Object-Oriented Analysis and Design with UML

Dr. Uwe Assmann
Research Institute for Integrational Software Engineering

Contents

- Modelling in Analysis und Design
- Functional Model
- Static Model (Structural model, object model)
  - Rearranging the static model
- Dynamic Model (Behavioral Model)

What is Object-Oriented Modeling?

Object-Orientation as a Design Method

- So far, we had design methods...
  - Parnas „Information Hiding“ Modular design
  - SA,...
- OO-Development is a Design Method
  - Notation (UML)
  - Process (different)
    - Rational Unified Process
    - Use-Case Driven Process with Object-Oriented Analysis
  - Heuristics
- OO-Development **models ....**

Modeling What?

- The world (**Analysis**)
  - WHAT?
  - prob lemss (Problem Analysis, Problem Spezifikation)
  - Requirements
  - Systems of the real world (Simulation)
  - ....Object-Oriented Analysis OOA
- Software systems (**Design**)
  - WHAT? HOW?
  - Object-Oriented Design OOD
- .... then **Implementation**.....
  - WHAT? HOW? BY WHAT? HOW GOOD?
  - Object-Oriented implementation OOP

5

Modeling the real world

6

Analysis vs Design

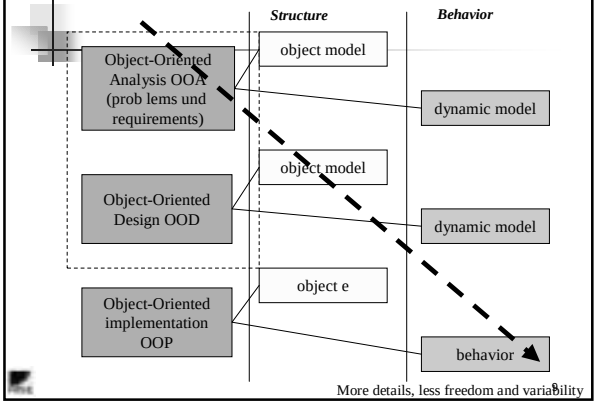
- Analysis is driven of the problem domain, I.e. the real world
  - Integrates constraints of the problem
- Design is driven by the solution domain
  - Integrates constraints of the implementation
  - Design should be reuse driven
  - Design treats parts of the HOW and BY WHAT, the rest is done by the implementation
- Good design
  - Can be measured (metrics)
  - Results of best practices (heuristics, design patterns)
  - Imposes an architectural style to a system

OO Analysis vs OO Design

- So, OO Development considers both problems and solutions as composed of *objects* and their *relations*
- Both OOA and OOD use OO Modelling
- Different Relations between Analysis and Design
  - Analysis == Design in simple cases
  - Analysis is Design in reengineering
  - Otherwise, Analysis model is evolved to Design model
- In all cases
  - Structural model (static model, architectural model)
    - Class model
    - Object model
  - Behavioral model (dynamic model) as a modeling of functional and behavioral aspects
    - state charts, interaction and sequence diagrams
  - Functional model
    - use cases
    - Often, the functional model is considered part of the behavioral model

8

The Classic OO Software Process

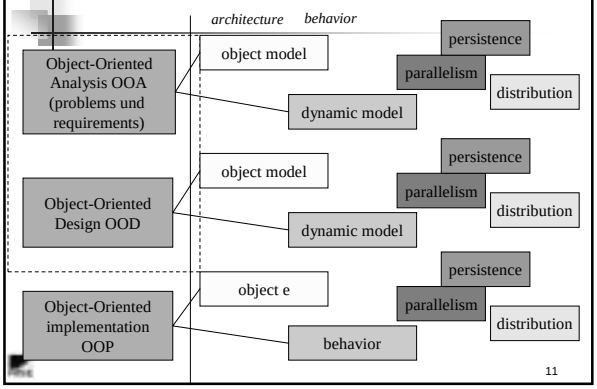


Object-Oriented Modelling

- Static Model describes Structure
  - Objects, their classes, associations (relations), inheritance
  - Packaging (subsystems)
- Dynamic Model shows behavior and function
  - Use cases describe uses and functions
  - Interaction, sequence diagrams show collaborations between objects
  - Statecharts describe how an object reacts on events (messages)

