

# Part II

## Design Patterns and Frameworks

1

Prof. Dr. U. Aßmann  
 Chair for Software Engineering  
 Faculty of Informatics  
 Dresden University of Technology  
 13-0.3, 11/19/13

10) Role-based Design  
 11) Design Patterns as Role Models  
 12) Framework Variability  
 13) Framework Extensibility

Version numbers greater 1.0 contain corrections and improvements after lecturing  
 Design Patterns and Frameworks, © Prof. Uwe Aßmann



## 10. Role-Based Design – A Concept for Understanding Design Patterns and Frameworks

3

Prof. Dr. U. Aßmann  
 Chair for Software Engineering  
 Faculty of Informatics  
 Dresden University of Technology

1) Role-based Design  
 2) Role-Model Composition  
 3) Role Mapping in the MDA  
 4) Implementing Abilities  
 5) More on Roles

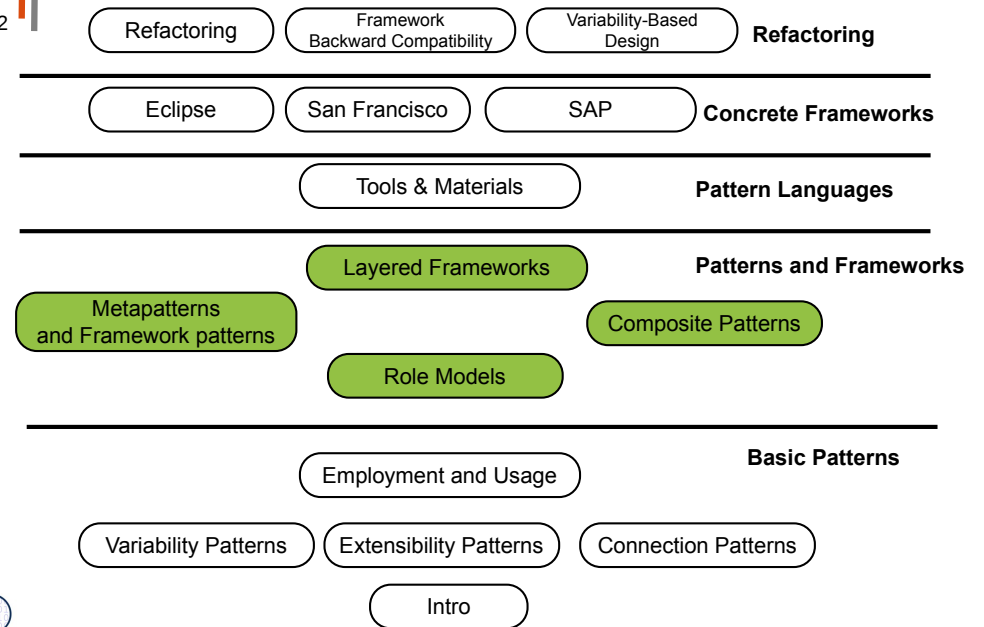
Design Patterns and Frameworks, © Prof. Uwe Aßmann



## Overview of the Course

2

Prof. Uwe Aßmann, Design Patterns and Frameworks



## Literature (To Be Read)

4

Prof. Uwe Aßmann, Design Patterns and Frameworks



- ▶ D. Riehle, T. Gross. Role Model Based Framework Design and Integration. Proc. 1998 Conf. On Object-oriented Programming Systems, Languages, and Applications (OOPSLA 98) ACM Press, 1998. <http://citeseer.ist.psu.edu/riehle98role.html>
- ▶ Liping Zhao. Designing Application Domain Models with Roles. In: Uwe Aßmann, Mehmet Aksit and Arend Rensink. Model Driven Architecture European MDA Workshops: Foundations and Applications, MDFA 2003 and MDFA 2004, Lecture Notes in Computer Science, Volume 3599, 2005, DOI: 10.1007/11538097
  - <http://www.springerlink.com/content/f8u0vmbbt2mf/#section=590861>

## Other Literature

- 5
- ▶ T. Reenskaug, P. Wold, O. A. Lehne. Working with objects. Manning publishers.
    - The OOram Method, introducing role-based design, role models and many other things. A wisdom book for design. Out of print. Preversion available on the internet at <http://heim.ifi.uio.no/~trygver/documents/book11d.pdf>
    - Same age as Gamma, but much farer..
  - ▶ H. Allert, P. Dolog, W. Nejdl, W. Siberski, F. Steimann. *Role-Oriented Models for Hypermedia Construction – Conceptual Modelling for the Semantic Web*. citeseer.org.



## Goal

- 7
- ▶ Understand the difference between roles and objects, role types (abilities) and classes
  - ▶ Understand role merging
  - ▶ and role mapping to classes
    - How roles can be implemented
  - ▶ Understand role model composition
  - ▶ Understand design patterns as role models, merged into class models
  - ▶ Understand composite design patterns
    - Understand how to mine composite design patterns
  - ▶ Understand role types as semantically non-rigid founded types
  - ▶ Understand layered frameworks as role models



## Other Literature

- 6
- ▶ B. Woolf. The Object Recursion Pattern. In N. Harrison, B. Foote, H. Rohnert (ed.), *Pattern Languages of Program Design 4 (PLOP)*, Addison-Wesley 1998.
  - ▶ Walter Zimmer. Relationships Between Design Patterns. *Pattern Languages of Program Design 1 (PLOP)*, Addison-Wesley 1994



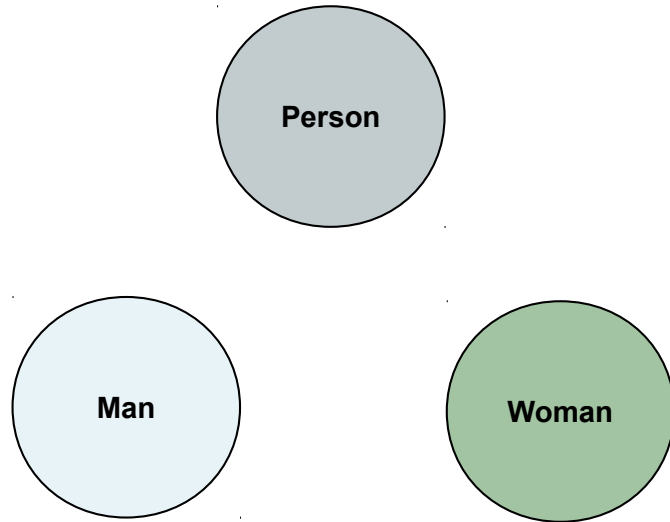
## 10.1 Role-based Design With Role Models

8



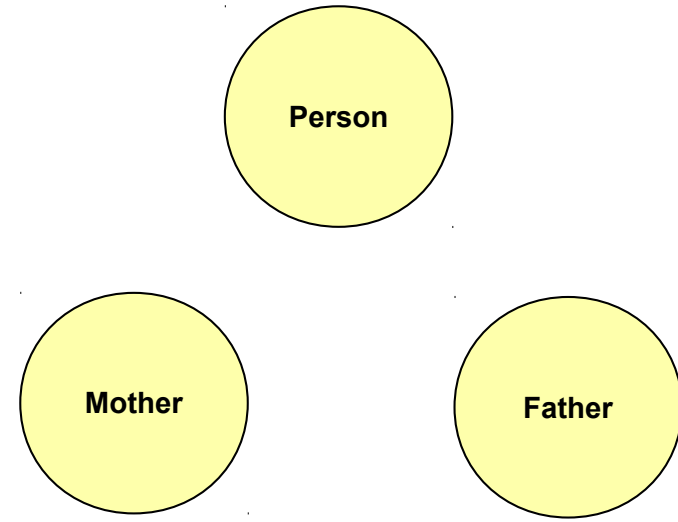
## A Riddle..

