Part II
Design Patterns and Frameworks

Prof. Dr. U. Aßmann

Chair for Software Engineering

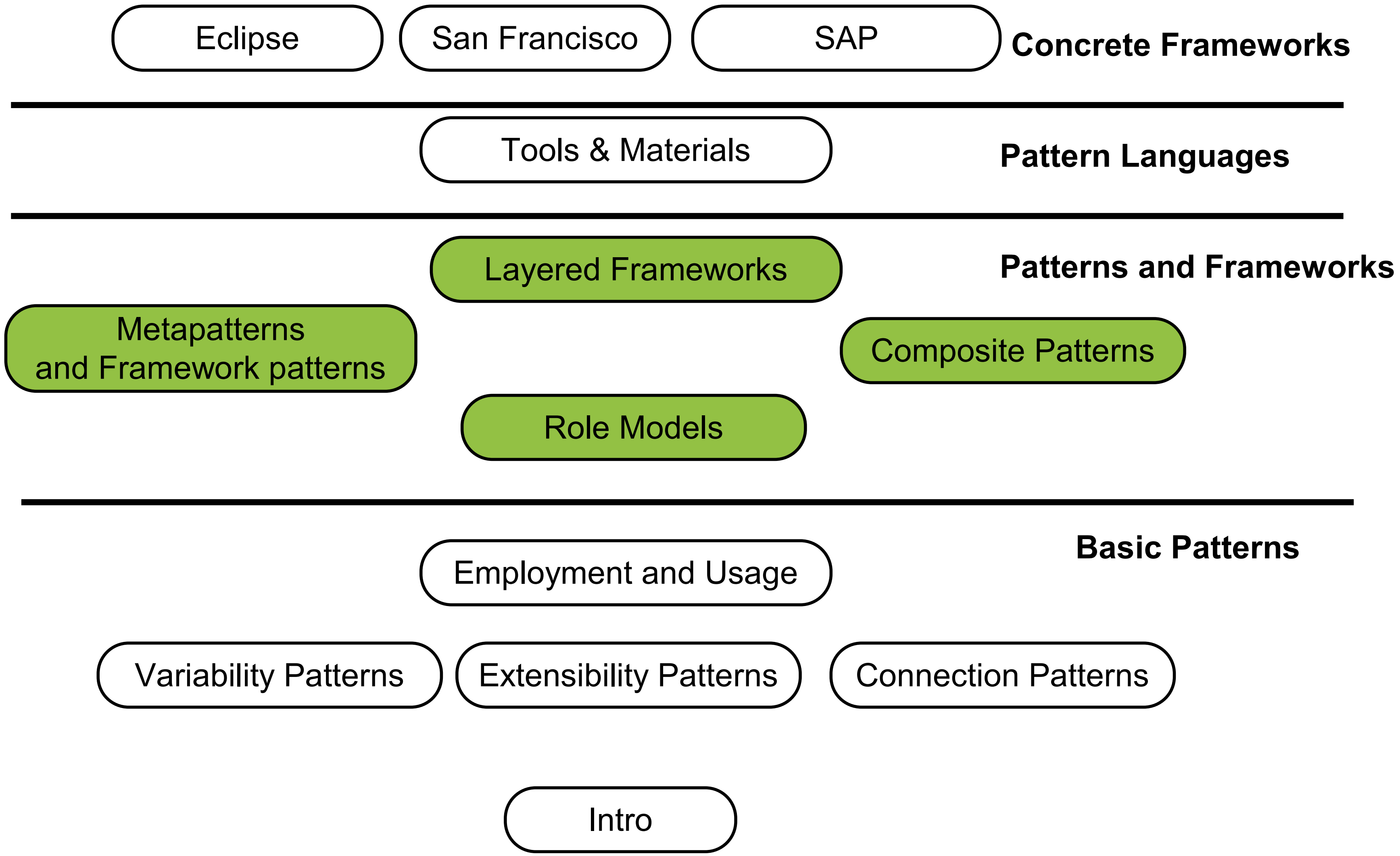
Faculty of Informatics

Dresden University of Technology

WS17/18, Nov 20, 2017

Lecturer: Dr. Sebastian Götz

Overview of the Course



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

Prof. Dr. U. Aßmann

Chair for Software Engineering

Faculty of Informatics

Dresden University of Technology

Literature (To Be Read)

- D. Riehle, T. Gross. **Role Model Based Framework Design and Integration**. Proceedings of Conference on Object-oriented Programing Systems, Languages, and Applications (OOPSLA), ACM Press, 1998.
  - http://dl.acm.org/citation.cfm?id=286951
- 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, MDAFA 2003 and MDAFA 2004, Lecture Notes in Computer Science, Volume 3599, 2005, DOI: 10.1007/11538097
  - http://link.springer.com/chapter/10.1007%2F11538097_1

Other Literature

- T. Reenskaug, P. Wold, O. A. Lehne. **Working with objects.** Manning publishers. 2001.
  - The OOram Method, introducing role-based design, role models and many other things. A wisdom book for design.
  - http://heim.ifi.uio.no/~trygver/1996/book/book11d.pdf
- H. Allert, P. Dolog, W. Nejdl, W. Siberski, F. Steimann. ***Role-Oriented Models for Hypermedia Construction – Conceptual Modelling for the Semantic Web.***
  - http://people.cs.aau.dk/~dolog/pub/ht2003.pdf

Other Literature

- 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

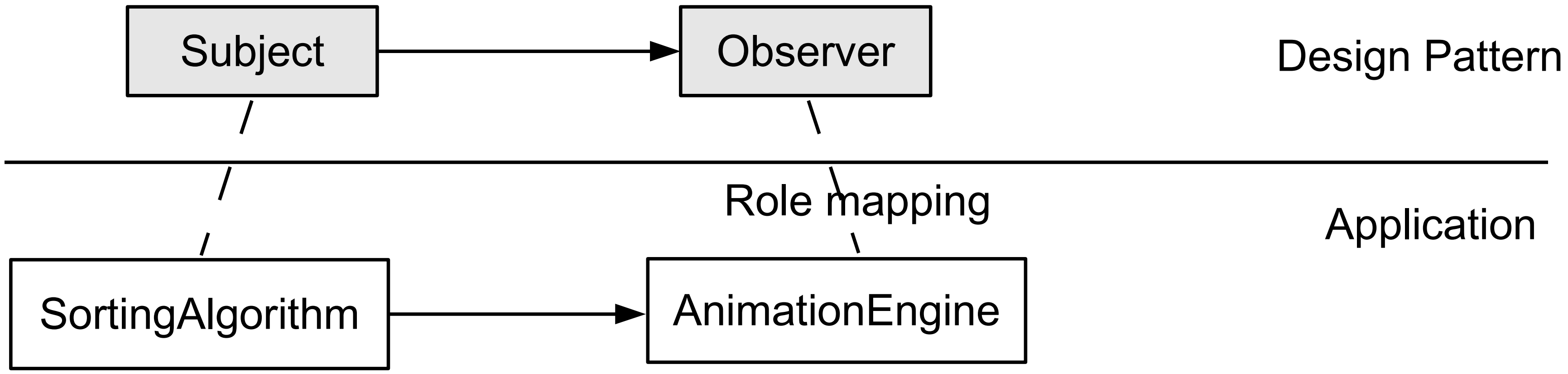
Goals

- 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
- Understand composite design patterns
  - Understand how to mine composite design patterns
- Understand role types as semantically non-rigid and founded
- Understand layered frameworks as role models
- Understand how to optimize layered frameworks and design patterns