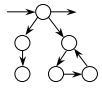
10

Other Aspects

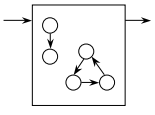


System Design

- Traditional modular decomposition:
  - Develop hierarchies
  - Decompose system, data and procedures



- Object-Oriented decomposition:
  - Better Modularity by encapsulation in objects
  - Dynamic architectures due to polymorphism



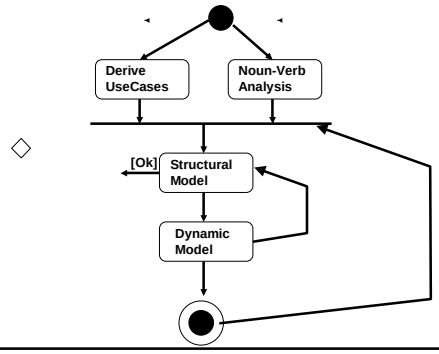
12

Object-Oriented Modeling with UML

.. In analysis and design

RISE

OO Modelling



14

Functional Model with Use Cases

Often is considered to be part of the dynamic model

RISE

Use Case Diagram

- A Use Case Diagram consists of several *use cases* of a system
- A *use case* describes an application, a coarse-grain function of a system, in a certain relation with *actors*
- A use case contains a *scenario sketch*
  - Pseudocode text which describes the functionality
- Use cases are good for
  - Documentation
  - Communication with customers and designers
  - Are started for the first layout of the structural model
  - Sometimes called „stories“ which should fit on one muddy card (XP)

16

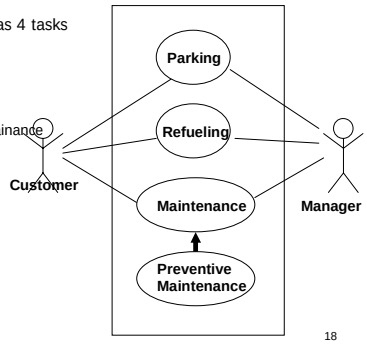
Questions for Use Cases

- What is the system/subsystem?
- Who is Actor?
  - A user
    - An active object
    - A person
    - A system
  - Must be external to the described system
- What are the Applications/Uses?
- What are the relations among Use Cases
  - Extends: Extend an existing use case (Inheritance)
  - Uses: Reuse of an existing use case (Sharing)

17

Example Service Station

- A Service Station has 4 tasks [Pfleeger]
  - Parking
  - Refueling
  - Maintenance
  - Preventive Maintainance



18

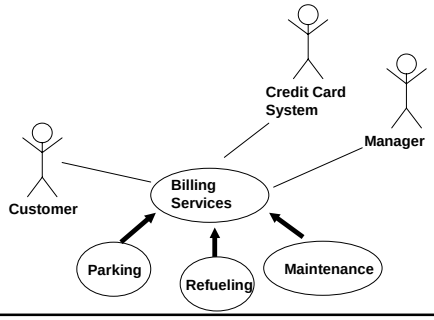
Questions to help

- What
  - Users
  - External systems
-
  - use
  - Need
- the system for which tasks?
- Are tasks or relations to complex?

19

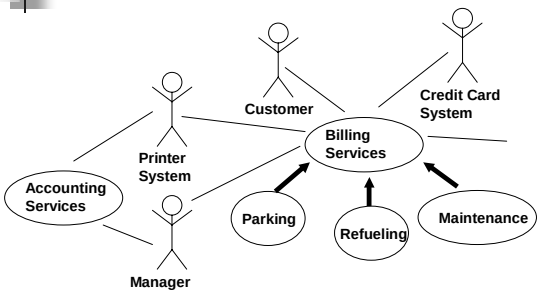
Refinement Service Station

- We introduce an abstraction of the services



20

Second Refinement Service Station



21

Check List Use Case Diagram

- Clarity
- Simplicity
- Completeness
- Match the stories of the customer?
- Missing actors?