9



## Another Riddle..

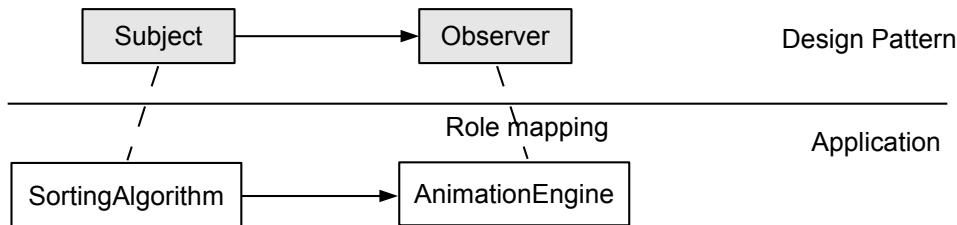
10



## Purpose of Teaching Role-based Design

11

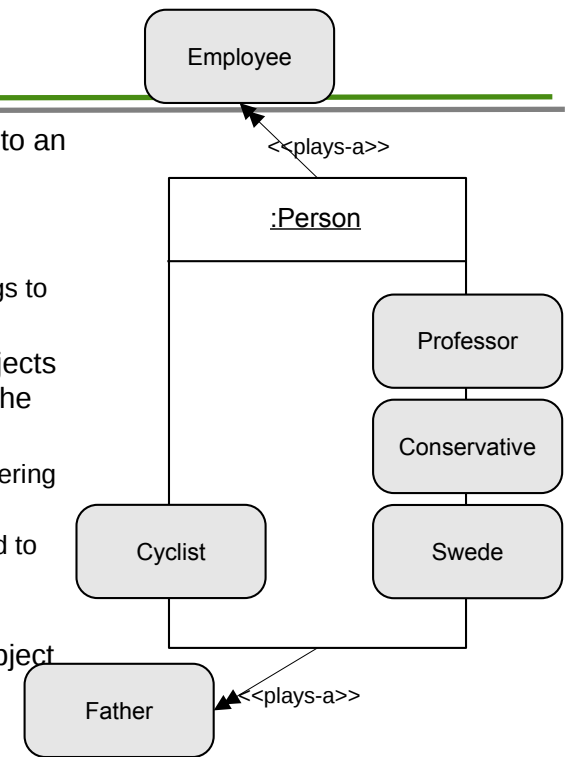
- ▶ Design patterns rely on the concept of *roles*
  - although not described as such in [Gamma]
- ▶ A design pattern must be matched in (mapped to) an application,
  - i.e., there must be some classes in the application that *play the roles* of the classes in the design pattern.
  - Every class in the design pattern is a role type
  - The matched class of the application plays the role of the class in the design pattern



## What are Roles?

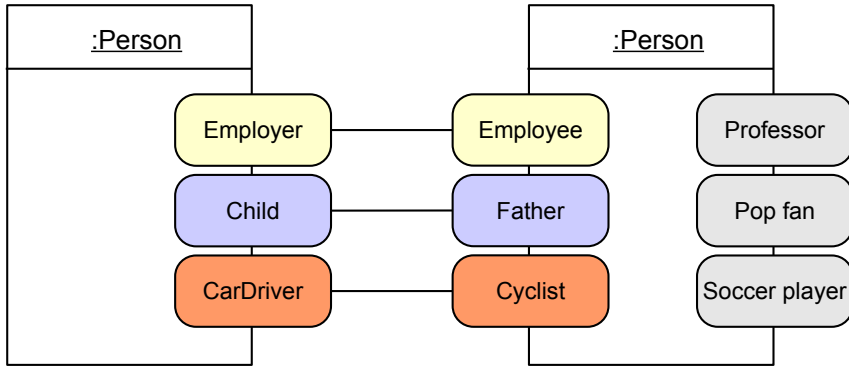
12

- ▶ A *role* is a *dynamic view* onto an object
  - The view can change dynamically
  - A role of an object belongs to a area of concern
- ▶ Roles are *played* by the objects (the object is the *player* of the role)
  - Playing a role means entering a state
  - Active roles correspond to states of an object
- ▶ Role playing is written by overlapping a role to an object or by the plays-a relation



# What are Roles?

- 13 ▶ Roles are *services* of an object *in a context*
  - Roles can be connected to each other, just as services are connected to client requests
- ▶ Roles are *founded*, i.e., tied to *collaborations* and form *role models*
- ▶ A role model captures an *area of concern* (Reenskaug)



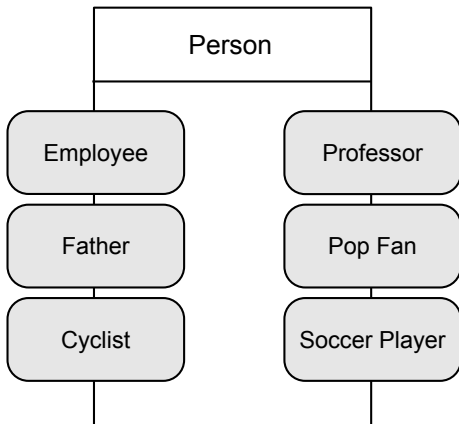
# What are Role Types?

- 14 ▶ A **role type (ability)** is a *service type* of an object
  - Role types are *dynamic view types* onto an object
  - The role type can change dynamically (*dynamic type*)
  - An object plays a role of a role type for some time
  - A role type is a *part of a protocol* of a class
    - A role is often implemented by interfaces
- ▶ A role type is *founded (relative to collaboration partner)*
- ▶ A *role model* is a set of object collaborations described by a set of role types
  - A constraint specification for classes and object collaborations
- ▶ Problem: often, we apply the word “role” also on the class level, i.e., for a “role type”



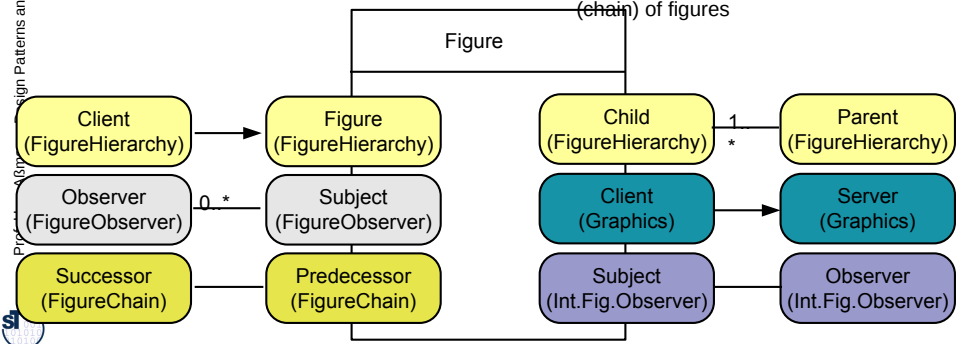
# A Class-Role-Type Diagram (Class-Ability Diagram)

- 15 ▶ Also called a **class-role model**
- ▶ Abilities (oval boxes) are put on top of classes (rectangles)
- ▶ The set of role types of a class is called its **repertoire** (*role type set*)
  - Any number of roles can be active at a time



