Chapter 2, Modeling with UML
Preliminaries (1)

Students from other departments than Informatik:  
*How do I get a Schein for this lecture?*

- Hörerschein: just ask (mailto:dutoit@in.tum.de).
- Vorlesung & Übung Schein: Feb 16, written exam.

Bachelor students:  
*Are there mandatory homeworks or a written exam in this lecture?*

- Optional homeworks, but no mandatory homeworks.
- Written exam on Feb 16
Preliminaries (2)

Praktikum registration:

http://www12.in.tum.de/projects/STARS2001/
before tonight 20:00

Hauptseminar Requirements Engineering

Thursdays 13:00-14:00
3 slots are still available

Book: “Object-Oriented Software Engineering: ...”

- Computerbücher am Obelisk
- Kanzler
- Lachner
Preliminaries (3)

Ground rules:

♦ If you stop understanding me for any reason (content, language, sound system), let me know.

♦ Ask (many) questions
  ♦ During the lecture
  ♦ After the lecture
  ♦ During the Sprechstunde
  ♦ Via E-mail
Overview

♦ What is modeling?
♦ What is UML?
♦ Use case diagrams
♦ Class diagrams
♦ Sequence diagrams
♦ Activity diagrams
♦ Summary
Motivation

♦ Realistic and useful systems are large and complex.
  ♦ Unix System V: 1 mio SLOC (source lines of code)
  ♦ HiPath telephone switch: 8.5 mio SLOC
  ♦ Windows2000: 40 mio SLOC

♦ Systems require the work of many people (developers, testers, managers, clients, users, etc.).

♦ Systems have an extended life cycle, hence they evolve.

♦ 1 mio SLOC with 100 persons ≠ 10 k SLOC with 1 person

-> Modeling
**Systems, Models, and Views**

- **Model:** Abstraction describing a system (or a subset)
- **View:** Selected aspects of a model
- **Notation:** Set of rules for representing views

- Views and models of a single system can overlap each other
Systems, Models, and Views

Aircraft

Blueprints

Flightsimulator

Scale Model

Electrical Wiring

Model 1

View 1

Model 2

View 2

View 3
Models, Views, and Systems (UML)
Concepts and Phenomena

♦ *Phenomenon*: An object in the world of a domain as you perceive it, for example:
  - The lecture you are attending
  - My blue watch

♦ *Concept*: Describes the properties of phenomena that are common, for example:
  - Lectures on software engineering
  - Blue watches

♦ A concept is a 3-tuple:
  - *Name*: distinguishes it from other concepts.
  - *Purpose*: properties that determine if a phenomenon is a member
  - *Members*: phenomena which are part of the concept.
## Concepts and Phenomena

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>A device that measures time.</td>
<td><img src="image" alt="Clock" /> <img src="image" alt="Sand Timer" /></td>
</tr>
</tbody>
</table>

- **Abstraction**: Classification of phenomena into concepts
- **Modeling**: Development of abstractions to answer specific questions about a set of phenomena while ignoring irrelevant details.
Concepts In Software: Type and Instance

♦ Type:
  ✦ An abstraction in the context of programming languages
  ✦ Name: int, Purpose: integral number, Members: 0, −1, 1, 2, −2, . . .

♦ Instance:
  ✦ Member of a specific type

♦ The type of a variable represents all possible instances the variable can take.

♦ The relationship between “type” and “instance” is similar to that of “concept” and “phenomenon.”

♦ Abstract data type:
  ✦ Special type whose implementation is hidden from the rest of the system.
Class

- Class:
  - An abstraction in the context of object-oriented languages
- Like an abstract data type, a class encapsulates both state (variables) and behavior (methods)
- Unlike abstract data types, classes can be defined in terms of other classes using inheritance
Object-Oriented Modeling

Application Domain
Application Domain Model

Solution Domain
System Model

UML Package

TrafficControl

Aircraft
TrafficController
FlightPlan
Airport

SummaryDisplay
MapDisplay

FlightPlanDatabase
TrafficControl
Application and Solution Domain

♦ Application Domain (Requirements Analysis):
  ♦ The environment in which the system is operating

♦ Solution Domain (System Design, Object Design):
  ♦ The available technologies to build the system
What is UML?

♦ UML (Unified Modeling Language)
  ♦ An emerging standard for modeling object-oriented software.
  ♦ Resulted from the convergence of notations from three leading object-oriented methods:
    ♦ OMT (James Rumbaugh)
    ♦ OOSE (Ivar Jacobson)
    ♦ Booch (Grady Booch)


♦ Supported by several CASE tools
  ♦ Rational ROSE
  ♦ Together/J
  ♦ ...
UML and This Course

♦ You can model 80% of most problems by using about 20% UML.