22

Static Model (Structural Model, Object Model)

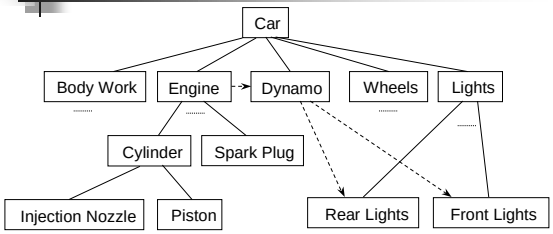
RISE

Static Model (Structural Model, Object Model)

- Simple form of architectural description
- On Classes
  - Types
  - Sets of objects
- Packages
  - Groups of classes
  - Subsystems

24

Hierarchical System Decomposition



Can be done with Classes (HOOD, not UML) or Packages (UML)

25

Questions to Find the Static Model

- Which objects do we need?
  - May be derived from actors and tasks in use cases
- Which features do they have?
- How can the objects be classified?
  - Which classes?
  - Which generic classes?
- How are the objects used?
  - Relations to others
  - Collaborations with others
- Which actions are performed by the objects?
- In which states are the objects and when do they change them?

26

How to find Objects and Classes

- Design with *responsibilities*
  - Ask the question:
    - Which object is *responsible* for which task?
- Use Noun-Verb-Analysis
  - Who is involved?
  - What does it do?
- Principle of Responsibility-based OOD:
  - Every object is responsible for certain tasks.
  - Either it has all the capabilities to do the job itself or
  - It cooperates with others to achieve the task

27

Class Responsibility Cards (CRC)

- Fix the role an object has in the system
  - Responsibilities
  - Cooperation partners

Class Name	*Cooperation with*
Responsible for	

28

Student	*Cooperating with*
Responsible for	
order Accept paket	Computer Dealer Parcel Service

Parcel Service	*Cooperating with*
Responsible for	
Accept paket Deliver paket	Computer Dealer Student

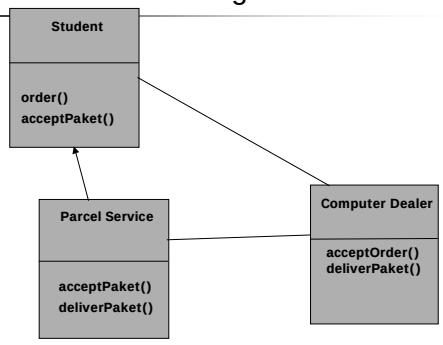
Computer Dealer	*Cooperating with*
Responsible for	
Accept Order Deliver paket	Student Parcel Service

Noun-Verb-Analysis

- Analyze Use Case Diagrams or Requirements Specification
- Find objects by
  - nouns and subjects
- Find cooperations form
  - Subject-object relations
  - genitivs
- Find activities from
  - Verbs
  - Substantivated verbs
- Example
  - "When the *driver* turns on the *lights* the *battery* is discharged. When the *engine* runs the *dynamo* recharges the *battery*..."

30

Our First Class Diagram



31

Modeling the real solution

Prototype of a
washing machine

Find the right abstractions!

32

The Basic Laws of Misunderstanding

Spoken is not heard
Heard is not listened
Listened is not understood
Understood is not accepted
Accepted is not done

33

Special Male/Female Instance

When did you design
for the last time
the dreams of your wife?

34

Arranging the Structural Model with Operations on Classes

RISE

Restructuring of Structural Model

- Factor out *commonalities*
  - into super classes
  - into generic classes
  - into delegated classes
- Separate *variabilities (differences)*
  - into subclasses
  - into Type parameters of generic classes
  - in delegierte classes
- Goal: product families (frameworks)
  - Their common code works for all products in the family
  - realized by polymorphism
- Hollywood Principle: Dont call us, we call you:
  - Old code can call new code
  - Framework classes can call user-based extensions

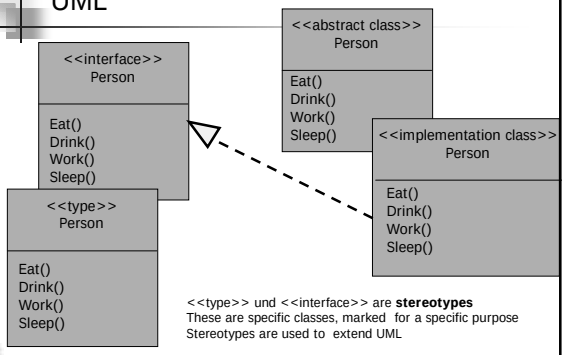
36

Abstract classes

- Abstract classes are between classes und types
  - At least one method is not implemented, i.e., only an interface (abstract method, method signature)
- Cannot be instantiated to objects
  - But used for inheritance
- *Interfaces* are fully abstract classes
  - Define a type
  - Define a service for a class to which they are inherited
  - Only consist of method signatures