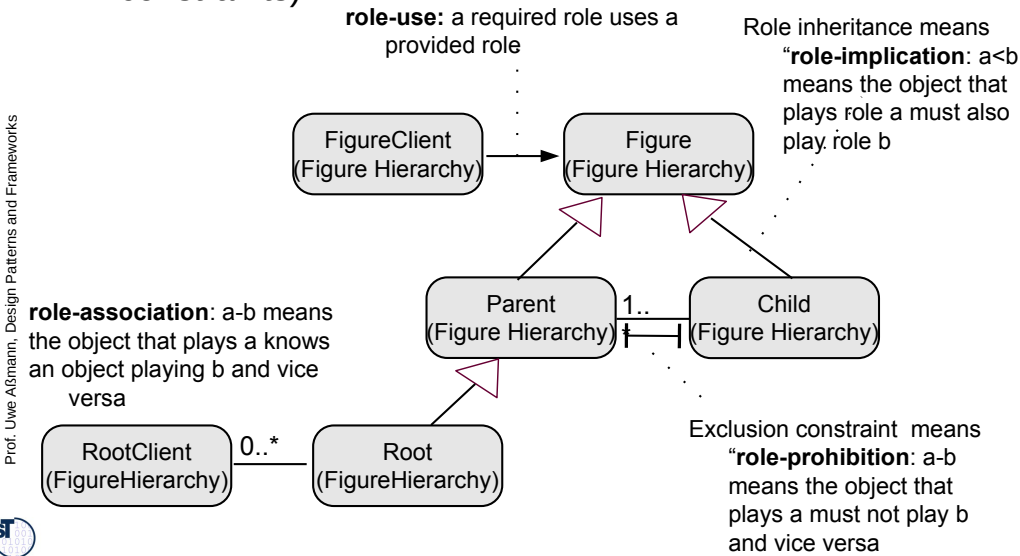
# A Class-Ability Model For Figures in a Figure Editor

- 16 ▶ A figure can play many roles in different *role models*
  - ▶ Roles may be qualified by a *role model identifier* in brackets
  - ▶ This class-role model is composed out of several simpler role models
- Explanation of some role types:
- ▶ FigureHierarchy.Figure: regular drawing functions
  - ▶ FigureHierarchy.Child: child in a figure hierarchy
  - ▶ FigureObserver.Subject: subject of a Observer pattern, for communication among figures
  - ▶ FigureHierarchy.Parent: parent in a figure hierarchy
  - ▶ IntFigObserver.Subject: subject of a Observer pattern, for communication among figures
  - ▶ FigureChain.Sucessor: sucessor in a threaded list (chain) of figures



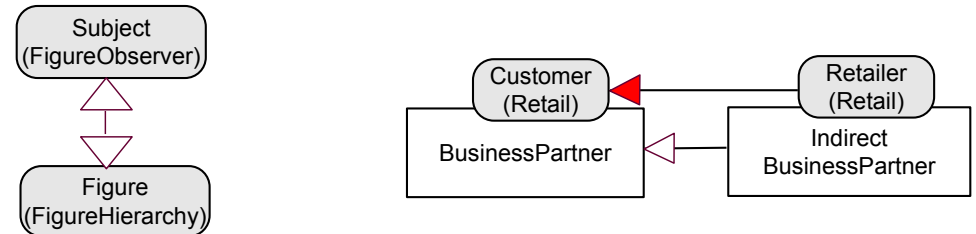
# Role Constraints in Role Models

- 17 ▶ Arrows denote constraints between roles (role constraints)



# More Role Constraints

- 18
- Bidirectional Inheritance means "role-equivalence: a<>b means the object that plays a must also play b and vice versa"
- Role-implication inheritance constraint: a role-implication constraint, stressing that the source can be mapped to a subclass of the target



# How To Develop Role Models

- 19 ▶ Ask the central question:
- Which role does my object play in this context?
  - Which responsibility does my object have in this context?
  - Which state is my object in in this context?
- ▶ If you develop with CRC cards, the questions lead to a grouping of the responsibilities (i.e., roles) on the CRC card
- Remember: a role model specifies roles of objects in context, i.e., in a specific scenario
  - Keep the role model slim, and start another one for a new scenario



# Role-Based Design with Role Models

- 20 ▶ Role-based design emphasizes *collaboration-based* design
- Starts with an analysis of the collaborations (e.g., with CRC cards)
  - Every partner of a collaboration is a role of an object
  - The role characterizes the protocol (interaction) of the object in a collaboration
- ▶ Benefit of role-based/collaboration-based design
- Roles split a class into smaller pieces
  - Roles emphasize the context-dependent parts of classes
  - Roles separate *concerns* (every role type is a concern)
  - Role models can be reused independently of classes



- ▶ Idea: why not develop with role models?

# 10.2 Composition of Role Models

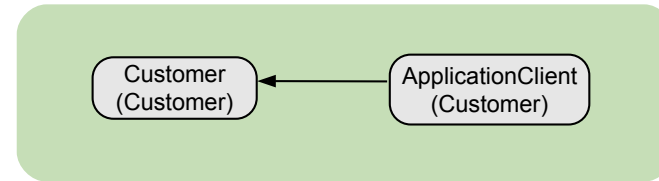
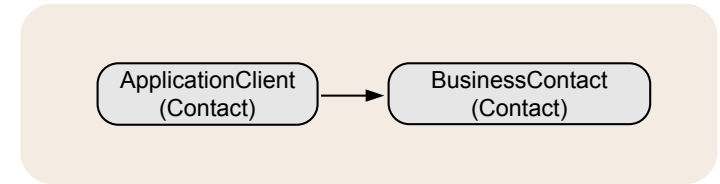
21