10.1 Role-based Design With Role Models

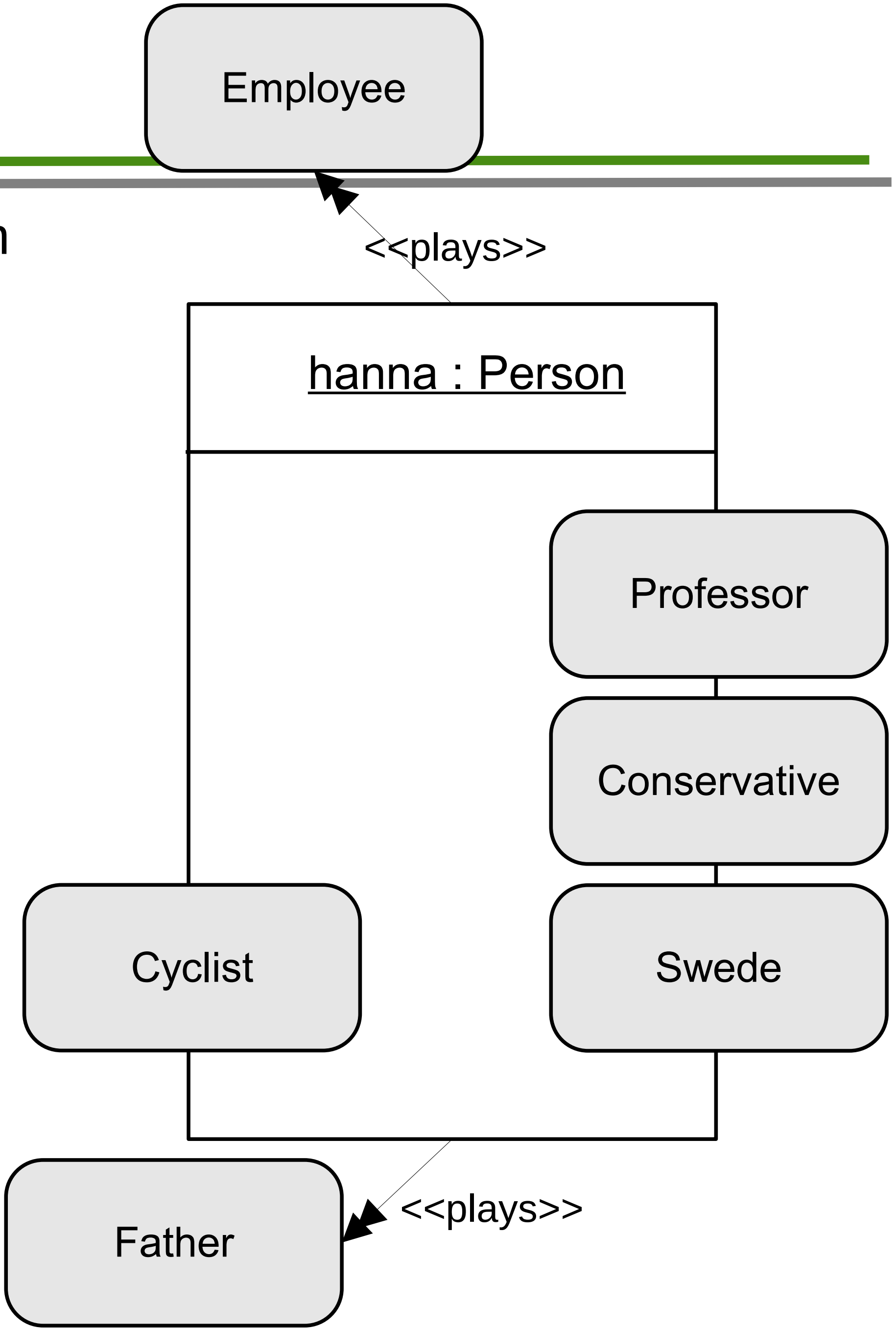
Purpose of Teaching Role-based Design

- 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



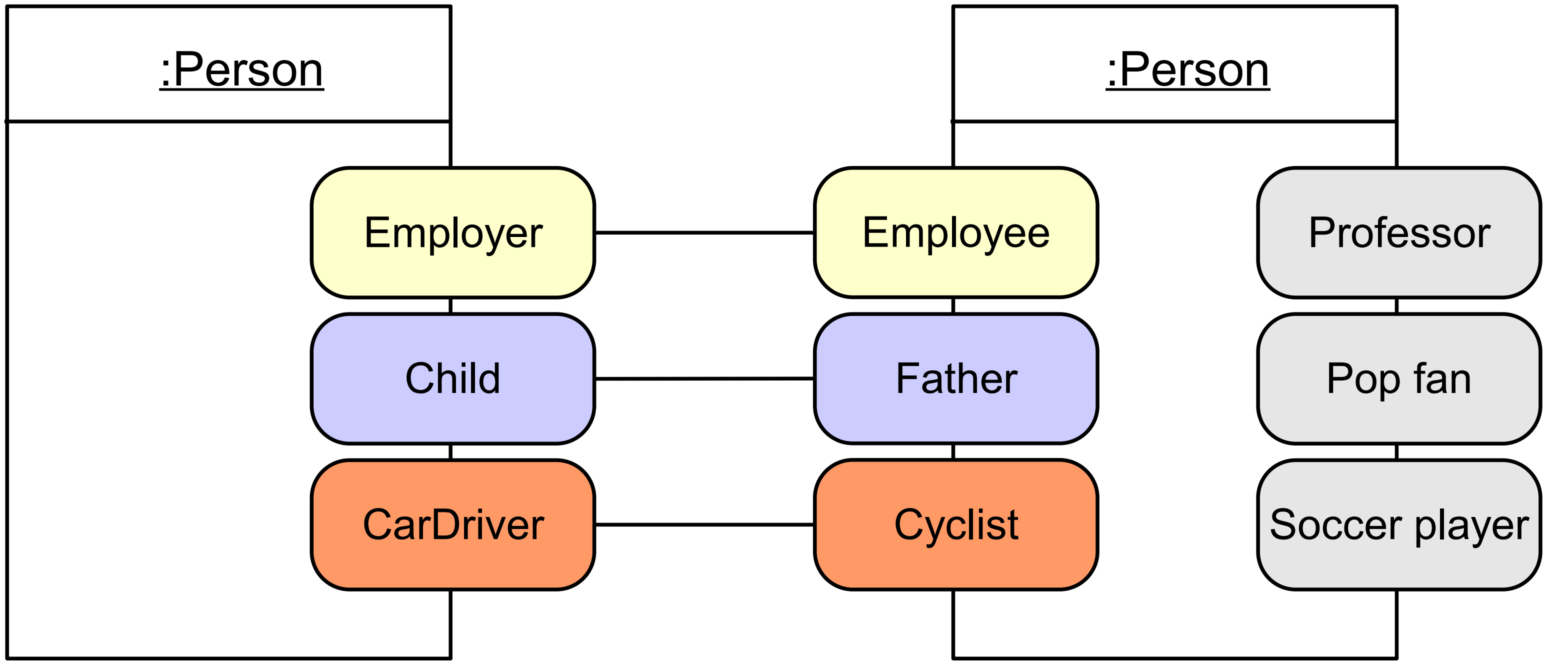
What are Roles?

- A *role* is a *dynamic view* onto an object
  - The view can change dynamically
  - A role of an object belongs to an 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 denoted by overlapping a role to an object or by the plays relation



What are Roles?

- 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)



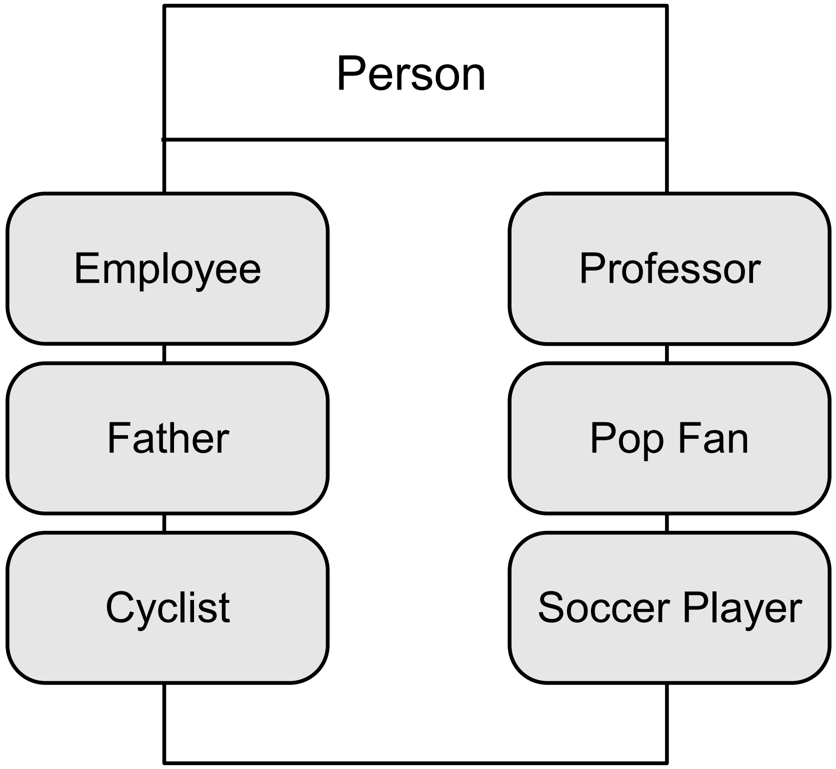
What are Role Types?

- 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

- 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

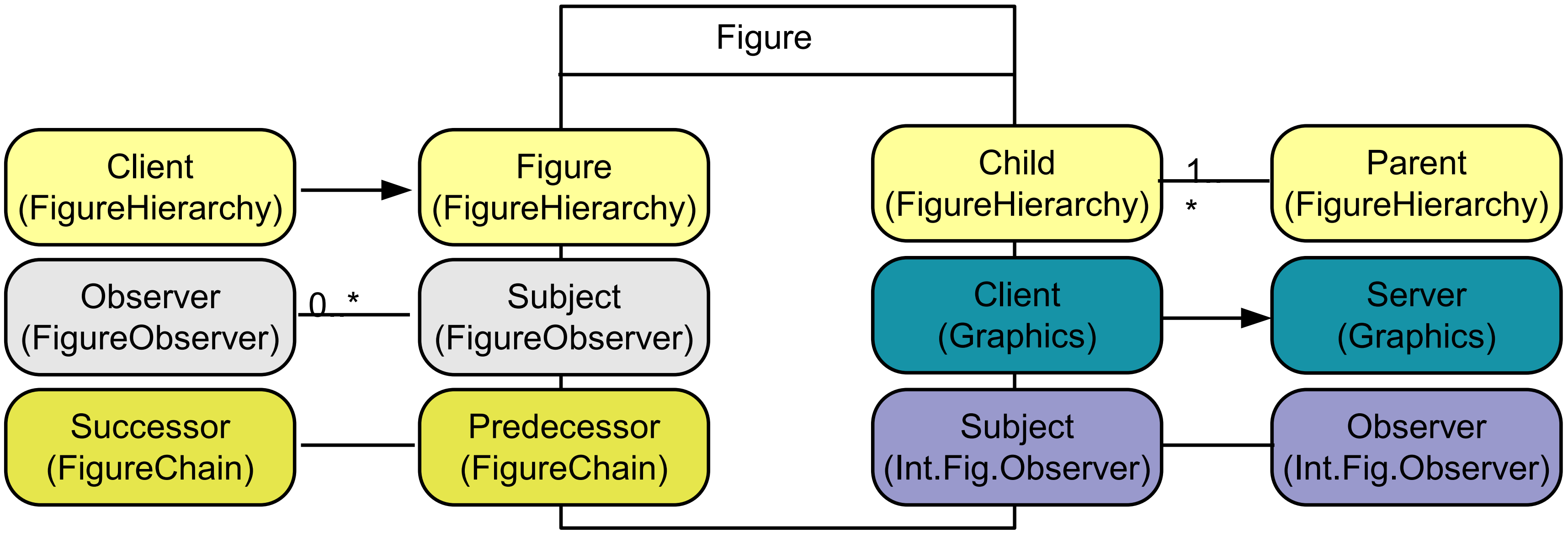


A Class-Role-Type Model For Figures in a Figure Editor

- 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

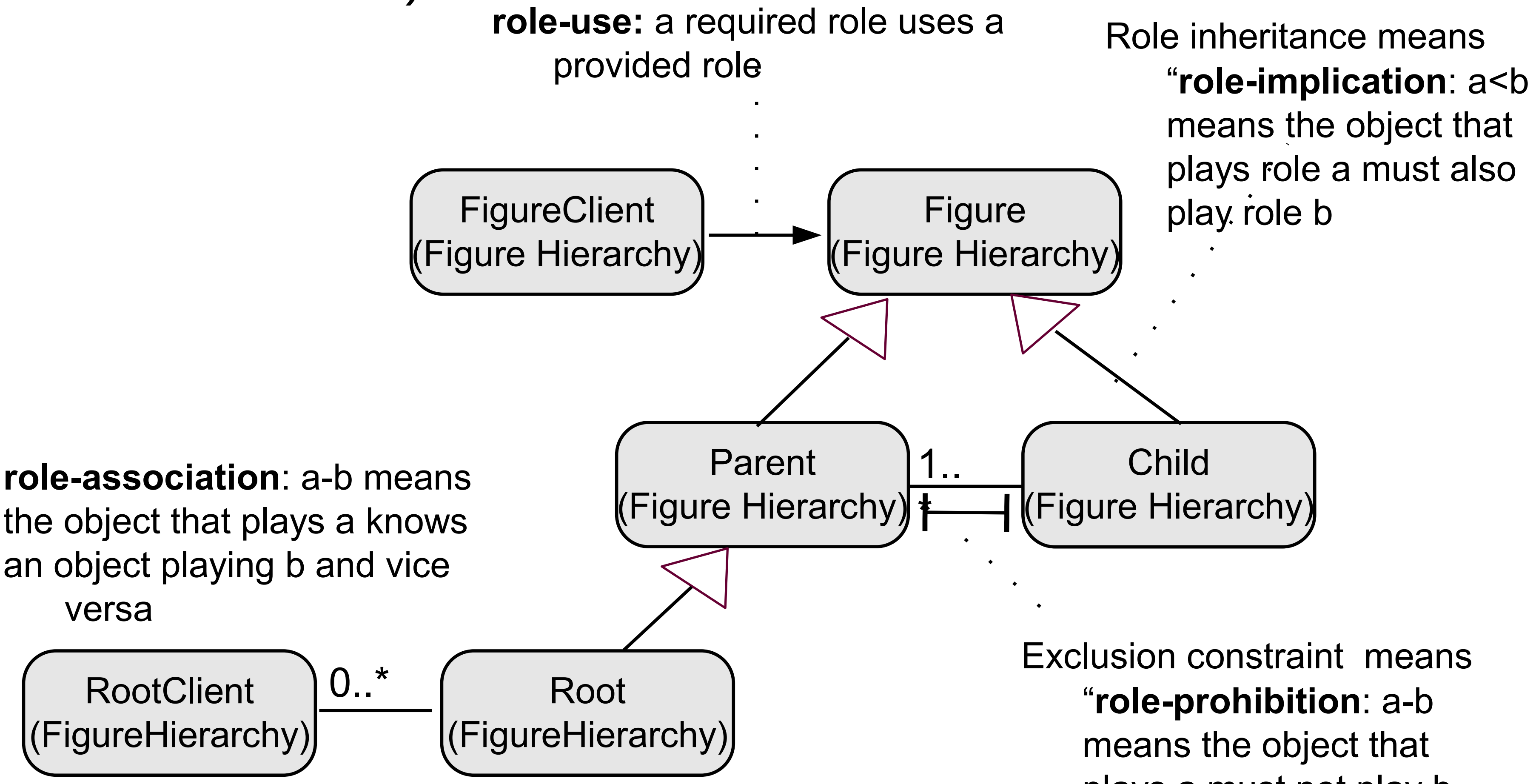
Explanation of some role types:

