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller</td>
<td>numOfAttempts</td>
<td>Used to determine whether another attempt is allowed</td>
</tr>
<tr>
<td></td>
<td>maxNumOfAttempts</td>
<td>Used to limit the retry attempts</td>
</tr>
<tr>
<td>HouseholdDeviceOperator</td>
<td>deviceStatuses</td>
<td>On, off, etc.</td>
</tr>
<tr>
<td>Key</td>
<td>userIdentityCode</td>
<td>Personal code of the resident</td>
</tr>
<tr>
<td></td>
<td>timestamp</td>
<td>Allows for tracking history</td>
</tr>
<tr>
<td></td>
<td>doorLocation</td>
<td>Allows possibility of different codes for doors and also relevant for tracking history</td>
</tr>
<tr>
<td>KeyChecker</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Domain Model (2a)

Associations: who needs to work together, not how they work together

Concept pair | Association description | Association name

Domain model for UC-1: Unlock

"Reading direction arrow."
Has no meaning; it only helps reading the association label, and is often left out.
Domain Model (2b)
General Recommendations

• Identifying the domain concepts is more important than identifying their associations and attributes
  • Every concept that the designer can discover, should be mentioned.
• For an association (or attribute), in order to be shown it should pass the “does it need to be mentioned?” test.
  • If the association in question is obvious, it should be omitted from the domain model.
  • For example, the association <<Controller–obtains–Key>> is fairly redundant.
  • Several other associations could as well be omitted, because the reader can easily infer them, and this should be done particularly in schematics that are about to become cluttered.
• Clarity should be preferred over detail!
More examples with explanation can be found in the Textbook
Structure of Lecture 04

• Preliminaries and Context
  • OO Development Background
  • UML: UC Diagram, Class Diagram, Sequence Diagram
  • Analysis versus Design
• Domain Analysis and Modelling Example
  • Identifying Concepts (Responsibilities)
  • Attributes
  • Associations
• Domain Analysis and Modelling Exercise
Statements about a Course Management System

• During a semester a lecturer reads one or more lectures
• Sometimes the lecturer is on leave to focus on doing research, in this case (s)he does not give a lecture
• A student usually attends one or more lectures, unless (s)he has something better to do
• During the semester there will be several exercises which are meant to be solved by small study groups
• Each student is assigned to one particular study group for the whole semester
• A study group consists of two to three students
• After submission of a solution by a study group it is graded by a tutor
• …
A class describes a set of objects with the same semantics, properties and behavior.

When used for domain modeling, it is a visualization of a real world concept.

- During a semester a lecturer reads one or more lectures.
- A student usually attends one or more lectures, ...
- During the semester there will be several exercises...
- Each student is assigned to one particular study group for the whole semester.
- ... it is graded by a tutor.
A class describes a set of objects with the same semantics, properties and behavior.

When used for domain modeling, it is a visualization of a real world concept.

- During a **semester** a **lecturer** reads one or more **lectures**
- A **student** usually attends one or more lectures, ...
- During the semester there will be several **exercises**...
- Each student is assigned to one particular **study group** for the whole semester
- ... it is graded by a **tutor**
Attributes are logical data values of an object
Attributes are logical data values of an object

Semester

Lecturer

Student

Exercise

Study Group

Tutor
An association is a relationship between classes.

The ends of an association are called roles. Roles optionally have a multiplicity, name and navigability.

- During a semester a **lecturer reads** one or more lectures
- A **student** usually **attends** one or more **lectures**, unless (s)he has something better to do
- Each **student** is **assigned to** one particular **study group** for the whole semester
- A **study group consists of** two to three **students**
An association is a relationship between classes.

The ends of an association are called roles. Roles optionally have a multiplicity, name and navigability.

- During a semester a **lecturer reads** one or more lectures
- A **student** usually **attends** one or more **lectures**, unless (s)he has something better to do
- Each **student** is **assigned to** one particular **study group** for the whole semester
- A **study group** **consists of** two to three **students**
First Draft of Domain Model

(Incomplete)
More examples of domain modeling

Course management and POS


Bank accounts

- [http://documentation.genesez.org/javaee/de.genesez.uml.modeling.domain.html](http://documentation.genesez.org/javaee/de.genesez.uml.modeling.domain.html)
Next Lecture

• Date/Time:
  • Friday, 06-Oct, 10:15-12:00
• Topic:
  • Development Infrastructure I (by Sander Soo, Nortal)
• For you to do:
  • Work on homework 2 assignment
  • Remember that all team members must be present in next week’s lab sessions (assessment of homework 2 progress)!