# Role Models of Persons in Business Applications

22



Design Patterns and Frameworks, © Prof. Uwe Aßmann



Prof. Uwe Aßmann, Design Patterns and Frameworks

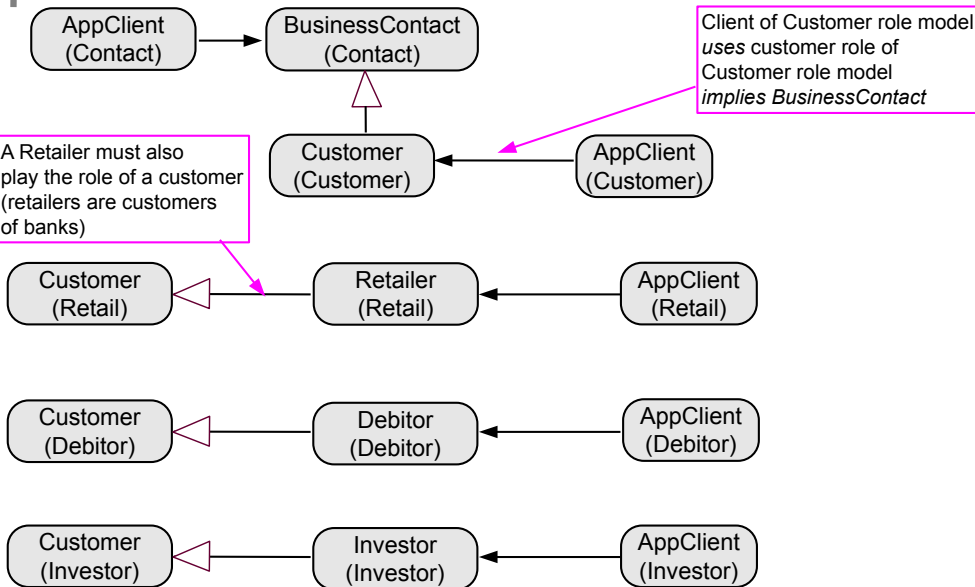


# Role Models of Persons in Business Applications

23

# Merging Role Models of Persons in Business Applications

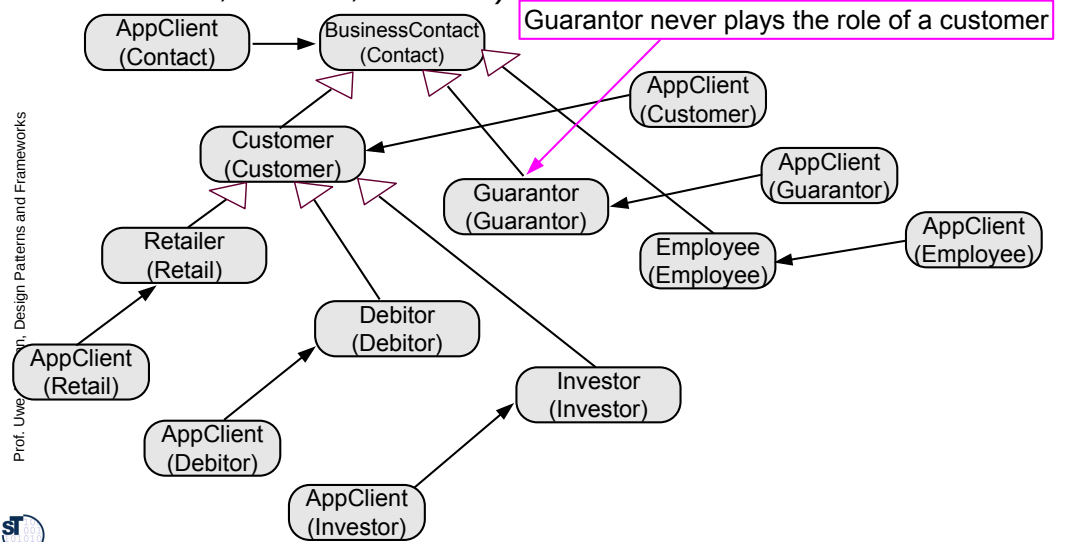
24



Prof. Uwe Aßmann, Design Patterns and Frameworks



▶ Merging role Customer from role models (Customer, Retail, Debtor, Investor)



Prof. Uwe Aßmann, Design Patterns and Frameworks



# 10.2.1 Merging Role Models into Class Diagrams

25

How role models are merged to class models

# Composing Role Models To Partial Class Diagrams

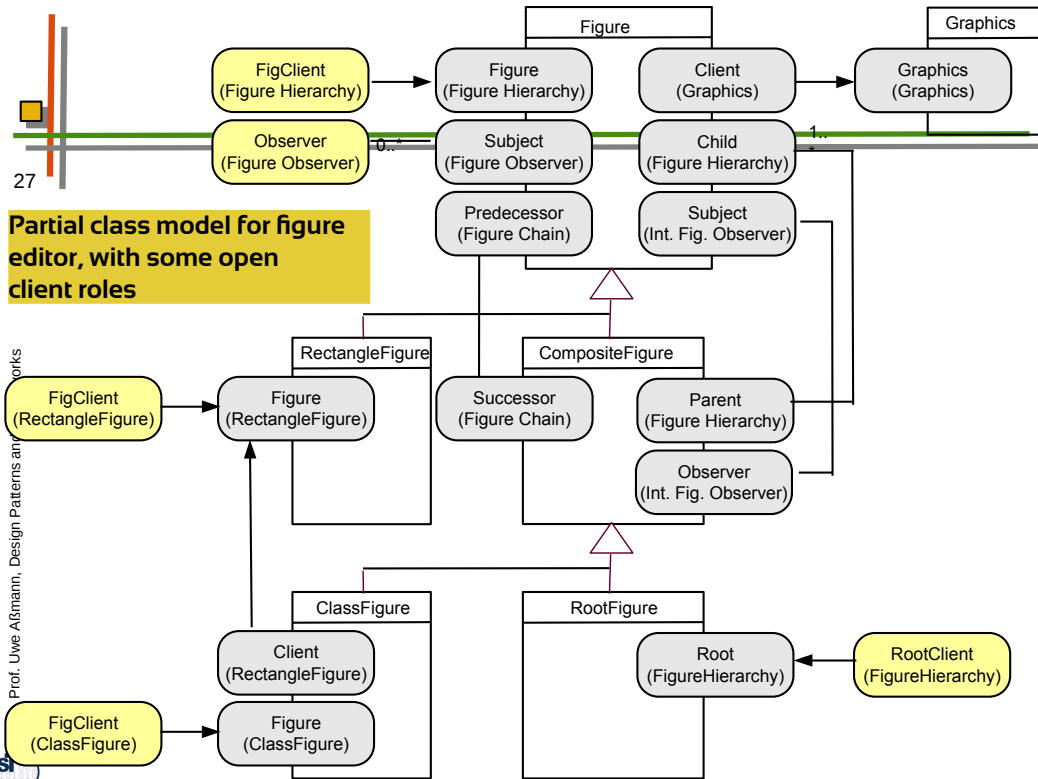
26

- ▶ Classes combine role types
  - Classes are composed of role types
  - Roles are dynamic items; classes are static items
  - So, classes group roles to form objects
- ▶ Class models combine role type models
  - Class models are composed of role models
  - One role model expresses a certain aspect of the class model
- ▶ Partial class models:
  - Role types in a role model can be left dangling (open) for further composition
  - The sub-role-models of a composed role model are called its dimensions
  - A partial class model results

Prof. Uwe Aßmann, Design Patterns and Frameworks



Design Patterns and Frameworks, © Prof. Uwe Aßmann



27

Partial class model for figure editor, with some open client roles

Prof. Uwe Aßmann, Design Patterns and Frameworks



# Role Models in the Example

28

- ▶ FigureHierarchy: composite figures (with root figure and other types, such as rectangular or class)
- ▶ FigureChain: How objects forward client requests up the hierarchy, until it can be handled
- ▶ FigureObserver: Observer pattern, for callback communication among clients and figures
- ▶ IntFigObserver: Observer pattern, for communication among figures

Prof. Uwe Aßmann, Design Patterns and Frameworks



# 10.3 Role Mapping in the MDA

29

From conceptual role models to physical class models  
 Merging and mapping role models to class models are steps of MDA [Zhao]



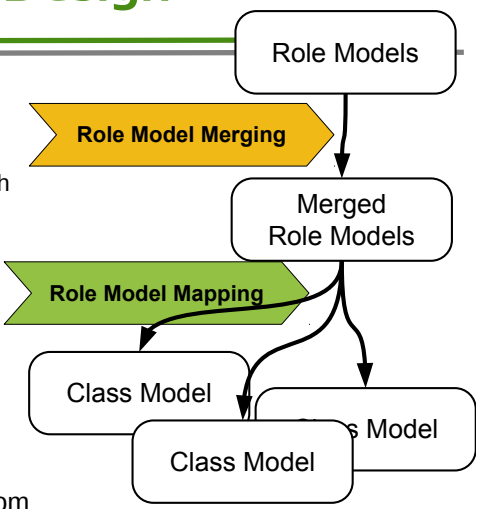
Design Patterns and Frameworks, © Prof. Uwe Aßmann

## Steps In Role-Based Design

Role Modeling

30

- First, do role models
  - Roles are all kept distinct
  - Find out about role constraints that constraint which objects execute which roles
- Secondly, compose (merge) them
  - And set up new constraints between roles of different models
- Thirdly, map role models to class diagram
  - By merging the roles to classes
  - Respecting the constraints
- Benefit: many different class models from one set of role models! (Gross variability)