- 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

- Arrows denote constraints between roles (role constraints)


FigureClient (Figure Hierarchy)
Figure (Figure Hierarchy)
Parent (Figure Hierarchy)
Child (Figure Hierarchy)
1..*
RootClient (FigureHierarchy)
0..*
Root (FigureHierarchy)

role-use: a required role uses a provided role

Role inheritance means "**role-implication**: a<b means the object that plays role a must also play role b

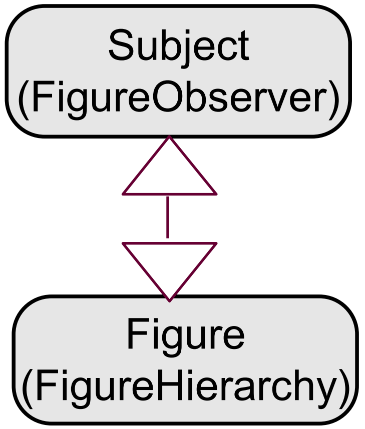
role-association: a-b means the object that plays a knows an object playing b and vice versa

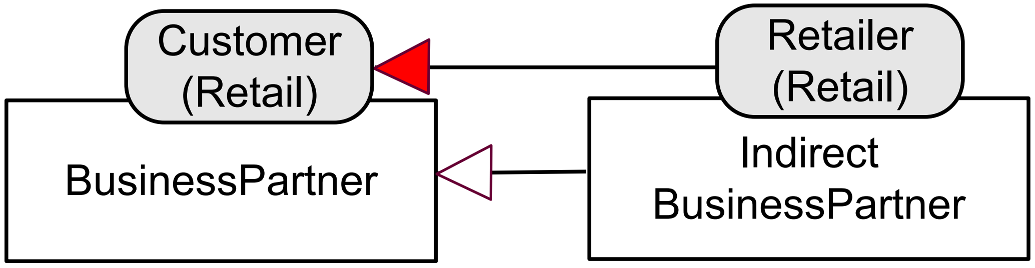
Exclusion constraint means "**role-prohibition**: a-b means the object that plays a must not play b and vice versa

More Role Constraints

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

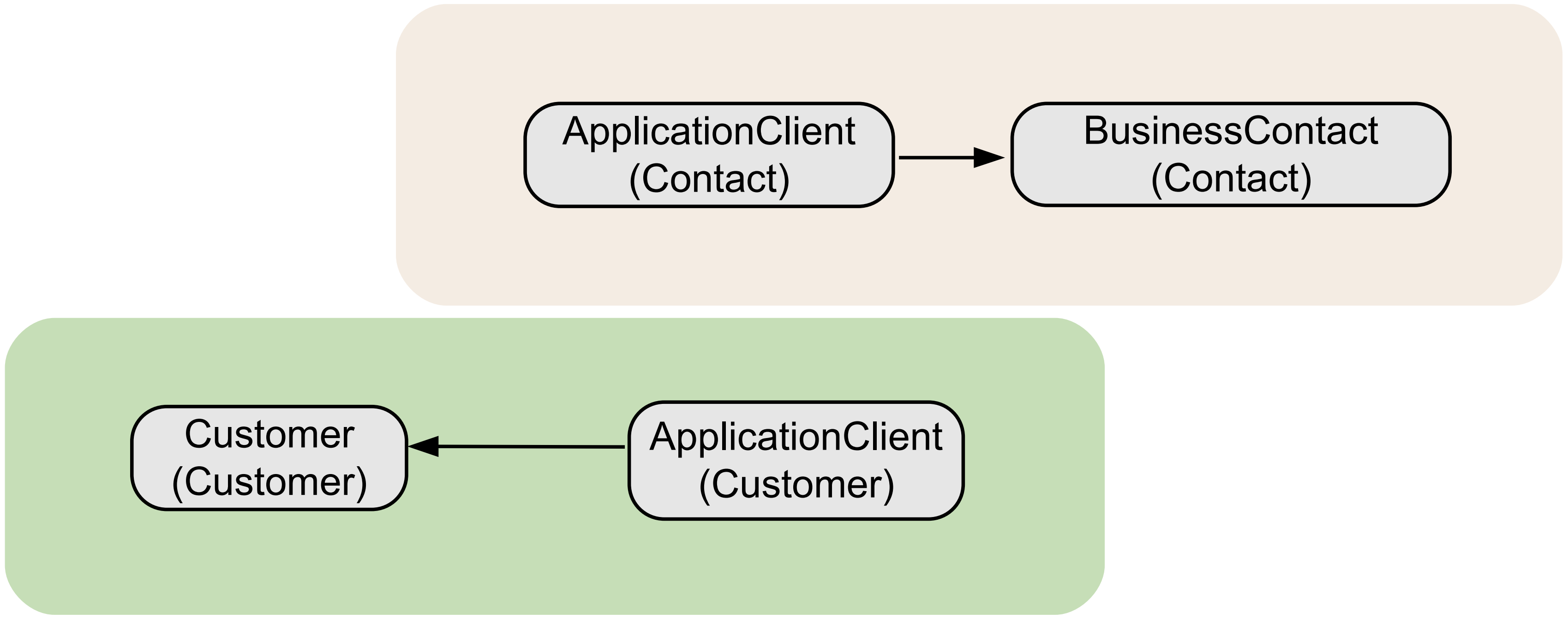
- Ask the central question:
  - Which role does my object play in this context?
  - Which responsibility does my object have in this context?
  - Which state is my object in in this context?
- If you develop with CRC cards, the questions lead to a grouping of the responsibilities (i.e., roles) on the 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

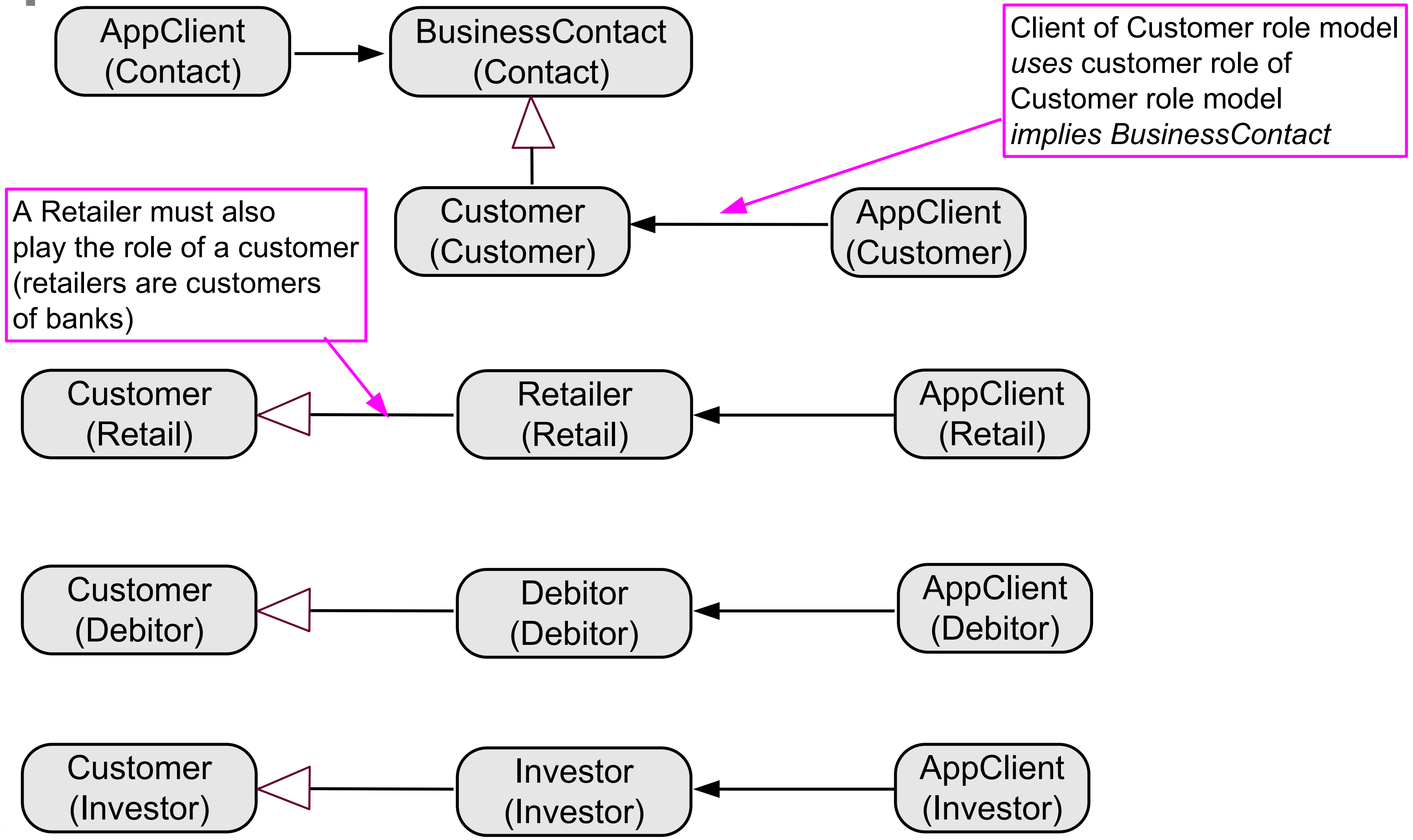
- Role-based design emphasizes *collaboration-based* design
  - Starts with an analysis of the collaborations (e.g., with CRC cards)
  - Every partner of a collaboration is a role of an object
  - The role characterizes the protocol (interaction) of the object in a collaboration
- Benefit of role-based/collaboration-based 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

Role Models of Persons in Business Applications


ApplicationClient (Contact)
BusinessContact (Contact)
Customer (Customer)
ApplicationClient (Customer)