♦ In this course, we teach you those 20%.

♦ Today, we give you a brief overview.

♦ In subsequent lectures, we will introduce more concepts as needed.
**UML First Pass**

- Use case diagrams
  - Describe the functional behavior of the system as seen by the user.

- Class diagrams
  - Describe the static structure of the system: Objects, Attributes, and Associations.

- Sequence diagrams
  - Describe the dynamic behavior between actors and the system and between objects of the system.

- Statechart diagrams
  - Describe the dynamic behavior of an individual object as a finite state machine.

- Activity diagrams
  - Model the dynamic behavior of a system, in particular the workflow, i.e. a flowchart.
UML First Pass: Use Case Diagrams

Use case diagrams represent the functionality of the system from user’s point of view.
Class diagrams represent the structure of the system
Sequence diagrams represent the behavior as interactions
UML First Pass: Statechart Diagrams

- **Event**
  - button1&2Pressed

- **Initial state**
  - Blink Hours
  - Blink Minutes

- **Transition**
  - button1&2Pressed
  - button1Pressed
  - button2Pressed

- **State**
  - Increment Hours
  - Increment Minutes
  - Increment Seconds

- **Final state**
  - Stop Blinking
  - Blink Seconds
Other UML Notations

UML provide other notations that we will be introduced in subsequent lectures, as needed.

♦ Implementation diagrams
  ♦ Component diagrams
  ♦ Deployment diagrams
  ♦ Introduced in lecture on System Design
♦ Object Constraint Language (OCL)
  ♦ Introduced in lecture on Object Design
UML Core Conventions

- Rectangles are classes or instances
- Ovals are functions or use cases
- Instances are denoted with an underlined names
  - \texttt{myWatch:SimpleWatch}
  - \texttt{joe:Firefighter}
- Types are denoted with nonunderlined names
  - \texttt{SimpleWatch}
  - \texttt{Firefighter}
- Diagrams are graphs
  - Nodes are entities
  - Arcs are relationships between entities
**UML Second Pass: Use Case Diagrams**

- **Actors** represent roles, that is, a type of user of the system
- **Use cases** represent a sequence of interaction for a type of functionality
- The use case model is the set of all use cases. It is a complete description of the functionality of the system and its environment

Used during requirements elicitation to represent external behavior

- Actor: Passenger
- Use case: PurchaseTicket
**Actors**

♦ An actor models an external entity which communicates with the system:
  - User
  - External system
  - Physical environment

♦ An actor has a unique name and an optional description.

♦ Examples:
  - Passenger: A person in the train
  - GPS satellite: Provides the system with GPS coordinates
Use Case

A use case represents a class of functionality provided by the system as an event flow.

A use case consists of:
- Unique name
- Participating actors
- Entry conditions
- Flow of events
- Exit conditions
- Special requirements
Use Case Example

Name: Purchase ticket

Participating actor: Passenger

Entry condition:
- Passenger standing in front of ticket distributor.
- Passenger has sufficient money to purchase ticket.

Exit condition:
- Passenger has ticket.

Event flow:
1. Passenger selects the number of zones to be traveled.
2. Distributor displays the amount due.
3. Passenger inserts money, of at least the amount due.
4. Distributor returns change.
5. Distributor issues ticket.

Anything missing?

Exceptional cases!
The <<extend>> Relationship

- <<extend>> relationships represent exceptional or seldom invoked cases.
- The exceptional event flows are factored out of the main event flow for clarity.
- Use cases representing exceptional flows can extend more than one use case.
- The direction of a <<extend>> relationship is to the extended use case.
The <<include>> Relationship

- An <<include>> relationship represents behavior that is factored out of the use case.
- An <<include>> represents behavior that is factored out for reuse, not because it is an exception.
- The direction of a <<include>> relationship is to the using use case (unlike <<extend>> relationships).
Class Diagrams

- Class diagrams represent the structure of the system.
- Class diagrams are used
  - during requirements analysis to model problem domain concepts
  - during system design to model subsystems and interfaces
  - during object design to model classes.
## Classes

- A **class** represents a concept.
- A class encapsulates state (**attributes**) and behavior (**operations**).
- Each attribute has a **type**.
- Each operation has a **signature**.
- The class name is the only mandatory information.