Prof. Uwe Aßmann, Design Patterns and Frameworks



## The Role Mapping Process and Model-Driven Architecture (MDA)

31

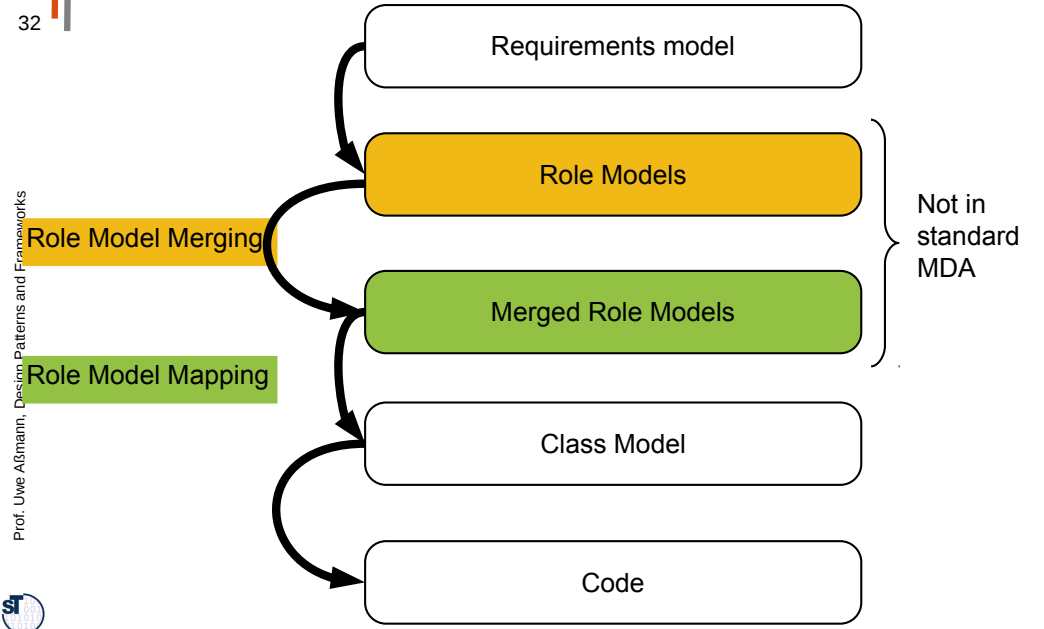
- The information which roles belong to which class can be regarded as a *platform information*
- A role model is more *platform independent* than a class model
  - The decision which roles are merged into which classes has not been taken and can be reversed**
  - We say: roles are *logical (conceptual)*, classes are *physical*
- In MDA, role models are found on a more platform independent level than class models
  - First design a set of role models
  - Then find a class model by mapping roles into classes
  - Respect role constraints
  - Usually, several class models are legal

Prof. Uwe Aßmann, Design Patterns and Frameworks



## Role Model Mapping is a Task in MDA

32



Prof. Uwe Aßmann, Design Patterns and Frameworks





# The Influence of the Role Constraints on Role Model Mapping

- 33 ▶ *Role-equivalent constraint*: strong constraint: same implementation class
- ▶ *Role-implication constraint*: weaker, leaves freedom, which physical class implements the roles
  - Map to same classes or subclasses
  - If implemented by the same class, the class model is stricter than the role model
  - Embedding roles in a class reduces the number of runtime objects, hence more efficient, less object schizophrenia
  - Split classes allow for better exchange of a role at runtime, since only the runtime object needs to be exchanged
- ▶ *Role-implication inheritance constraint*: a role-implication constraint, stressing that the source must be mapped to a subclass of the target
- ▶ *Role-use constraint*: translation to delegation possible (different classes)



# 10.4 Implementing Abilities By Hand

35

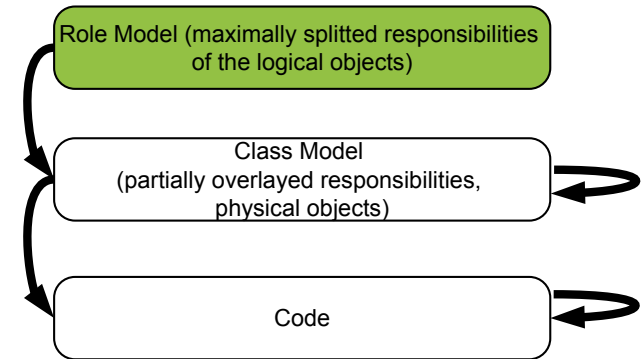


# Computing Physical Objects by Role Mapping

- 34 ▶ The role mapping process determines, which physical object inherits from which role-interface
- ▶ The role mapping *computes* the physical objects from maximal splits of the logical objects



Role model mapping



# Implementation of Abilities

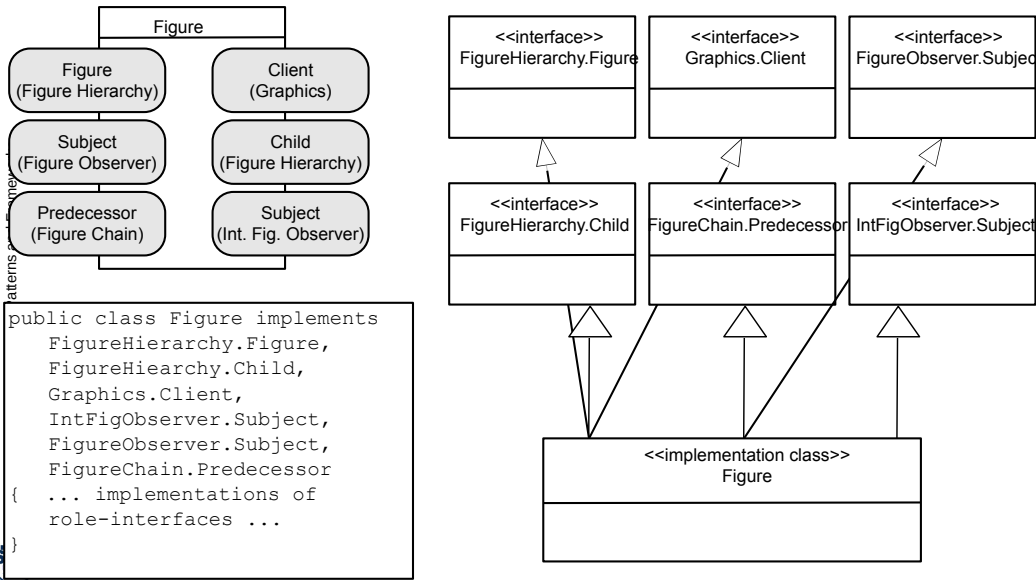
36 Abilities can be mapped into classes (role mapping) in several ways:

- ▶ With interfaces
  - Then, code for the interfaces must be written by hand
- ▶ With multiple inheritance
  - Then, there are two layers of classes: role classes and standard classes
- ▶ With mixin classes
  - Some language allow for composing “mixin” classes into classes
    - CLOS, Scala
    - “include inheritance” (Eiffel, Sather)
  - A role is like a mixin class
  - No code has to be written by hand
- ▶ With multi-Bridges



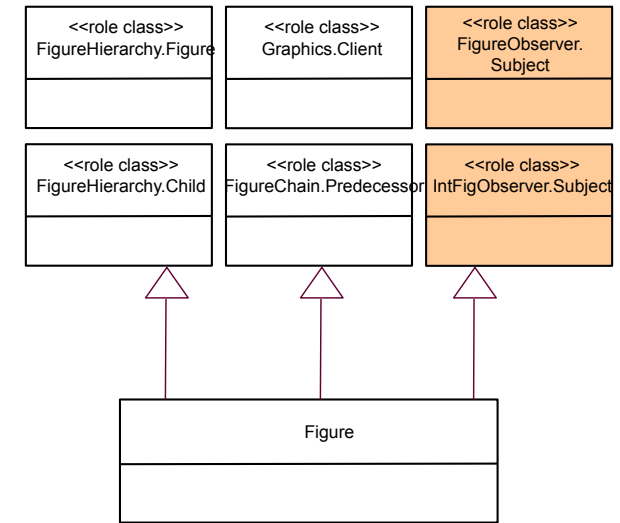
## With Interfaces

- 37 ▶ Then, code for the interfaces must be written by hand



## With Multiple Inheritance

- 38 ▶ Then, there are two layers of classes: role classes and standard classes
- ▶ A standard class must inherit from several role classes
- ▶ Disadvantage: a standard class can inherit from a role class only once