Role Models of Persons in Business Applications

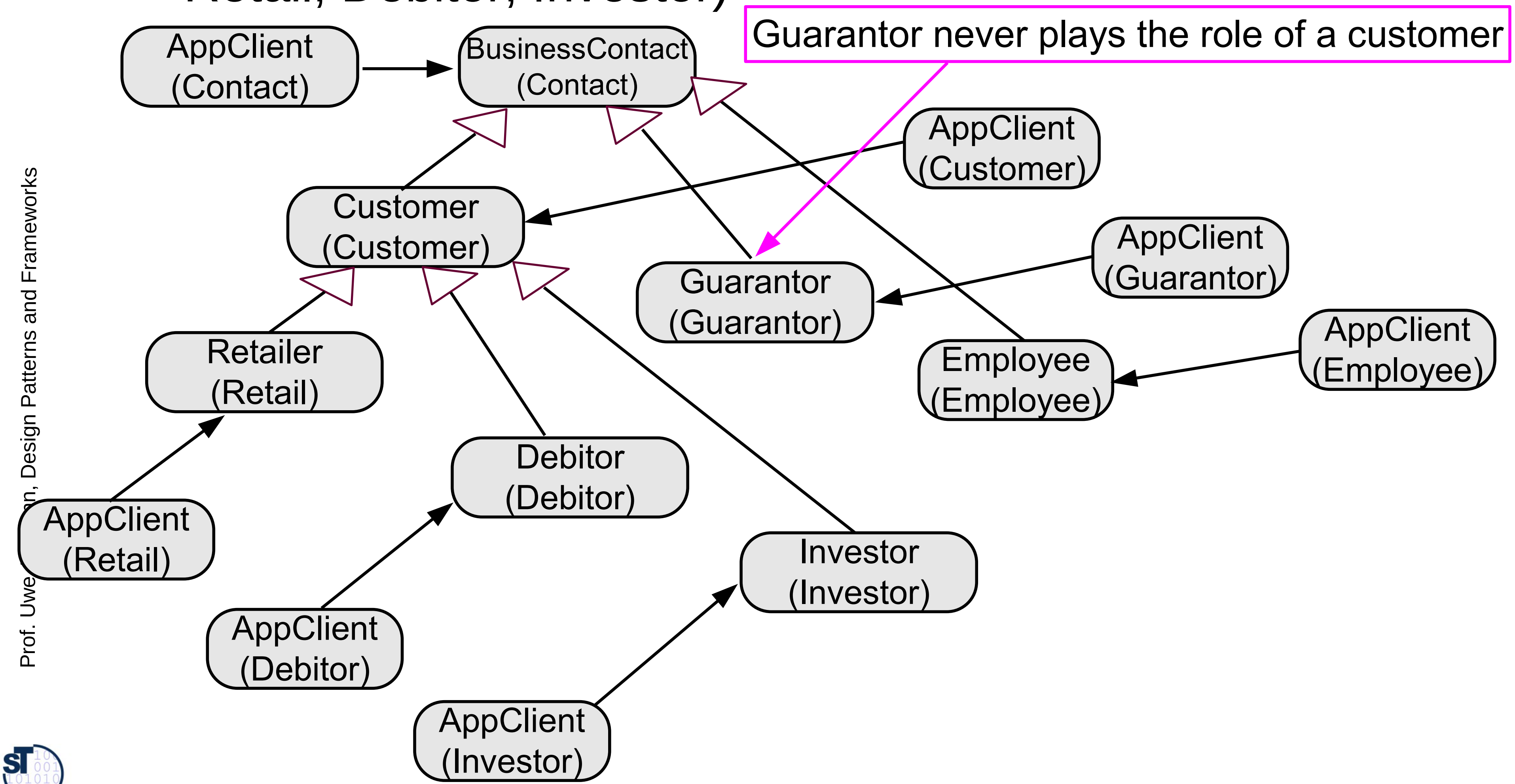


Client of Customer role model *uses* customer role of Customer role model *implies BusinessContact*

A Retailer must also play the role of a customer (retailers are customers of banks)

Merging Role Models of Persons in Business Applications

- Merging role Customer from role models (Customer, Retail, Debitor, Investor)



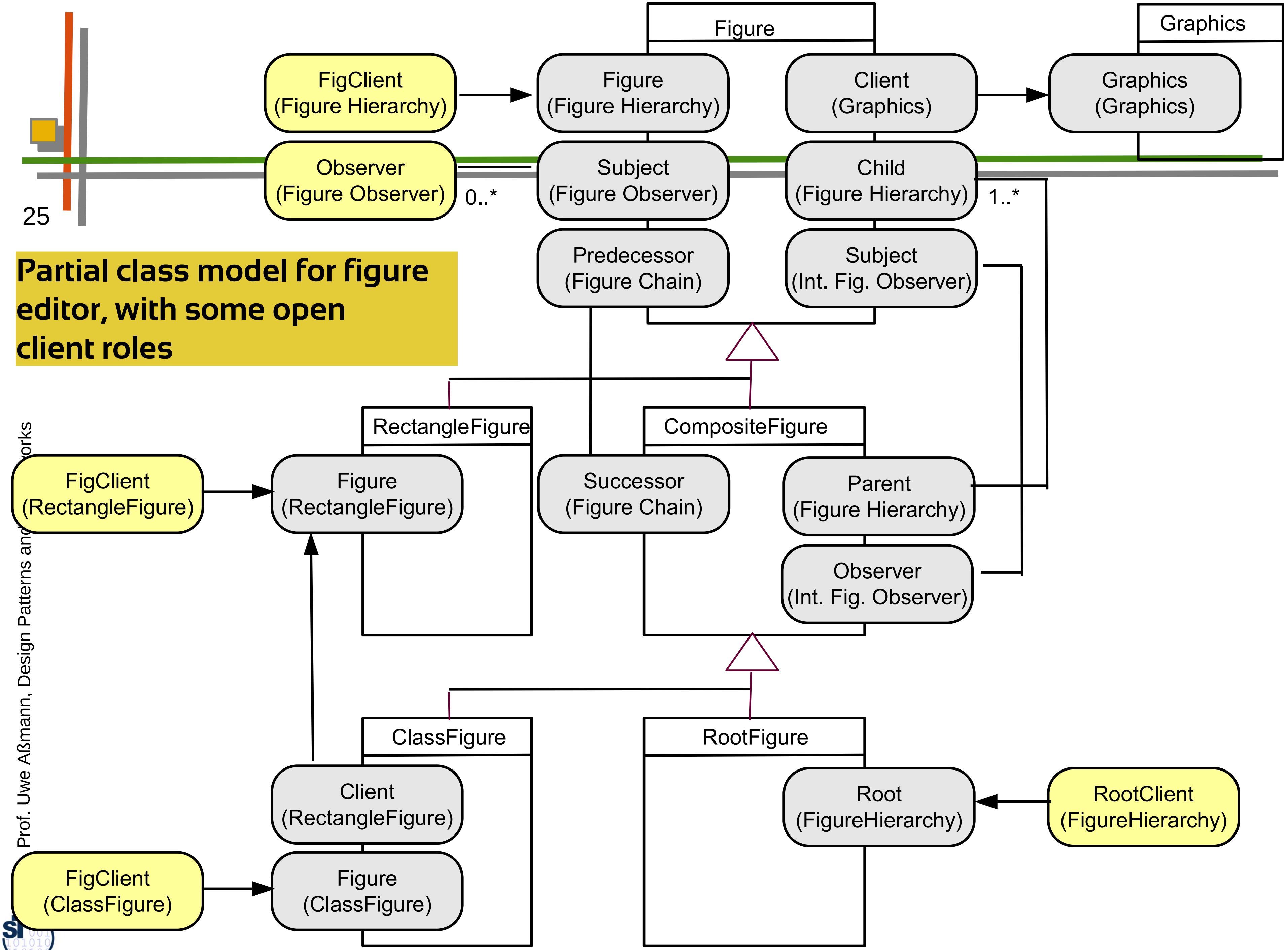
10.2.1 Merging Role Models into Class Diagrams

How role models are merged to class models

Composing Role Models To Partial Class Diagrams

- 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
  - Then not all roles are associated to classes

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



Role Models in the Example

- Composite: composite figures (with root figure and other types, such as rectangluar or class)
- Chain of Responsibility: How objects forward client requests up the hierarchy, until it can be handled
- Observer 1: Observer pattern, for callback communication among clients and figures
- Observer 2: Observer pattern, for communication among figures

10.3 Role Mapping in the MDA

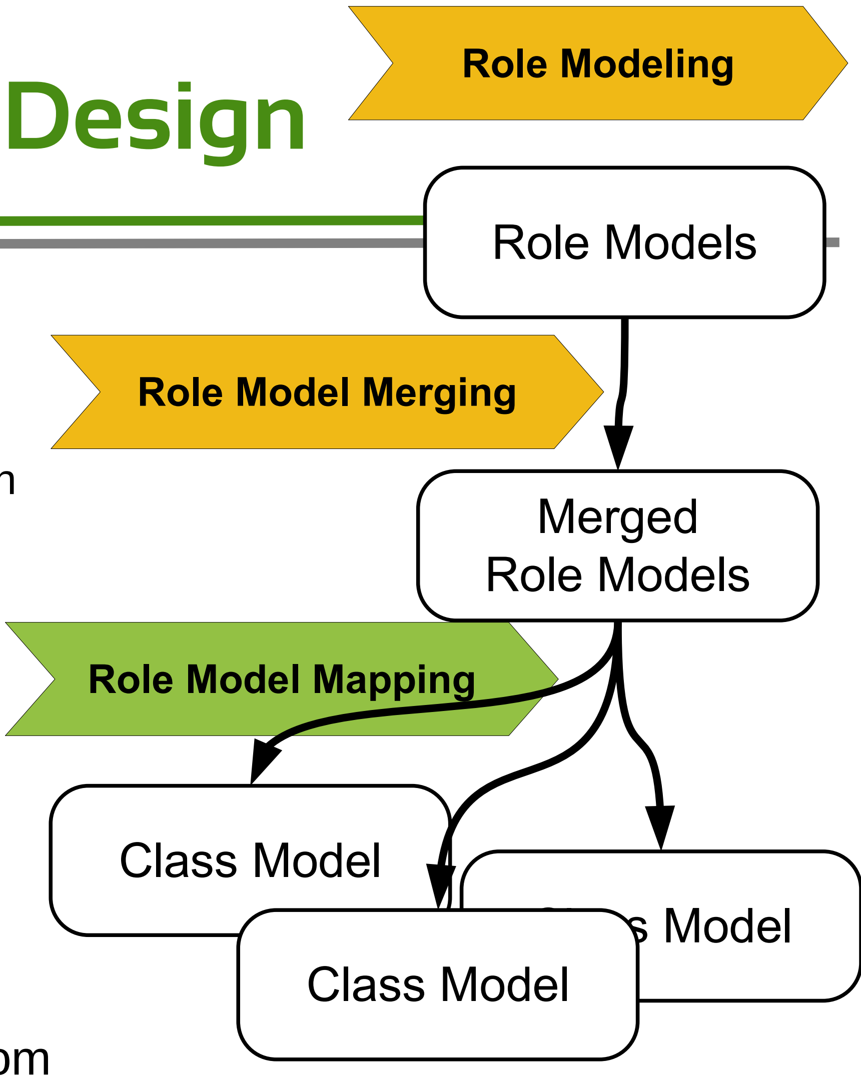
From conceptual role models to class models

Merging and mapping role models to class

models are steps of MDA [Zhao]

Steps In Role-Based Design

- 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! (variability)