37

Abstract Classes and Interfaces in UML



<<type>> und <<interface>> are **stereotypes**
These are specific classes, marked for a specific purpose
Stereotypes are used to extend UML

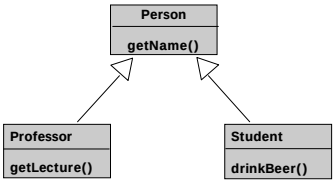
38

Single and Multiple Inheritance

- Single Inheritance
  - Every class has at most one super class from which it inherits features common with brothers and sisters
  - The inheritance relation is a tree
- Multiple Inheritance
  - Several super classes possible
  - Inheritance relation is a directed acyclic graph (dag, partial order)
  - Class may inherit a feature with the same name twice!

39

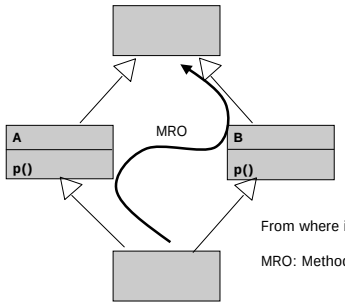
Single Inheritance



Super class S contains common features for all subclasses

40

Feature Resolution in Multiple Inheritance

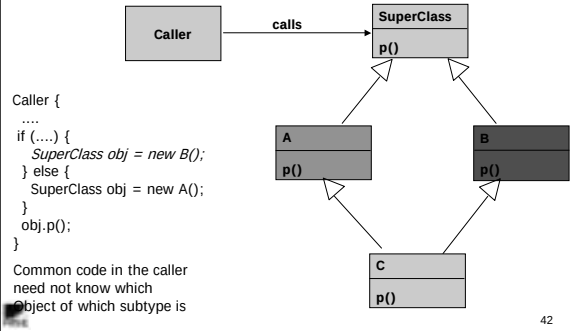


From where is p() inherited?

MRO: Method-Resolution-Order

41

Dynamic Architecture with Inheritance and Polymorphism




```
Caller {
  ....
 if (....) {
    SuperClass obj = new B();
  } else {
    SuperClass obj = new A();
  }
  obj.p();
}
```

Common code in the caller need not know which object of which subtype is

42

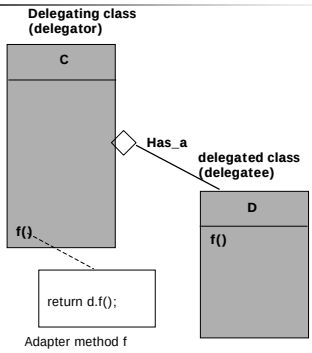
Inheritance - Evaluation

- \+ Reuse easy
- \+ With few effort new specialized classes can be built
- \+ Easy Change Propagation
  - \+ Changes impact the whole subclass tree
- \+ Dynamic architectures with exchange of subclass objects (polymorph.)
- – Violation of information hiding
  - Subclasses know too much about super classes; invalidating super class code is easy
  - – Fragile base class problem: Changes of super classes lead may to invalidation of sub classes
- – Bad readability
  - – To understand a class, all super classes must be understood
- – Inheritance does not ensure substitutability (product families are not easy)

43

Delegation

An object may *delegate* a task to another object



Adapter method f

44

Evaluation of Delegation

- Delegation simulate
  - Simple and Multiple Inheritance
  - Genericity
- Delegation creates „object schizophrenia“
  - If delegatee and delegator belong logically together they are physically different
  - What happens if delegatee calls `self`? Who is meant?