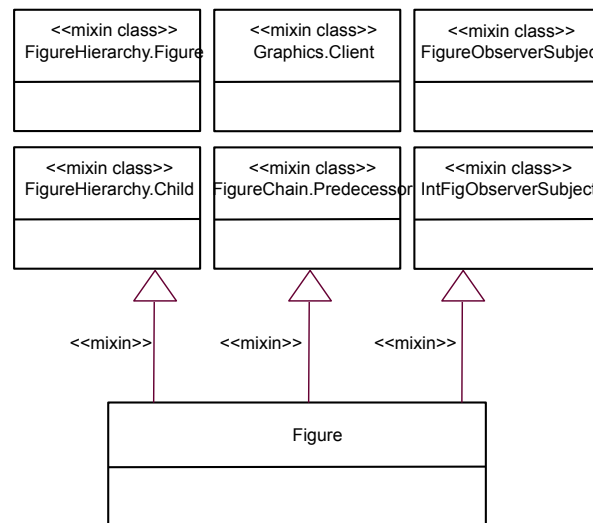


## With Mixin Classes

- 39 ▶ Some language allow for composing "mixin" classes into classes

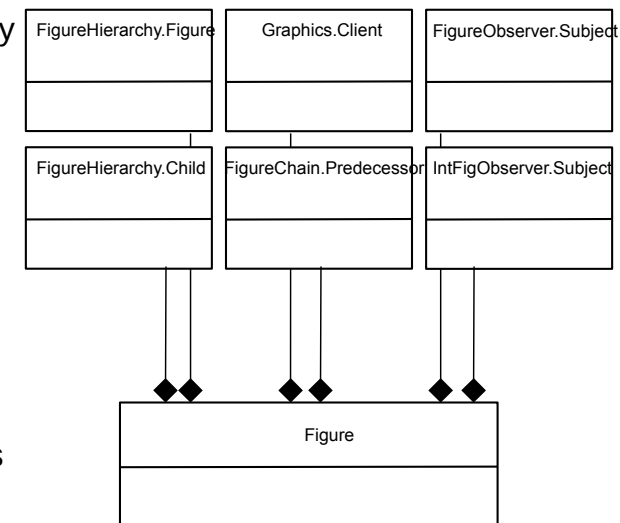
- CLOS, Scala
- "include inheritance" (Eiffel, Sather)

- ▶ A mixin is a superclass parameterizing a generic super declaration of a base class
- ▶ A role type is like a mixin class
- ▶ Role code can be inherited
- ▶ Features of a mixin are renamed, if it is inherited a second time



## Implementation With Multi-Bridges and "Role Objects"

- 40 ▶ A *role object* represents only one role
- ▶ A *role class* only one role type
- ▶ The implementation pattern has a core object that aggregates all role objects
- ▶ Also with "Role Object Pattern" (later)
- ▶ Bridge and Multi-Bridge are typical role implementations



# Connecting Role Behavior with Embedding Context

41

- ▶ The body of an ability must be embedded into the control- and data-flow of the context code of the class.
- ▶ Wrapper/Decorator:
  - If an ability is implemented as Wrapper (Decorator), it intercepts the control flow inward and outward of a method or class
  - Then, roles can be stacked at run-time (Decorator list)
- ▶ Input Filter/Interceptor:
  - Then the role code is executed before the method or the methods of a class
- ▶ Output Filter:
  - Then the role code is executed after the method or the methods of a class



# The Difference of Roles and Facets

42

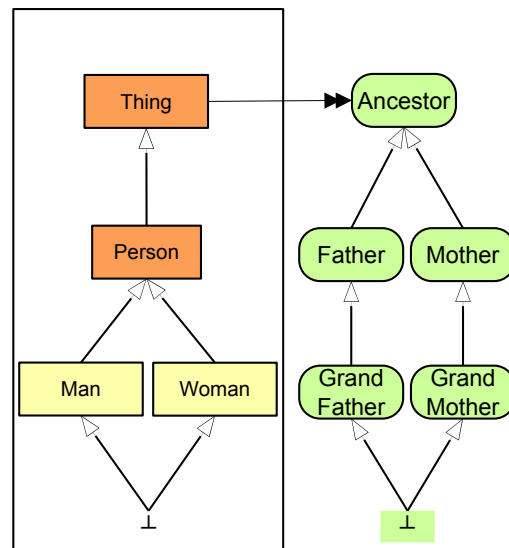
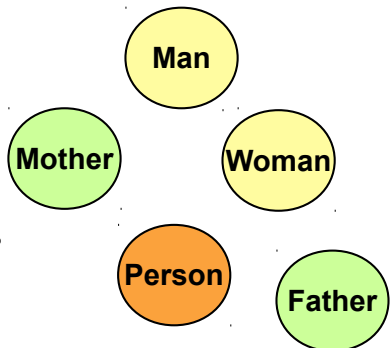
- ▶ A faceted class is a class with n dimensions
- ▶ If the facet has a collaboration partner, it turns out to be a role
  - Each facet is a role type
  - Role types are independent of each other
  - However, the role type is static, not dynamic: facets are lasting



# Solution to the Little Riddles..

43

- ▶ Mother and Father are abilities of classes
- ▶ Man and Woman are facets
- ▶ Person is a natural type