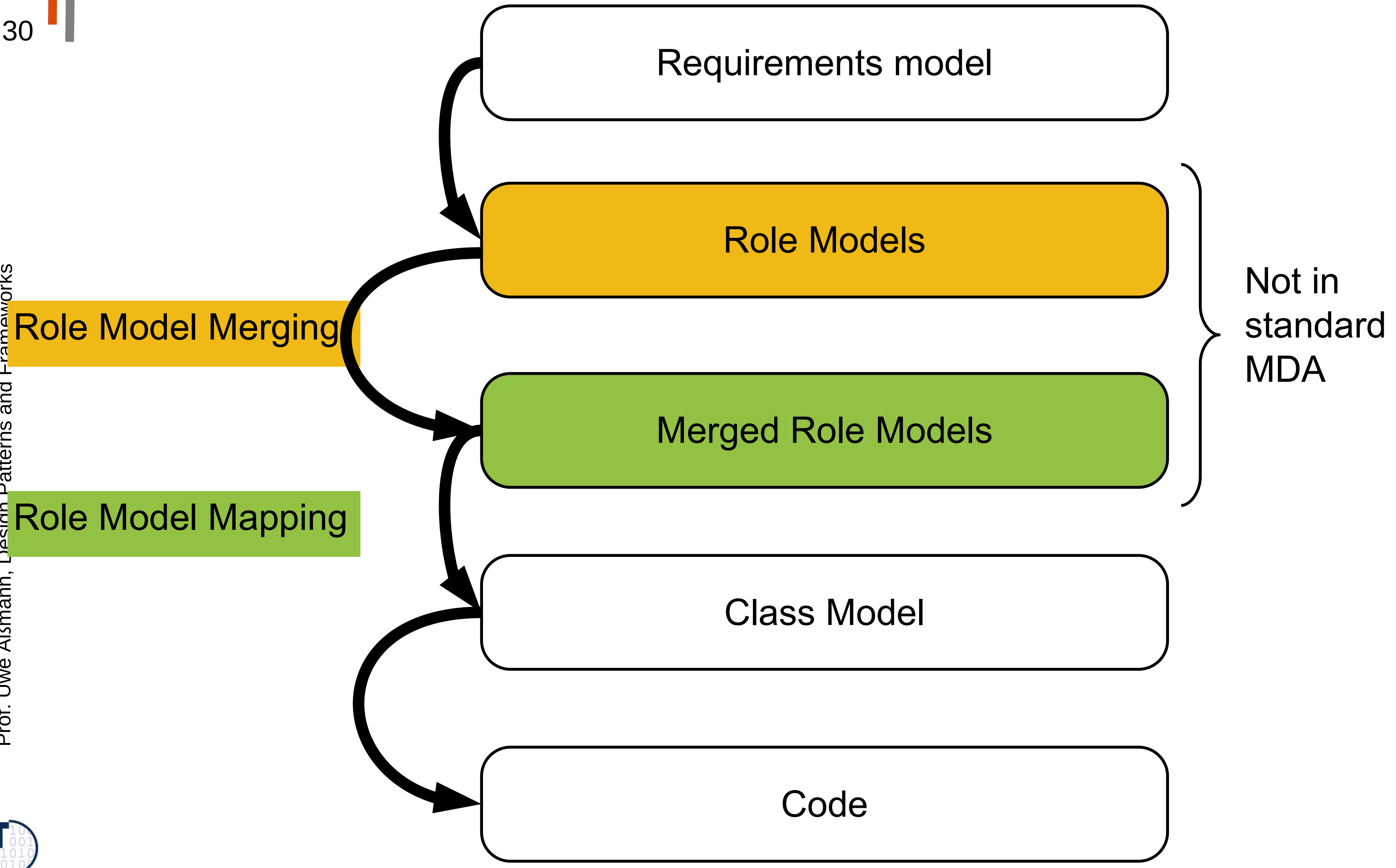


Step 1 Role modeling
Step 2 Merge
Step 3 Map

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

- 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

Role Model Mapping is a Task in MDA



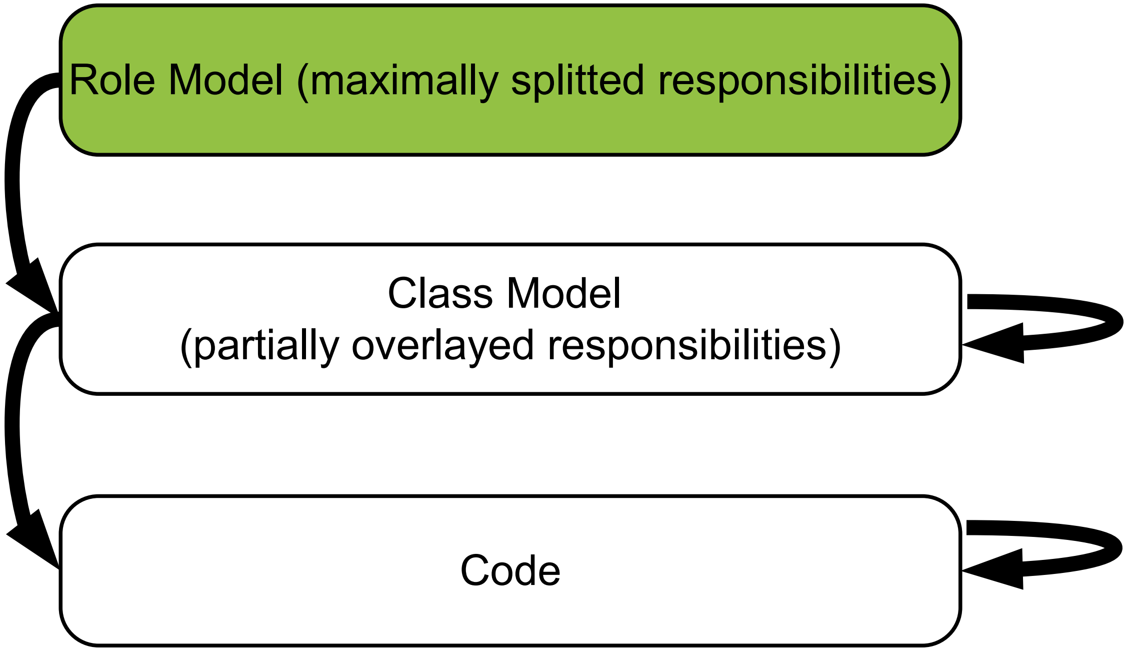
The Influence of the Role Constraints on Role Model Mapping

- *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 allows 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)

Refining Class Models by Role Mapping

- The role mapping process determines, which class inherits from which role-interface
- The role mapping *computes* the classes from maximal splits of the logical objects

Role model mapping


Role Model (maximally splitted responsibilities)
Class Model
(partially overlayed responsibilities)
Code

10.4 Implementing Roles By Hand

Implementation of Roles

Roles 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 languages allow for composing “mixin” classes into classes
    - CLOS, Scala
    - “include inheritance” (Eiffel, Sather)
- With delegation (Role Object Pattern)
- With conditional aspects

With Interfaces

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

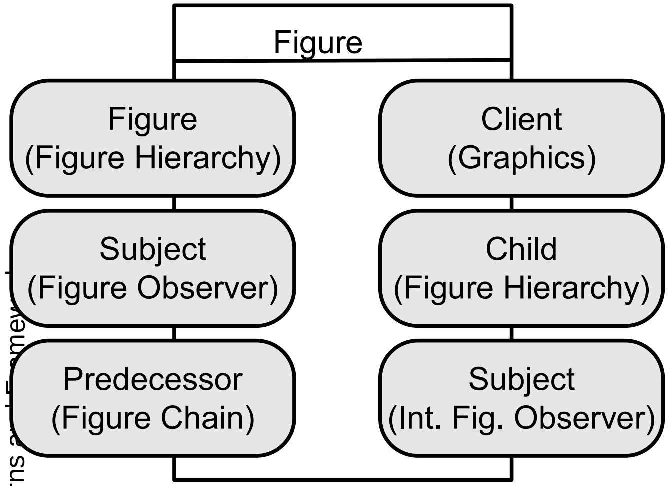
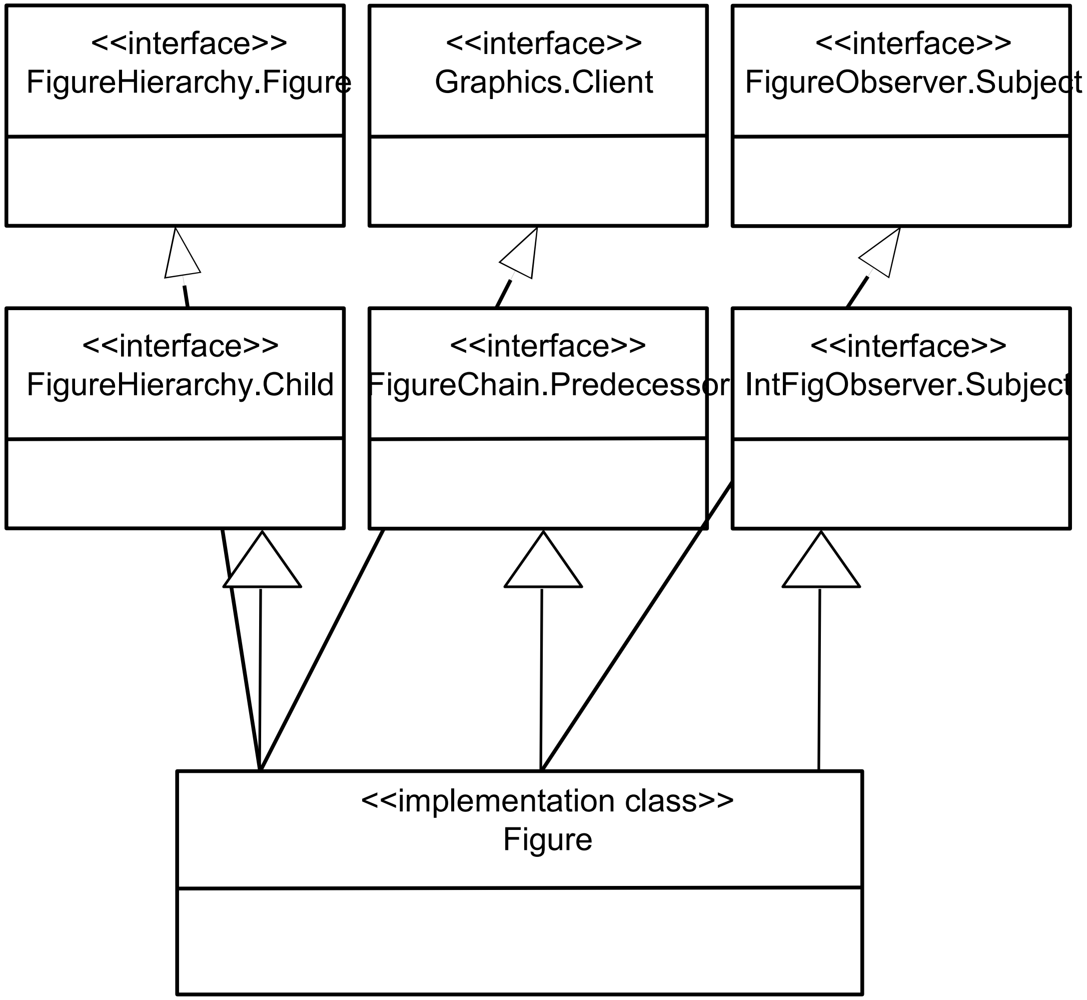