What‘s that?

```
λ type. class Set {
     boolean contains(type element);
     insert(type element);
     remove(type element);
}


appleSet = Set(Apple);
bananaSet = Set(Banana);
```

Generic Classes

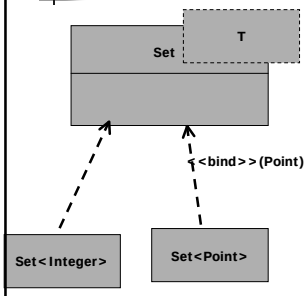
- A class template is to be bould by a type parameter
- Notation in Generic Java:

```
class Set<type> {
     boolean contains(type element);
     insert(type element);
     remove(type element);
}
Set<Apple> appleSet = new Set(Apple);
Set<Banana> bananaSet = appleSet;  // WRONG
```

Generic Classes

- For algorithmic schemata
  - Commen schema is in the generic class
  - Variable behavior is in the type parameter
- Often used for container
  - Lists, sets, trees, graphs
- Better type check

Generic Classes in UML



- In Generic Java:

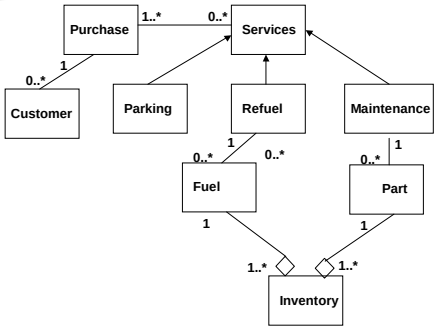
```
public class Set<T> {
...
}
```

- Binding:

```
Set<Integer> s;
Set<Point> s;
```

49

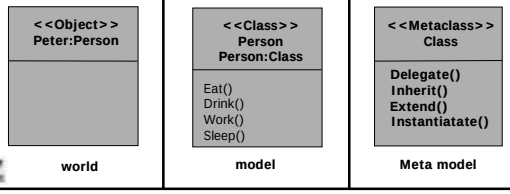
Structure Diagram Service Station



50

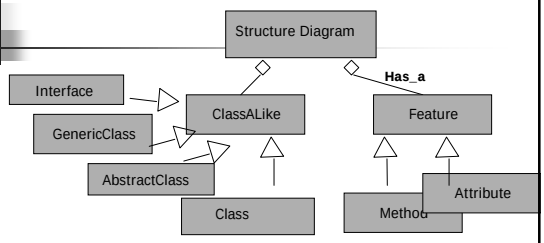
Objects, Classes, Meta Classes

- Meta data *describes* data
- A class can be considered as an object of the *meta level* because it describes an object on the *base level*
- A meta class describes a class and is part of the *meta model*



51

Meta Model of UML



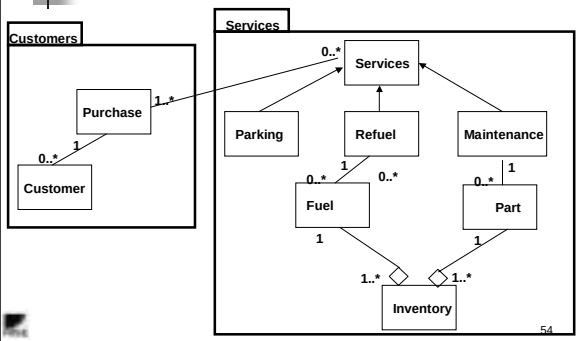
The UML meta model *describes* UML. It contains all concepts of UML as classes.
Also, stereotypes are described as classes.
Some stereotypes are predefined.
I.e, Interface, AbstractClass, GenericClass, Class are standard classes in the meta model of UML.
User-defined stereotypes are considered as *extensions* of the UML meta model.

52

Package Diagrams

- Packages are subsystems and allow to structure the classes.
- Packages represent an element of the modular design in the object-oriented notation UML
- Package relations should be acyclic
  - Similar to the uses relation in layered systems

Example Packages Service Station



However: Coplien's Law on Software Structure

Software is always structured in the same way as the organisation which built it.