# 10.4.1. Example of Roles of Persons in Business Applications

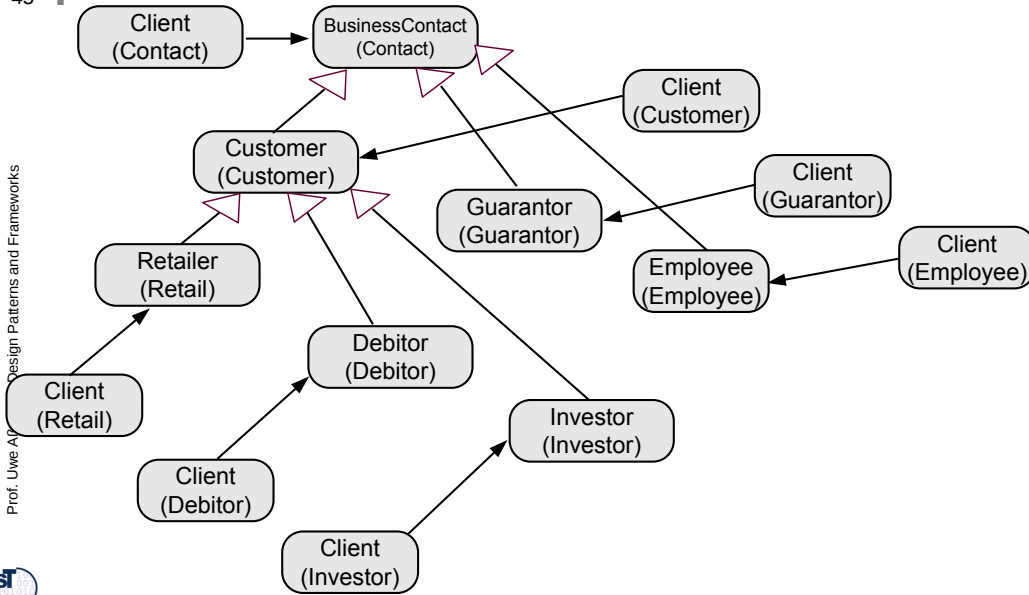
44



# Role Models of Persons

45

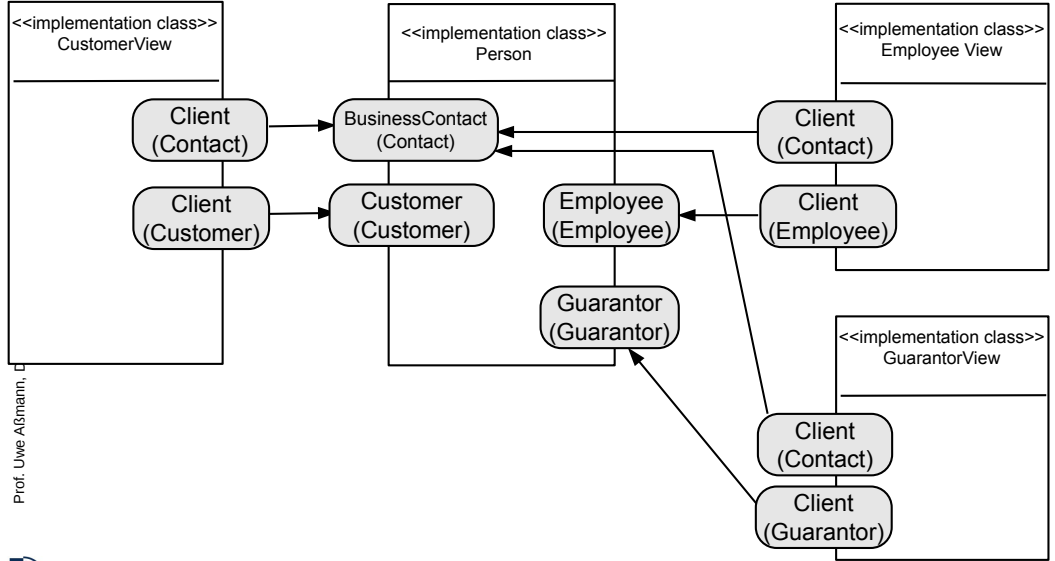
Prof. Uwe Aßmann, Design Patterns and Frameworks



# Implementation With Interfaces (or Mixins)

46

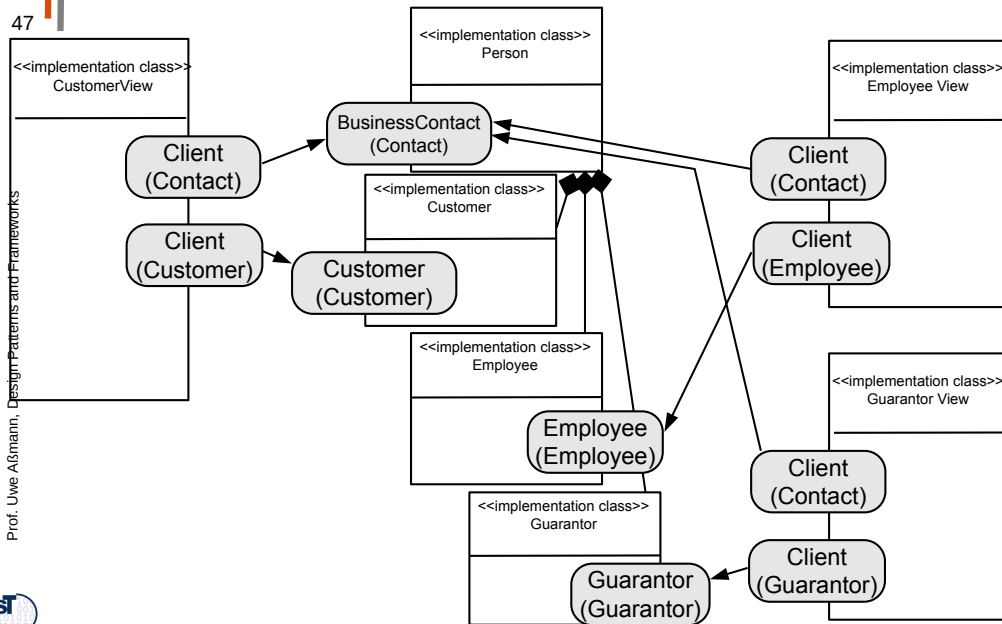
Prof. Uwe Aßmann,



# Implementation of Person With Multi-Bridge (Role Objects)

47

Prof. Uwe Aßmann, Design Patterns and Frameworks



# 10.4.2 Example: Actors, Films, and Directors

48



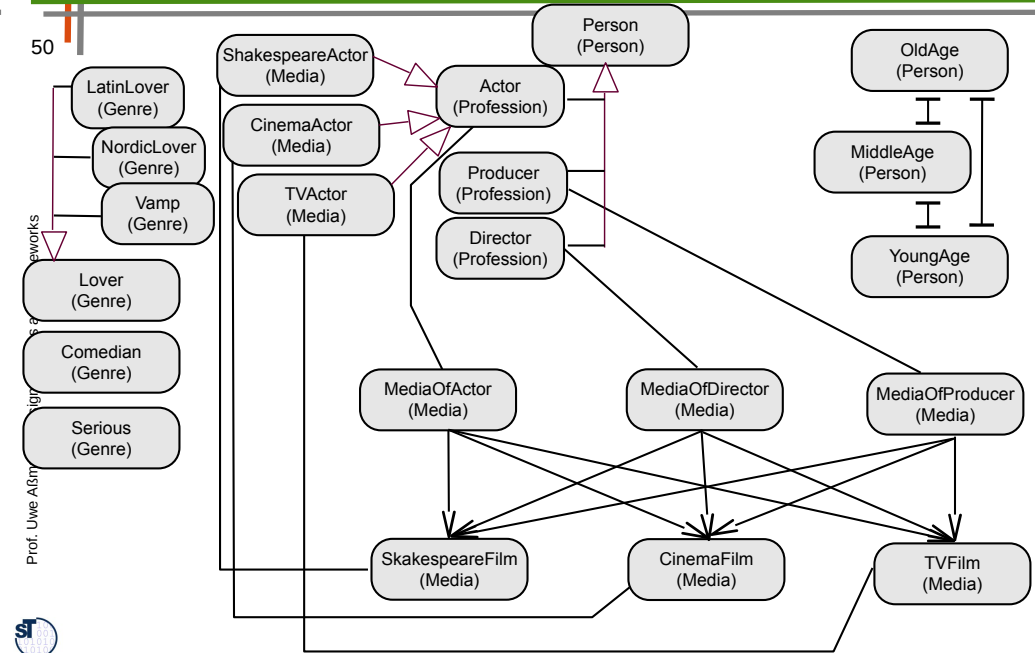
# Actors, Films, and Directors

- ▶ We model actors, directors, producers, and their films
- ▶ Actors have a genre (lover, serious, comedian) and play on a certain media (TV, cinema, Shakespeare)
- ▶ Directors and producers have similar attributes
- ▶ Films also
- ▶ Actors have an age (young, medium, old)