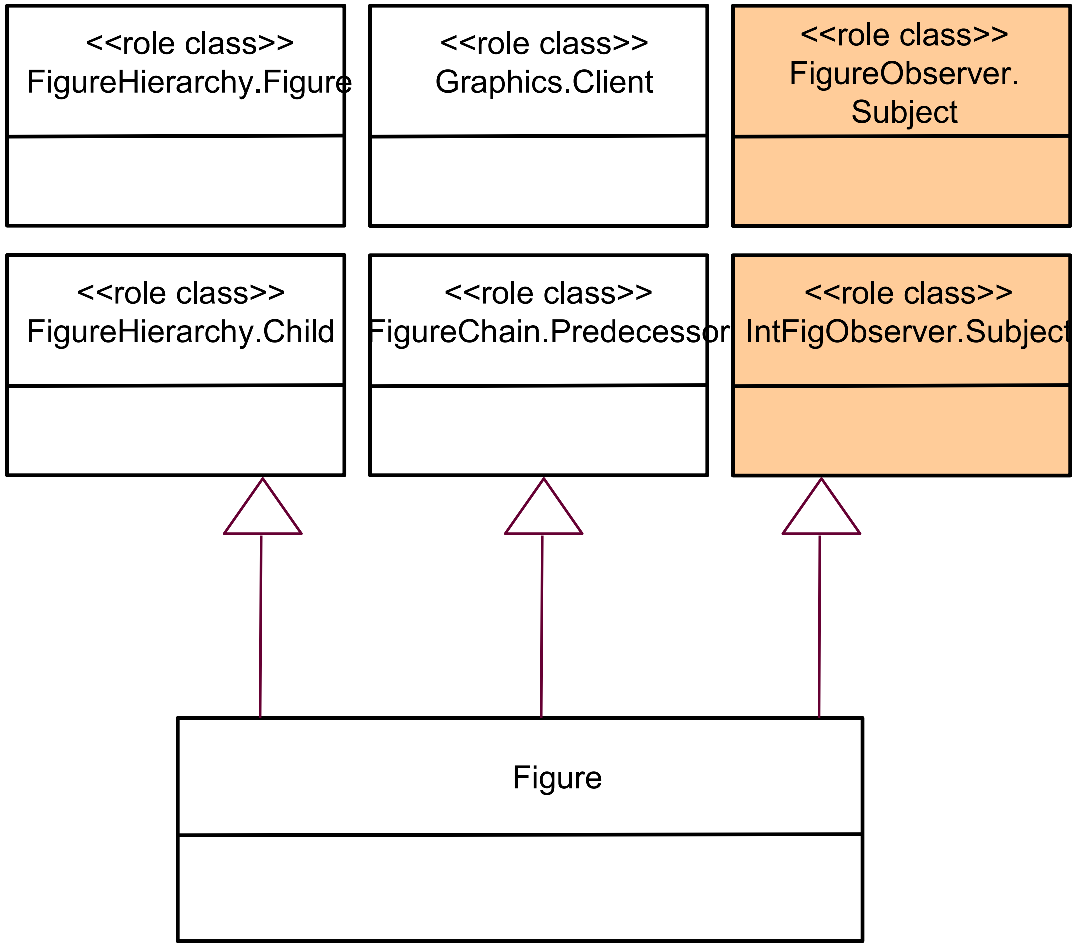
Figure
Figure (Figure Hierarchy)
Client (Graphics)
Subject (Figure Observer)
Child (Figure Hierarchy)
Predecessor (Figure Chain)
Subject (Int. Fig. Observer)


<<interface>> FigureHierarchy.Figure
<<interface>> Graphics.Client
<<interface>> FigureObserver.Subject
<<interface>> FigureHierarchy.Child
<<interface>> FigureChain.Predecessor
<<interface>> IntFigObserver.Subject
<<implementation class>> Figure

```
public class Figure implements
    FigureHierarchy.Figure,
    FigureHiearchy.Child,
    Graphics.Client,
    IntFigObserver.Subject,
    FigureObserver.Subject,
    FigureChain.Predecessor
{   ... implementations of
    role-interfaces ...
}
```

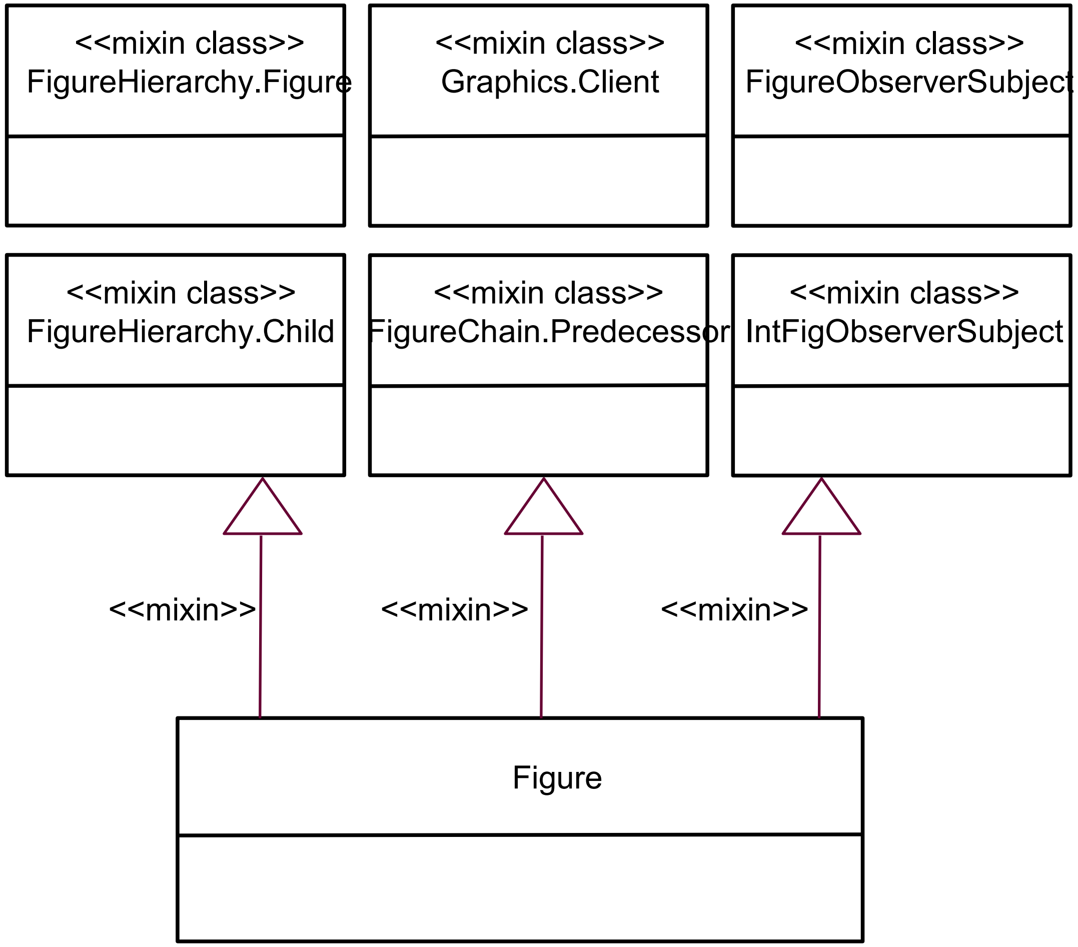
With Multiple Inheritance

- 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
  - Problem: context



With Mixin Classes

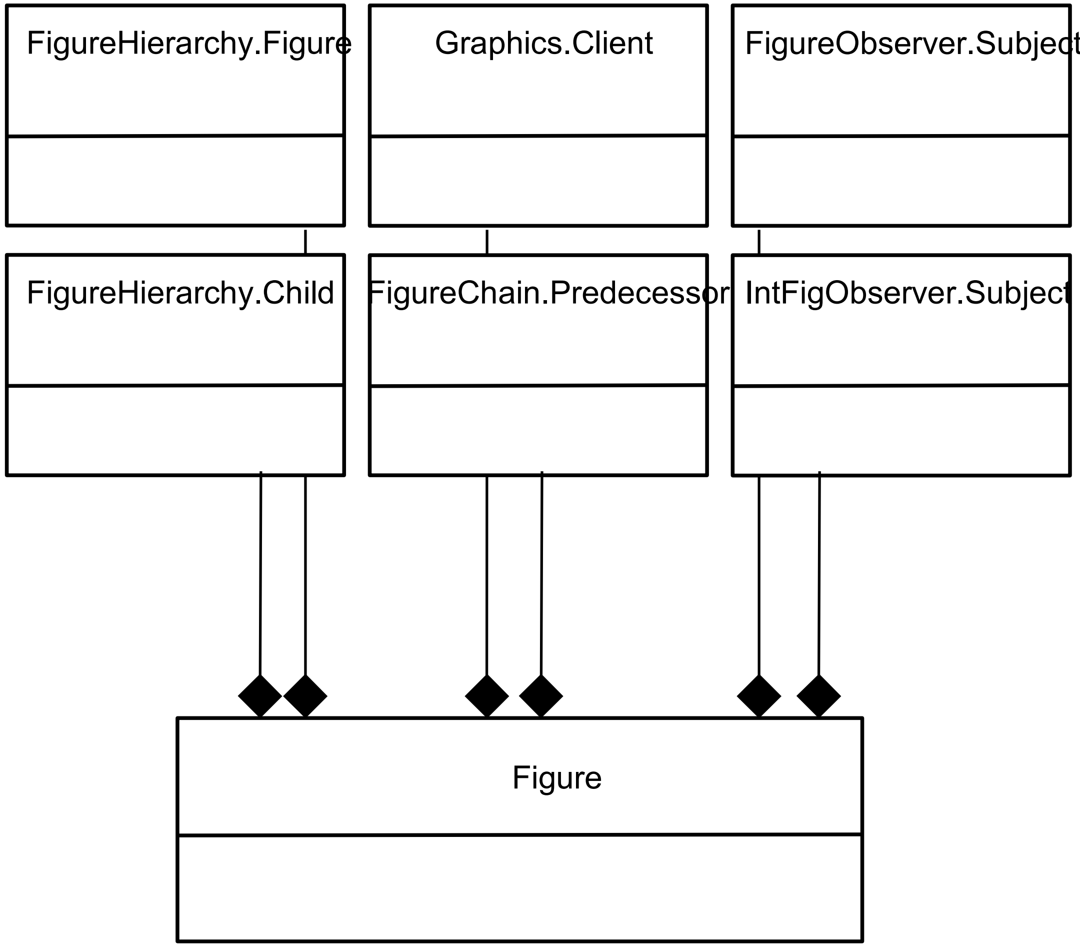
- Some languages allow for composing "mixin" classes into classes
  - CLOS, Scala
  - "include inheritance" (Eiffel, Sather)
- A mixin is a superclass parameterizing a generic super declaration of a base class
- A role type is like a mixin class
- Role code can be inherited
- Features of a mixin are renamed, if it is inherited a second time



<<mixin class>> FigureHierarchy.Figure
<<mixin class>> Graphics.Client
<<mixin class>> FigureObserverSubject
<<mixin class>> FigureHierarchy.Child
<<mixin class>> FigureChain.Predecessor
<<mixin class>> IntFigObserverSubject
<<mixin>>
<<mixin>>
<<mixin>>
Figure

Implementation With Multi-Bridges and "Role Objects"