<table>
<thead>
<tr>
<th>TariffSchedule</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>zone2price</td>
<td></td>
</tr>
<tr>
<td>getPrice</td>
<td></td>
</tr>
<tr>
<td>getZones()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TariffSchedule</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table zone2price</td>
<td></td>
</tr>
<tr>
<td>Enumeration getZones()</td>
<td></td>
</tr>
<tr>
<td>Price getPrice(Zone)</td>
<td></td>
</tr>
</tbody>
</table>
Instances

<table>
<thead>
<tr>
<th>tariff_1974:TarifSchedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>zone2price = {</td>
</tr>
<tr>
<td>{‘1’, .20},</td>
</tr>
<tr>
<td>{‘2’, .40},</td>
</tr>
<tr>
<td>{‘3’, .60}</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
♦ An *instance* represents a phenomenon.
♦ The name of an instance is *underlined* and can contain the class of the instance.
♦ The attributes are represented with their *values*.
Actor vs. Instances

♦ What is the difference between an actor and a class and an instance?

♦ Actor:
  ♦ An entity outside the system to be modeled, interacting with the system (“Pilot”)

♦ Class:
  ♦ An abstraction modeling an entity in the problem domain, inside the system to be modeled (“Cockpit”)

♦ Object:
  ♦ A specific instance of a class (“Joe, the inspector”).
Associations

- Associations denote relationships between classes.
- The multiplicity of an association end denotes how many objects the source object can legitimately reference.
1-to-1 and 1-to-Many Associations

1-to-1 association

Country

name:String

1

Has-capital

City

name:String

1

1-to-many association

Polygon

draw()

1

Point

x:Integer

y:Integer

*
Aggregation

- An **aggregation** is a special case of association denoting a “consists of” hierarchy.
- The **aggregate** is the parent class, the **components** are the children class.

![Diagram](image-url)
Composition

♦ A solid diamond denote composition, a strong form of aggregation where components cannot exist without the aggregate.
Generalization

♦ Generalization relationships denote inheritance between classes.
♦ The children classes inherit the attributes and operations of the parent class.
♦ Generalization simplifies the model by eliminating redundancy.
From Problem Statement to Code

Problem Statement
A stock exchange lists many companies. Each company is identified by a ticker symbol

Class Diagram

Java Code
```java
public class StockExchange {
    public Vector m_Company = new Vector();
}
public class Company {
    public int m_tickerSymbol;
    public Vector m_StockExchange = new Vector();
}
```
UML Sequence Diagrams

- Used during requirements analysis
  - To refine use case descriptions
  - To find additional objects ("participating objects")
- Used during system design
  - To refine subsystem interfaces
- **Classes** are represented by columns
- **Messages** are represented by arrows
- **Activations** are represented by narrow rectangles
- **Lifelines** are represented by dashed lines

Diagram:

- Passenger
- TicketMachine
- selectZone()
- insertCoins()
- pickupChange()
- pickUpTicket()
UML Sequence Diagrams: Nested Messages

- The source of an arrow indicates the activation which sent the message
- An activation is as long as all nested activations
Sequence Diagram Observations

♦ UML sequence diagram represent behavior in terms of interactions.
♦ Complement the class diagrams which represent structure.
♦ Useful to find participating objects.
♦ Time consuming to build but worth the investment.
Activity Diagrams

♦ An activity diagram shows flow control within a system

![Activity Diagram]

♦ An activity diagram is a special case of a state chart diagram in which states are activities (“functions”)

♦ Two types of states:
  ♦ *Action state:*
    ♦ Cannot be decomposed any further
    ♦ Happens “instantaneously” with respect to the level of abstraction used in the model
  ♦ *Activity state:*
    ♦ Can be decomposed further
    ♦ The activity is modeled by another activity diagram
Activity Diagram: Modeling Decisions

Open Incident

Allocate Resources

[lowPriority]

[not fire & highPriority]

Notify Fire Chief

Notify Police Chief

[fire & highPriority]
Activity Diagrams: Modeling Concurrency

♦ Synchronization of multiple activities
♦ Splitting the flow of control into multiple threads
Activity Diagrams: Swimlanes

- Actions may be grouped into swimlanes to denote the object or subsystem that implements the actions.
Summary

♦ UML provides a wide variety of notations for representing many aspects of software development
  ♦ Powerful, but complex language
  ♦ Can be misused to generate unreadable models
  ♦ Can be misunderstood when using too many exotic features

♦ We concentrate only on a few notations:
  ♦ Functional model: use case diagram
  ♦ Object model: class diagram
  ♦ Dynamic model: sequence diagrams, statechart and activity diagrams
Next steps

♦ UML modeling tool: *Together/J* tutorial in November
♦ UML concepts will be revisited in subsequent lectures.
  ♦ Requirements lectures: Use case diagrams & Class diagrams
  ♦ System design lectures: Deployment diagrams
  ♦ Object design lectures: More class diagrams
  ♦ ...
♦ Stay tuned for the Requirements Elicitation lecture