49



# Example Role Model for Actors



50



# There are Many Ways to Implement This Role Model

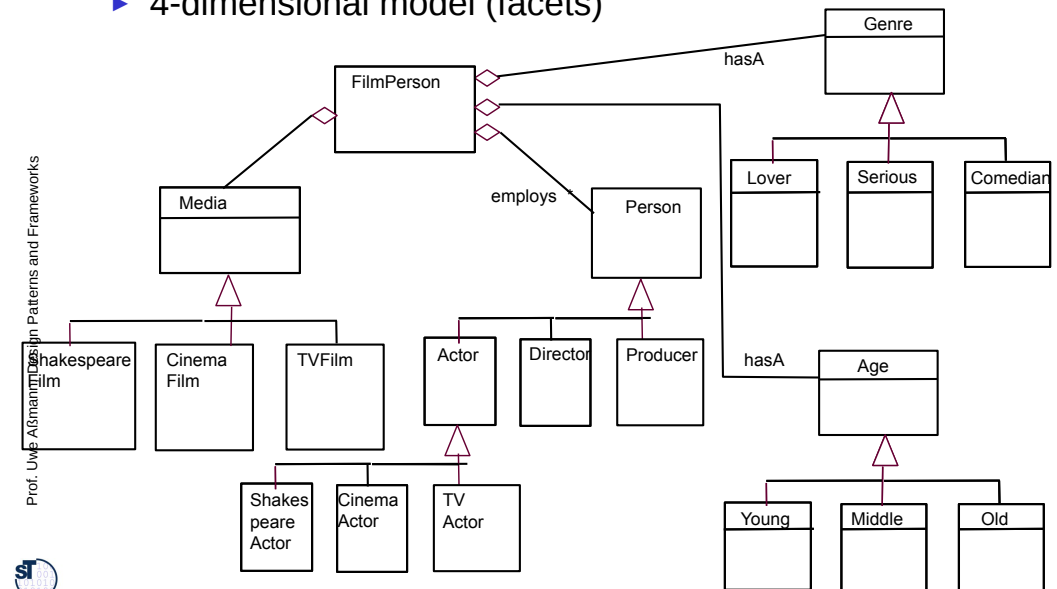
- ▶ With a facet based model, modelling some role models as class hierachies of a Dimensional Hierarchies model

51



# Very Simple Class Model for Actors and Films

- ▶ 4-dimensional model (facets)



52



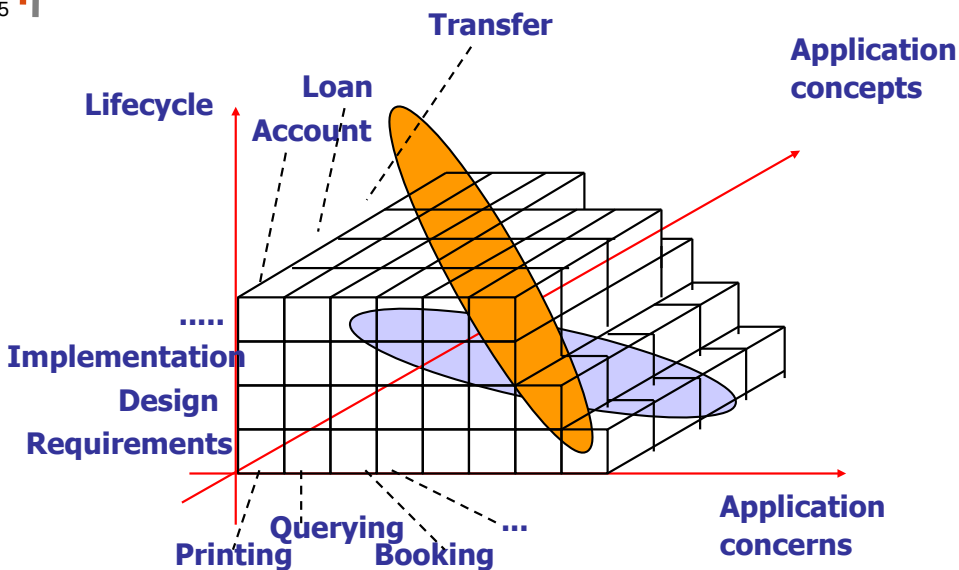
Some roles implemented by overlaying, inheritance, delegation, and role objects

53



## Hyperslices are Named Slices Through the Concern Matrix

55



## 10.5 More on Roles

54

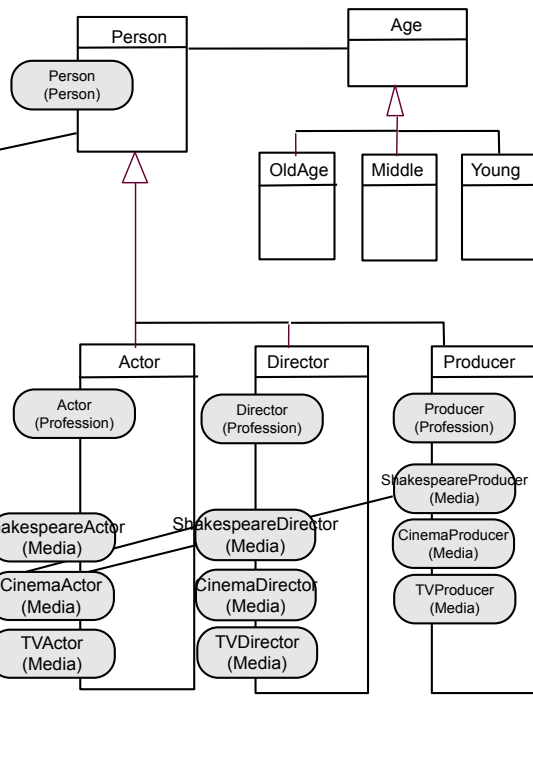
### 10.5.1 Relation of Role Modelling to Other Software Engineering Technologies

## Hyperslice Composition and Role Mapping

56



- ▶ Hyperslices (views) are essentially the same concept as role models
  - But work also on other abstractions than classes and feature sets
  - Hyperslices can be defined on statements and statement blocks
  - Role models are more unstructured since they do not prerequisite slices, dimensions, or layers
- ▶ Hyperslice composition is similar to role mapping
  - Is guided by a composition that merges views (roles)
  - Hyperslices are independent (no constraints between hyperslices)
- ▶ Role models implement aspects
  - Because the roles are related by role constraints
- ▶ More in “Component-based Software Engineering”



## Roles vs Facets

- 57
- ▶ A facet is concerned always with *one* logical object
    - A facet classification is a *product lattice*
  - ▶ Role models may *crosscut many objects*
    - They are concerned with collaboration of at least 2 objects
    - Hence, a facet is like a role of one object, but from n facet dimensions.
    - A class can have arbitrarily many roles, but only n facets
  - ▶ Roles may be played for some time; facets must have a facet value the entire lifetime of the object



## Rigid Types

If an object that has a (*semantically*) *rigid* type, it cannot stop being of the type without losing its identity

- 59
- ▶ Example:
    - A Book is a rigid type
    - A Reader is a non-rigid type
    - A Reader can stop reading, but a Book stays a Book
  - ▶ A *semantically rigid type* is  *tied to the identity* of objects
    - A semantically rigid type is tied to a class invariant (holds for all objects at all times)
  - ▶ A *semantically non-rigid type* is a dynamic type that is indicating a state of the object



## 10.5.2 Role Types Formally

58



## Founded Types

- 60
- ▶ A *founded type* is a type if an object of the type is always in collaboration (association) with another object.
    - Example: Reader is a founded type because for being a reader, one has to have a book.

A *role type (ability)* is a founded and non-rigid type  
Role types (abilities) are in collaboration and if the object does no longer play the role type, it does not give up identity

*Natural types* are non-founded and semantically rigid.  
Book is a natural type.  
A natural type is *independent* of a relationship  
The objects cannot leave it



# The End: Summary

- 61
- ▶ Role-based modelling is more general and finer-grained than class-based modelling
  - ▶ Role mapping is the process of allocating roles to concrete implementation classes
  - ▶ Hence, role mapping decides how the classes of the design pattern are allocated to implementation classes (and this can be quite different)
  - ▶ Roles are important for design patterns
    - If a design pattern occurs in an application, some class of the application plays the role of a class in the pattern
  - ▶ Roles are dynamic classes: they change over time (non-rigid) and are context-dependent (founded)