- 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

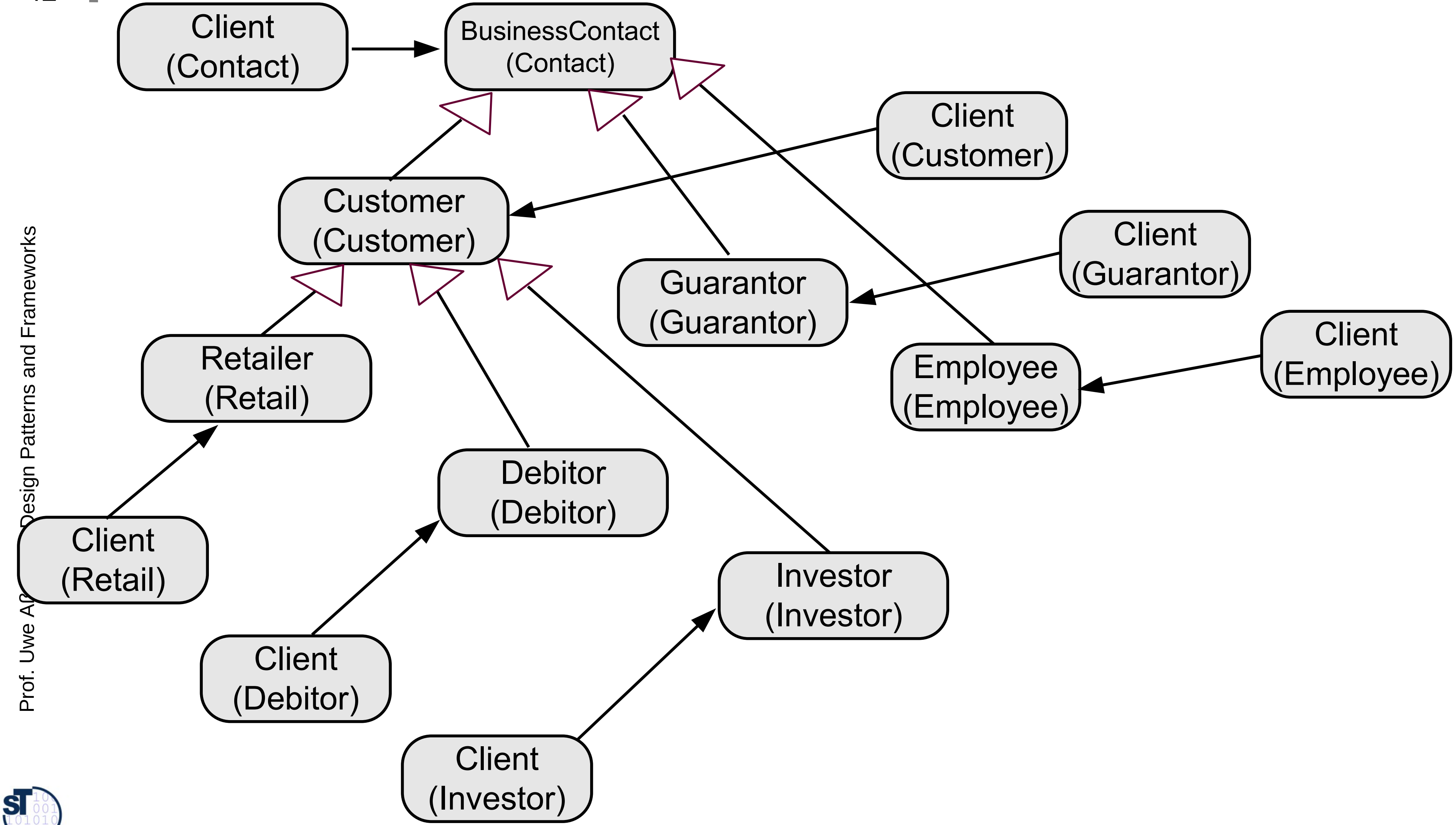
- The body of a role must be embedded into the control- and data-flow of the context code of the class.
- Wrapper/Decorator:
  - If a role is implemented as Wrapper (Decorator), it intercepts the control flow inward and outward of a method or class
  - Then, roles can be stacked at run-time (Decorator 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

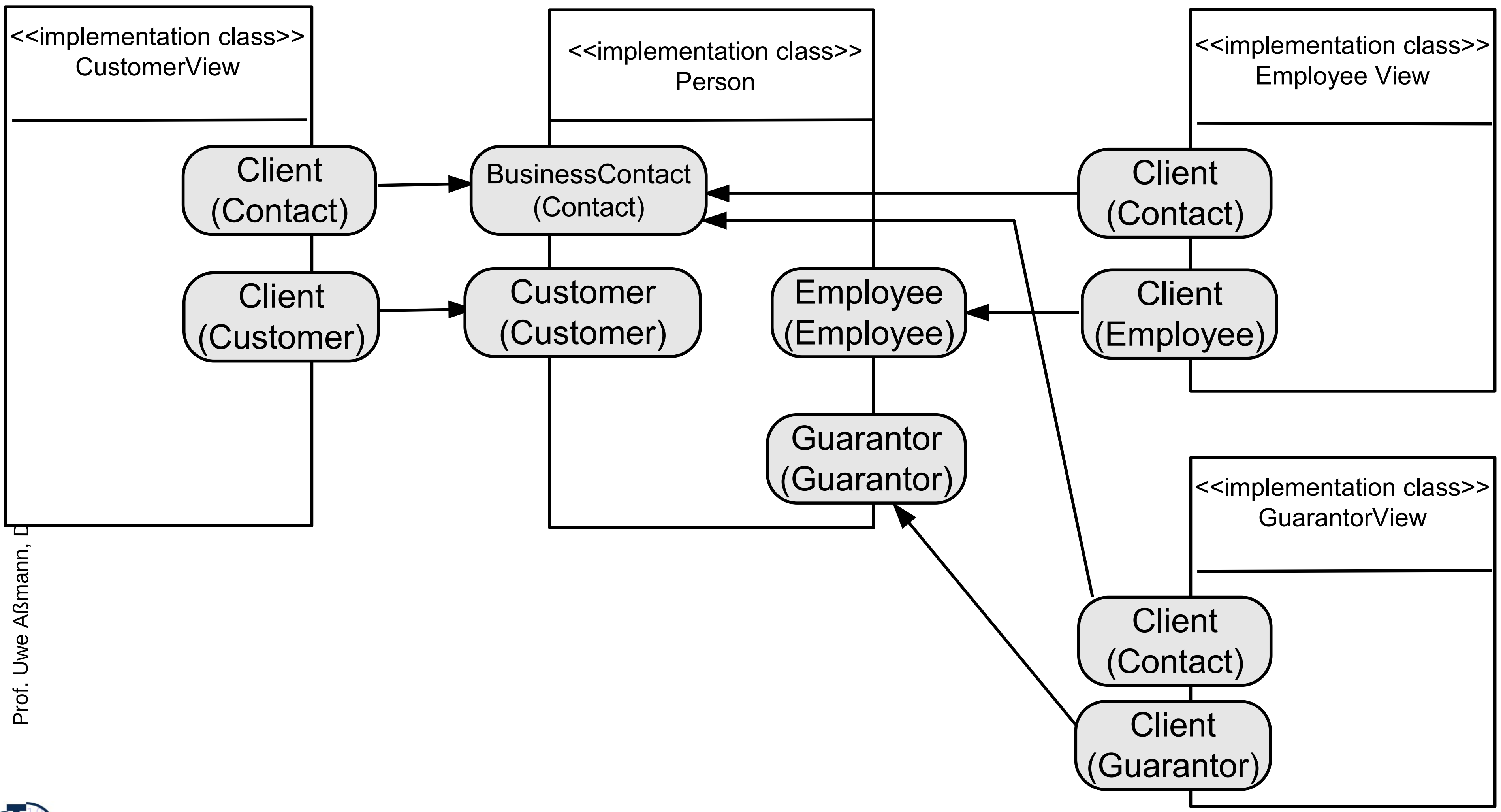
- A faceted class is a class with n dimensions
- If the facet has a collaboration partner:
  - Than the facet is a role type
  - Role types are “founded” against other role types
  - Roles are played temporarily, whereas facets are lasting

10.4.1. Example of Roles of Persons in Business Applications

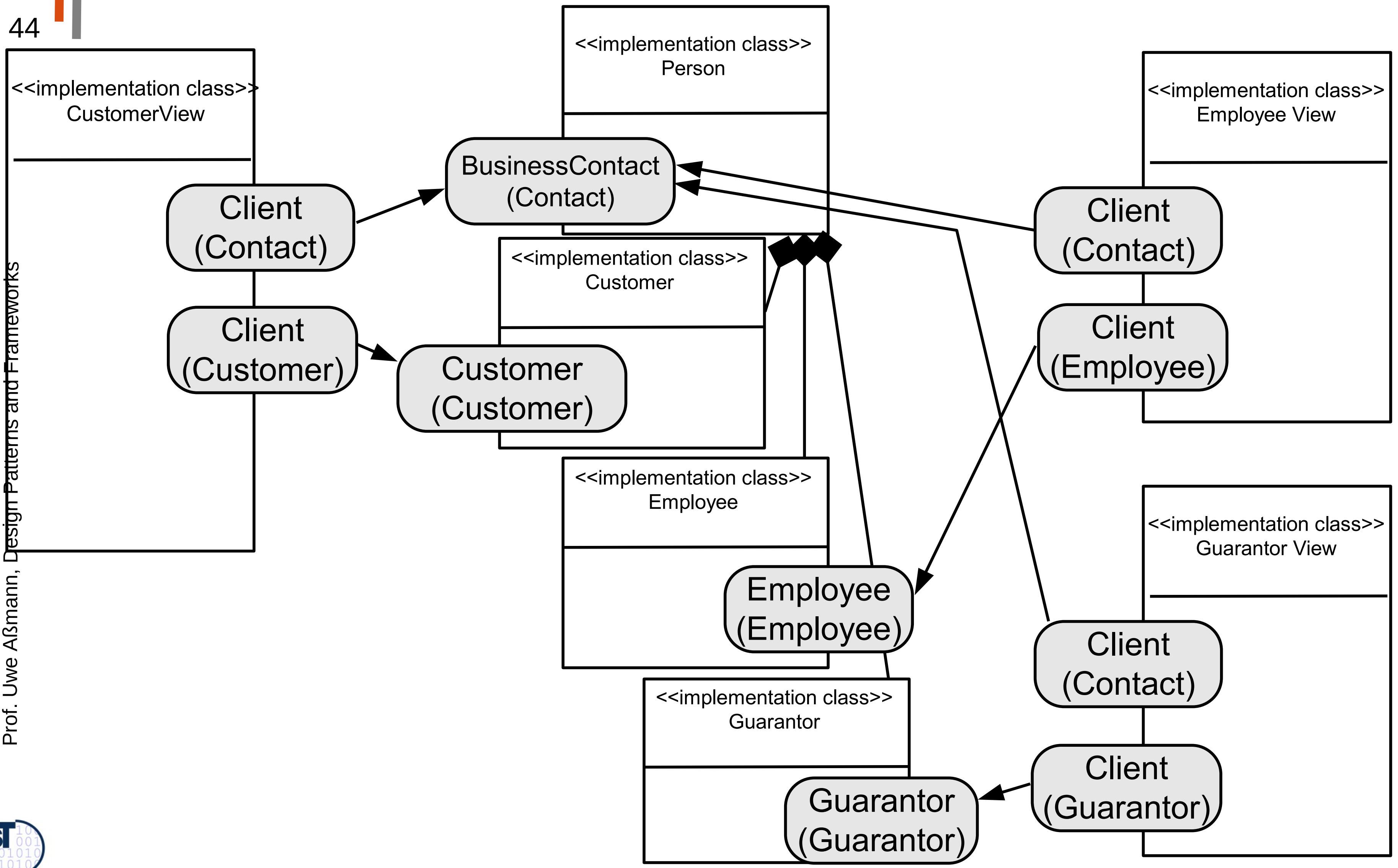
Role Models of Persons



Implementation With Interfaces (or Mixins)



Implementation of Person With Multi-Bridge (Role Objects)