Dynamic Model (Behavioral Model)

Diagrams for Dynamic Scenarios

- Sequence diagram
  - Sequence of messages over time
  - Object „life lines“
- Collaboration diagram
  - Numbering of messages over time
- Statechart
  - Nested finite automata describing states of classes

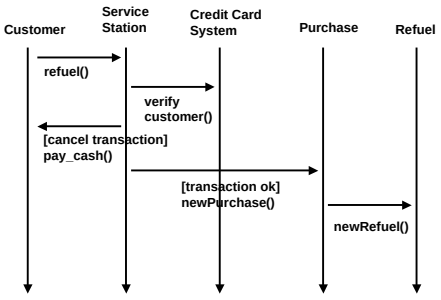
57

Steps towards the Dynamic Model

- Derive from the use cases
  - Sequence diagram
  - Cooperation diagram
- Which describe how objects collaborate
- Describe states of classes in statechart
  - Derive from sequence and cooperation diagram, as well as use cases
- Add operations to class diagram

58

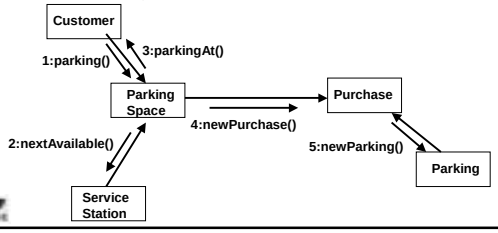
Sequence Diagram Service Station



59

Collaboration diagram

- Forget timeline, layout flexible
- Sketch objects
- Sketch operations as arrows between objects
- Number operations



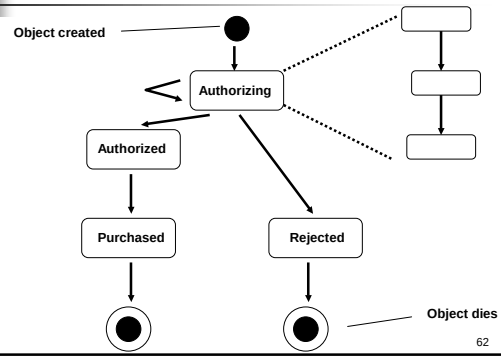
60

Statechart

- A *finite state machine* consists of *states* and *transitions*
  - In a state, an *event* occurs and invokes a transition to another state
- Statecharts are nested finite automata
  - States can be refined to subautomata
  - Parallel branches can be executed in parallel

61

Example Statechart Service Station



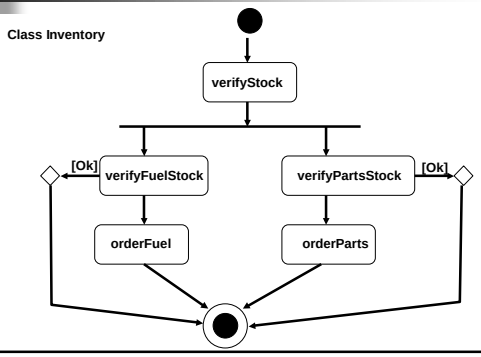
62

Activity diagram

- Activities with flow of data between them
  - Data flow diagram
- Similar to Petri nets
  - Petri nets are even better because they can be analyzed so that tools can be built for them
- Parallel branches are parallel
- Guard conditions specify branches in control flow (conditional data flow)

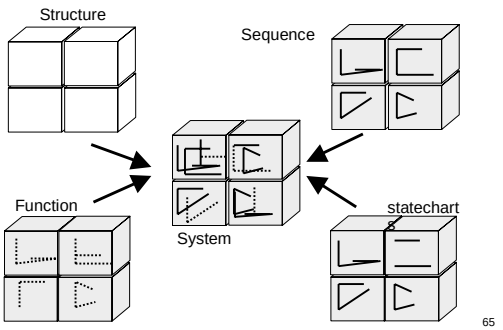
63

Example Activity Diagram Service Station



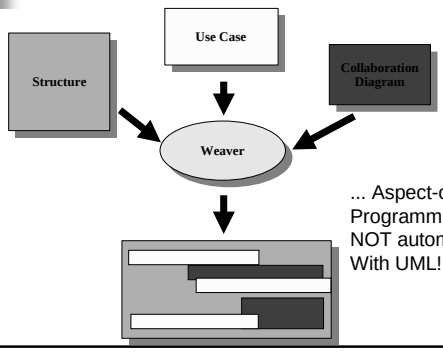
64

UML is an Aspect-Oriented Language



65

The Dream: A UML Weaver



... Aspect-oriented Programming is NOT automatic With UML!

66

The End

RISE