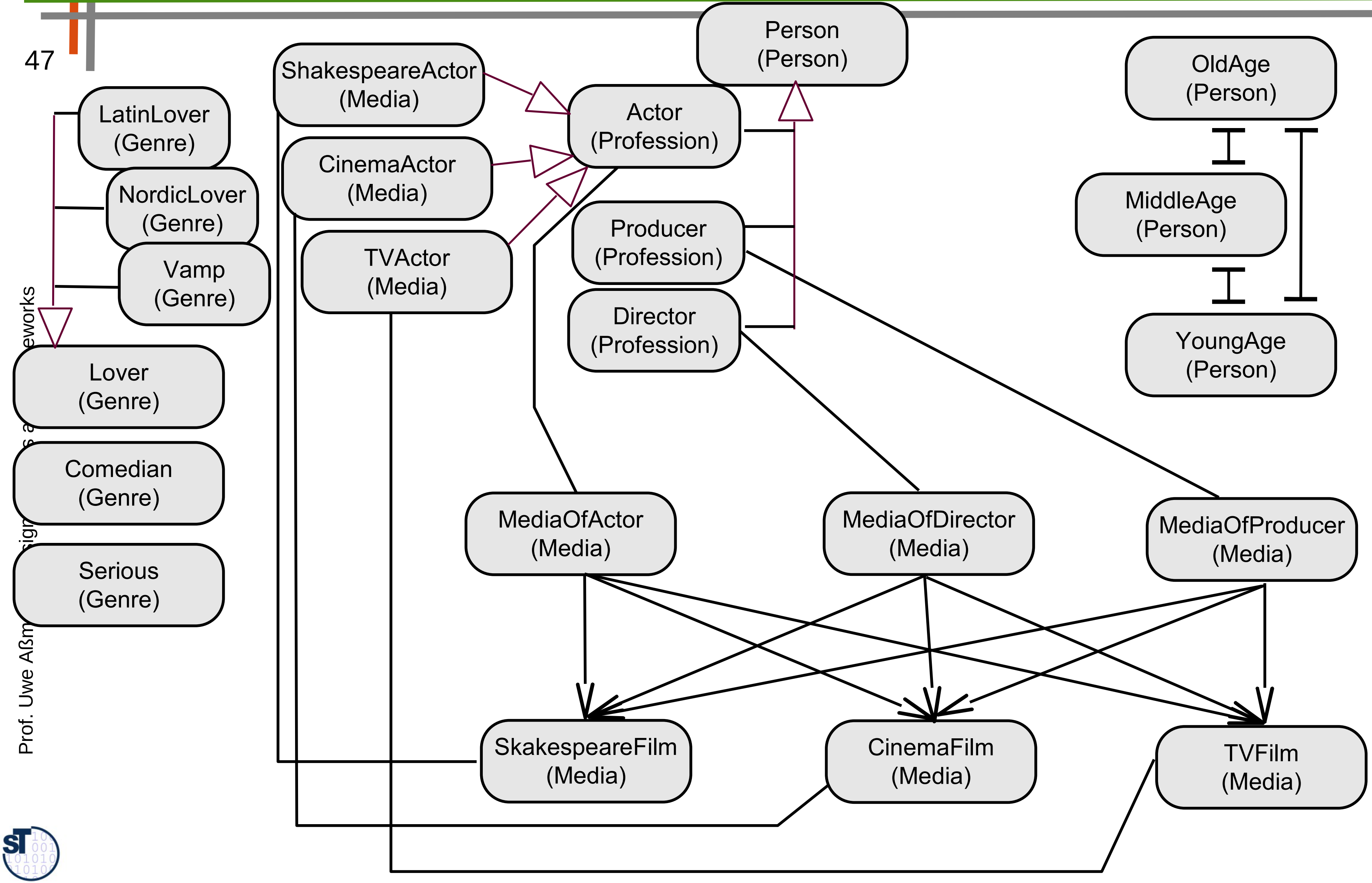


10.4.2 Example: Actors, Films, and Directors

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)

Example Role Model for Actors

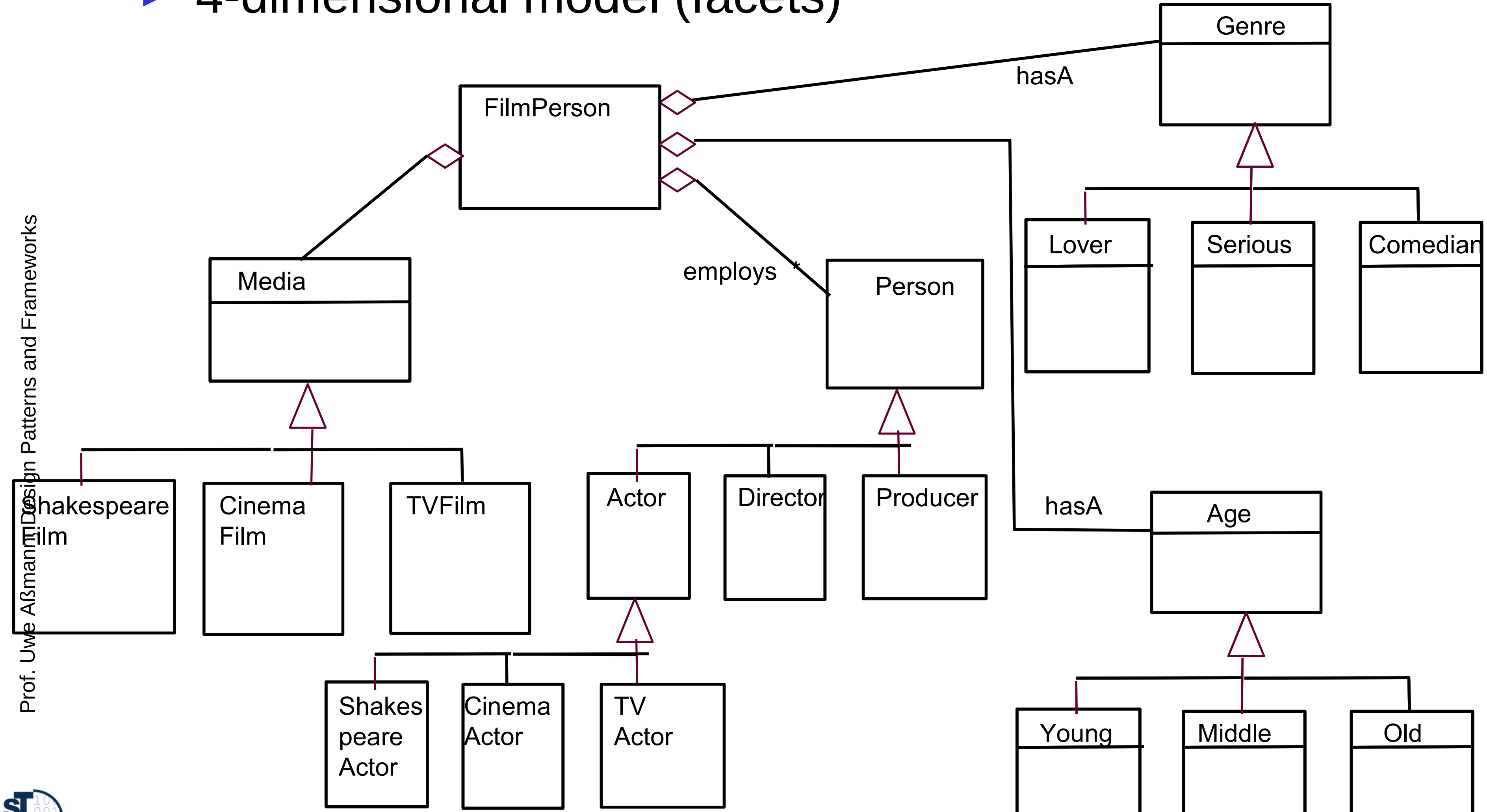


There are Many Ways to Implement This Role Model

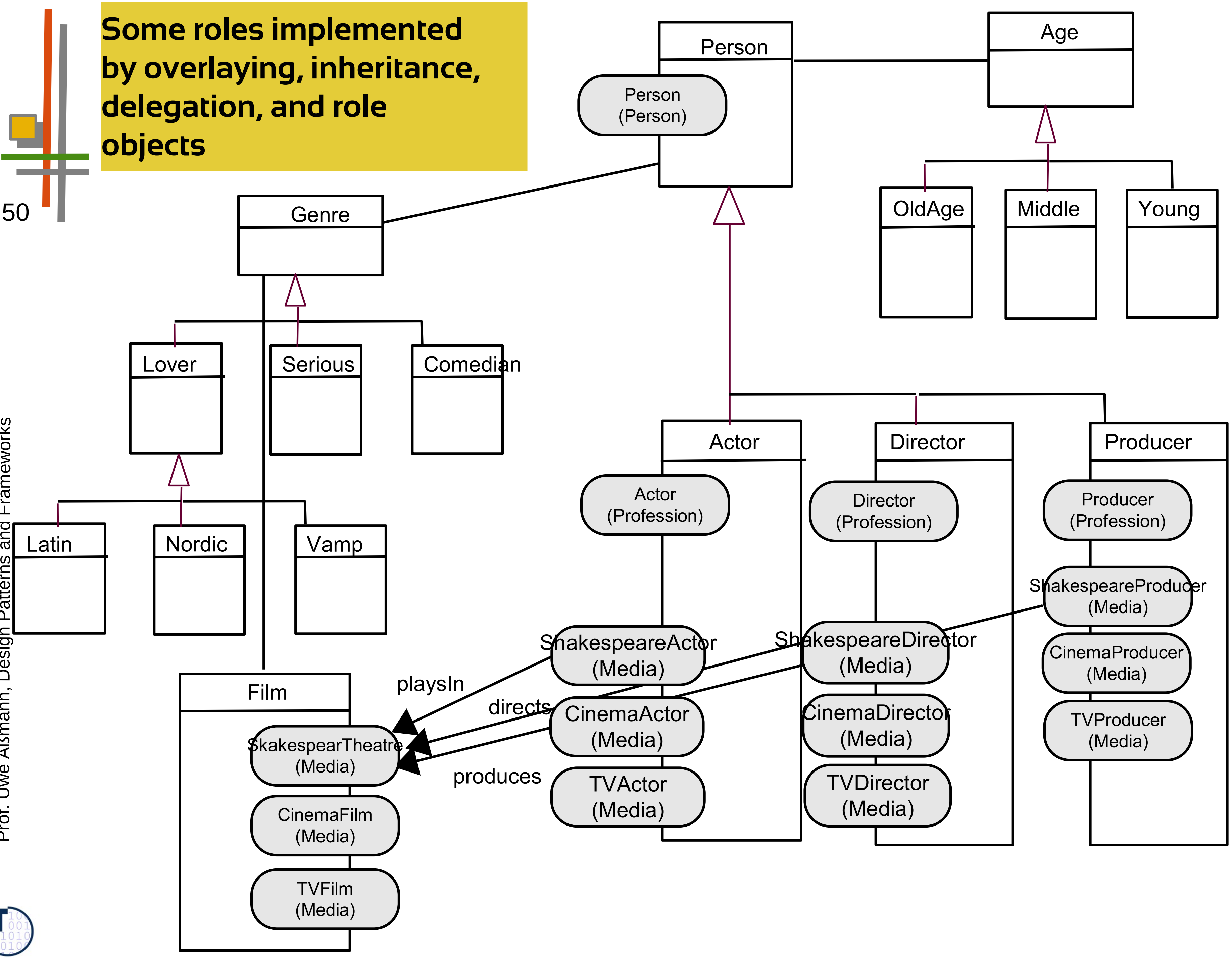
- With a facet based model, modelling some role models as class hierachies of a Dimensional Hierarchies model

Very Simple Class Model for Actors and Films

- 4-dimensional model (facets)



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



10.5 Role Types Formally

Rigid Types

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

- 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

Founded Types

- 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

- Role-based modelling is more general and finer-grained than class-based modelling
  - Focus on collaborations (i.e., tasks in context)
- 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)